Elimination of Negation in a Logical Framework

Alberto Momigliano

December 15, 2000

CMU-CS-00-175

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Thesis Committee:
Frank Pfenning, Chair
Dana Scott
Dale Miller, Pennsylvania State University

This research was supported in part by the the National Science Foundation under grants CCR-9619584 and CCR-9988281 (Principal Investigator: Frank Pfenning). The views and conclusions contained in this document are those of the the author and should not be interpreted as representing the official policies, either expressed or implied, of NFS or the U.S. Goverment

Keywords: logical frameworks, higher-order logic programming, negation, negation-as-failure, strict λ-calculus, regular world assumption

Abstract

We address the issue of endowing a logical framework with a logically justified notion of negation. Logical frameworks with a logic programming interpretation such as hereditary Harrop formulae cannot directly express negative information, although negation is a useful specification tool. Since negation-as-failure does not fit well in a logical framework, especially one endowed with hypothetical and parametric judgments, we adapt the idea of *elimination* of negation from Horn logic to a fragment of higher-order hereditary Harrop formulae. The idea is to replace occurrences of negative predicates with positive ones which are operationally equivalent. This entails two separate phases.

Complementing terms, i.e. in our case higher-order patterns. Due the presence of *partially applied* lambda terms, intuitionistic lambda calculi are not closed under complementation. We thus develop a strict lambda calculus, where we can directly express whether a function depends on its argument.

Complementing clauses. This can be seen as a negation normal form procedure which is consistent with intuitionistic provability. It entails finding a middle ground between the Closed World Assumption usually associated with negation and the Open World Assumption typical of logical frameworks. As this is in general not possible, we restrict ourselves to a fragment in which clause complementation is viable and that has proven to be expressive enough for the practice of logical frameworks. The main technical idea is to isolate a set of programs where static and dynamic clauses do not overlap.

Acknowledgements

This dissertation would not have been possible without the help and support of many different people. First and foremost my thesis advisor, Frank Pfenning, who has guided me through every step of this research, from the high level design decisions down to the most technical details. There is probably not a single line that does not carry his imprint. Of course, any remaining mistake is solely my responsibility. In the process, he has taught me a rigorous discipline and a style in approaching formal problems that has made me a far better researcher and perhaps a better person, too.

I would also like to thank wholeheartedly the members of my dissertation committee, Dale Miller and Dana Scott for their useful comments and encouragement.

I am also deeply grateful to Carsten Schürmann, who, under the motto "What do you want to prove today?" has always been ready and willing to discuss a wide range of issues from from type-theory to the practicalities of the Twelf system. I'm also indebted to Roberto Virga for many discussions and his precious help in debugging the strict calculus rules. Thanks to Iliano Cervesato for his comments to an early version of the material in Chapter 3 and for making available his manuscript [Cer].

I'd also like to thank Mario Ornaghi for his friendship and hospitality at DSI, Milan and the many discussions on the essence of logic programming. Thanks also to Ugo Moscato for his support at DSI.

In the health department, I want to thanks from the deep of my heart Dr. Garfinkel, Dr. Velletri, Dr. Oliva, Dr. Zigler, Jean Bender and the wonderful people at GAV.

I wouldn't have been able to survive all those years in Pittsburgh without the company of the local Italian community and associates: Barbara and Jerry, Cristina and Massimo, Violetta and Ben, Cristina and Dude, Ulrike and Massimo, Stefano and Kathy, Giovanna and Aldo. A particular thanks to Lena.

Finally, I want to thank my beloved Claudia, who has been waiting for me on the other part of the ocean, so very patient and affectionate during those hard and long years of separation.

It is so sad that my mother is not here anymore to witness the happy conclusion of such a difficult journey. All I can do is dedicate this dissertation to her memory.

Contents

List of Figures

Chapter 1

Introduction

Suppose we are giving a formal definition of a programming language in the style of natural semantics [Kah87]; after we have specified the abstract syntax, the type system ($e : \tau$) and a small step evaluation semantics ($e \mapsto e'$), it is time to check their consistency via a proof of type soundness. We may start by attempting the *Progress* lemma, which will eventually guarantee that well-typed expressions cannot go wrong; in symbols, this could go like this: for every expressions e such that $e : \tau$ and e is *not* a value, there exists an expression e' such that $e \mapsto e'$. Now, imagine that our language is fairly sophisticated and our funding agency requires a machine-checkable verification of our results. Thus, we decide to use an automated reasoning tool, possibly an interactive one. A proof of progress is a rather trivial structural induction for a human, but in order to be machine checked, it needs to be spelled out in every detail. We may have implemented judgments defining expressions, values, typing and evaluation: but what about the notion of *not* being a value? We may try to reason in a strictly intuitionistic way and view $\neg value(e)$ as the derivation of a contradiction $\perp$ from the assumption $value(e)$. This is possible, but certainly not in the spirit of the proof; $\neg value(e)$ is just a test, a way to sift out expressions that are already fully evaluated. What we really need is a *positive* (inductive) definition of "not being a value", say $nonvalue(e)$; indeed, this should be possible, since non-values are exactly those expressions which are not values. Nevertheless, manually coding this notion may be tedious and error-prone, especially considering evolution of our initial specification. Moreover, we will also have the obligation of *proving*, at least to our satisfaction, that the explicit definition of 'non-value' coincides with the negation of 'value'. Lacking this, our formal verification cannot be entirely trusted.

For another example consider a simple instance of reasoning about process algebra, such as Peterson's algorithm for mutual exclusion [Pet81]. The problem here is to ensure that two processes can never be simultaneously in their critical section. A process can be in several (possibly many) states, such as `sleeping, trying, critical`; a transition relation describes how the system moves from a state to another, according to whether a process is allowed to change its status. Suppose we want to verify some property of the system such as *safety*: for any possible sequence of transitions if the initial state is *safe*, so is the final one. Now, from the description of the problem, it is apparent that a state is safe if both process are *not* in their critical section. It would benefit the verification attempt to have a *positive* explicit specification of being a safe state, rather than working with an implicit negative one. Not only the number of states can be fairly large, but consider the natural extension of the same problem to n-processes: a hand-written positive specification of a state being safe can be incomplete or plain wrong. Again, over time, the number of states will increase or possibly decrease and the safety specification needs to evolve accordingly.

The bottom line is that negation is a very common connective in a specification – and rightly so, since it is one of the most basic logic operators. Formalizations that use negation are often sharper and more concise. Nevertheless, not every automated reasoning tool available nowadays is able to provide an appropriate handling of this connective. This is particularly problematic for *logical frameworks* based on higher-order logic or type-theory with a logic programming interpretation, such as *Twelf* [SP98] and λProlog [NM88, Mil89b]. While the latter provide a very advanced unified environment for the specification, implementation and verification of deductive systems, they inherit the traditional problems with negation, which Prolog has struggled with since its inception. These problems are further augmented by some of their more beneficial features; namely by being 'higher-order' and being based on intuitionistic provability. Those characteristics are the main key elements of the success of those frameworks and should be preserved under every extension. This dissertation presents an approach to endow those languages with a logically sound notion of negation without sacrifying any part of their representation power.

1.1 Logical Frameworks

A *logical framework* [Pfe00a] is a meta-language for the specification, implementation and verification of deductive systems and their meta-theory. Deductive systems consist of axioms and rules defining derivable judgments; they can be used to specify logics and aspects of programming languages such as operational semantics, type systems, abstract machines and compilation.

Logical frameworks offer a bridge between the success of declarative programming languages (logic and functional) and the unsatisfactory results of general theorem proving. There is perhaps a reasonable middle way between Poincare's derision to the logicist approach:

> "If you need twenty-seven equations to prove that 1 is a natural number, how many will you need to prove a real theorem?"[1].

and Wos' claim to have solved with OTTER real mathematical open questions (see [WM91] for a depressing list).

Many logical framework have been proposed in the literature (see [Pfe00a] for an overview) and many extensions are also under consideration. However, we must carefully balance the benefits that any proposed extension can bring against the complications its meta-theory would incur. We have two main issues to consider:

1. It is been argued that logical frameworks should be by design as weak as possible [dB91], in order to:
   - Simplify proofs of adequacy of encodings.
   - Allow effective checking of the validity of derivations.
   - Reduce the complexity of proof-search.
   - Inherit a treatable unification problem.
2. At the same time logical frameworks must provide powerful tools to support the design process of deductive systems. Experience has shown that the strength of a logical framework is proportional to the ease it makes encodings simple and concise. The more direct is the encoding, the easier is to reason about it. One well known example is higher-order abstract syntax [PE88], which moves renaming and substitution principles to the meta-language; this avoids the explicit programming and proving of a large series of low-level results about those trivial but ubiquitous concepts. Another example is the reification of derivations as proof terms in type-theoretic languages, which reduces run-time check for correctness of derivations to type-checking in the meta-language.

The approach taken in the *Twelf* project is a "pay as you go" one. In other words, every extension is carefully crafted so as to be *conservative* on the operational and declarative semantics of the core language. Examples are the *linear* extension [CP96] or refinement types [Pfe93].

1.2 Negation

The aim of this thesis is to develop a framework for the synthesis of the negation of logic programs in logical frameworks such as hereditary Harrop formulae (HHF) [MNPS91] and its implementation in λProlog [NM88]. We intend this to set the basis for type-theoretic frameworks such as LF [HHP93] and its implementation *Twelf* [SP98] and possibly their linear refinement as Lolli [HM94] and *LLF* [CP96]. This approach could

[1] Les dernieres efforts des logiciens, in Science et Methode, p. 193.

also be useful for (inuitionistic) generic theorem proving systems, especially ones based on higher-order logic or type theory such as Isabelle [Isa98] and Coq[DFH$^+$93].

Those systems (Isabelle and Coq excluded) do not provide a *primitive* negation operator. Indeed, constructive logics usually implement negative information as $\neg A \equiv A \rightarrow \bot$, where $\bot$ denotes absurdity and the Duns Scoto Law is the elimination rule. Thus negative predicates have no special status; that would correspond to explicitly coding negative information in a program, which is entirely consistent with the procedural interpretation of hypothetical judgments available in logical frameworks with a logic programming interpretation. However, this would not only significantly complicate goal-oriented proof search (as it is manifested in the difficulty of implementing, for example, the full logic of *Forum* [Mil94]), but providing negative definitions seems to be particularly error-prone, repetitive and not particularly interesting; more importantly, in a logical framework we have also to fulfill the *proof obligation* that the proposed negative definition does behave as the complement (of its positive counterpart).

Providing a viable negation operator has an immediate practical relevance in programming in those languages, since it relieves the user from the burden of explicitly encoding negative information in the form of clauses which express the condition for a predicate not to hold. Automating the synthesis of negative information has not only a clear benefit in the logic programming sense, but it may also have a rather dramatic effect on the possibility of implementing deductive systems that would prove to be too unwieldy to deal with otherwise. The synthesis of the negation of predicates such as *typable, well-formed, canonical form, subsort, value* etc.–as well as Prolog-like predicates such as equality, set membership and the like–will increase the amount of meta-theory that can be formalized.

Of course, the addition of negation does not change the recursion-theoretic expressive power of a language, but we claim that it does make a difference at the representation level. To bring this to the extreme, deductive systems can be expressed eventually in, say, first-order Horn logic and ideally proved-checked or even demonstrated by a resolution theorem prover or more likely by an interactive one. In practice, this has turned out to be very problematic, if not a total failure; hence the refinement of the tools to higher-order logic and type-theory.

Traditionally, negation-as-failure (*NF*) [Cla78] has been the overwhelmingly used approach in logic programming (see [AB94] for a recent survey): that is, infer $\neg A$ if every proof of A fails finitely. The operational nature of this rule and its ultimately troublesome logical status is a serious threat to the logical frameworks' endeavor. We will return on the topic of why *NF* is an absolutely inadequate way to address the issues of negation in a such a framework in Section 1.4.

While the topic of negation has been pursued to the extreme in first-order logic programming (we shall try a small review of closely related approaches in Section 1.3), the field is almost virgin as far as higher-order logic and type theory is concerned: languages such as λProlog implement *NF* with the usual cut-fail combination: a logical reconstruction for the first order fragment has been attempted, with somewhat disappointing results, in Harland's thesis [Har91b].

Though the impetus of this enterprise may seem at first sight mainly pragmatic, it should not be underrated. In short, we are trying to design a reasonable notion of negation, a basic building block of any logic under severe computational constraints:

> "The problem is difficult because it seeks a notion of negation which is simultaneously semantically elegant and computationally feasible: in both execution and mathematical/logical semantics the extended language should cleanly extend the definite clause language" [JLLM91].

The reason why *NF* is so popular in the logic programming paradigm is that it essentially requires no modification to the search structure of an logic programming interpreter. The real question is whether it also satisfies the other aforementioned criteria. Nonetheless this is just a part of it:

> "... this notion [*NF*] is a basic logical notion, a notion of value to pure logic (as studied since the Ancient Greeks) of equal importance and theoretical standing as notion like Possibility, Deduction, Axiom and the like. The role of negation by failure in logic programming in only a special case: one manifestation of its role in logic" [Gab91].

Our answer to this plea will be to show that, paradoxically, the best way to deal with negation in the logic programming setting is to eliminate it through transformation.

It is a basic fact of classical propositional logic that connectives are inter-definable; more precisely, a sufficiently expressive set of sentential operators can provide, by definition, the missing ones, as an immediate consequence of their truth-value semantics. It is therefore customary and economically convenient to assume as primitive only this basis and define the others operators in terms of the former. In almost every definition negation is taken as primitive, paired either with conjunction, disjunction or implication; even those more succinct presentations based on a singleton connective as *nand* retain an implicit flavor of negation.

There is yet another way to address negation which is related to the transformational approach we are interested in. This is known as *negation normal form* and it is used for example in Tait's concise proof of cut-elimination for classical logic. For every atomic predicate symbol p we also have a symbol denoting the opposite, say $\overline{p}$. Then, with the essential usage of double negation elimination and De Morgan's laws, negation is *defined* as follows:

$$\begin{aligned}
\neg p &\stackrel{\text{def}}{=} \overline{p} \\
\neg \overline{p} &\stackrel{\text{def}}{=} p \\
\neg\neg p &\stackrel{\text{def}}{=} p \\
\neg(A \wedge B) &\stackrel{\text{def}}{=} \neg A \vee \neg B \\
\neg(A \vee B) &\stackrel{\text{def}}{=} \neg A \wedge \neg B
\end{aligned}$$

Thus, as far as the classical propositional structure is concerned, negation can be accomplished simply by renaming. Consistency is then achieved by adding axioms that specifies that is it inconsistent to hold both p and $\overline{p}$, namely $p \leftrightarrow \neg\overline{p}$ an $\neg p \leftrightarrow \overline{p}$.

This is another way to look at the approach to negation that we shall investigate, that is the *transformational* one, also known as *intensional negation*, initiated in [ST84] and developed in Pisa for Horn logic [BMPT87, BMPT90, BLLM94]. Roughly, given a clause with occurrences of negated predicates, say $Q \leftarrow G, \neg P, G'$, where P is an already defined atom, the aim is to derive a *positive* predicate, say non_P, which implements the complement of P, preserving operational equivalence; then, it is merely a question of replacement, yielding the negation-less clause $Q \leftarrow G, non\_P, G'$. This has the neat effect that negation and its problems are *eliminated*, i.e. we avoid any extension to the (meta) language. Technically, we can achieve this by transforming a the body of the iff-completion [Cla78] of a Horn program into negation normal form and then by negating atoms via complementing terms, a problem first addressed in [LM87] for first-order terms. To mention the simplest example possible, suppose we have a procedure `p` that calls somewhere a check for a number not to be `even`, where the latter is already defined:

$$p(X) \leftarrow \ldots \neg even(X) \ldots$$

$$\begin{aligned}
&even(0).\\
&even(s(s(Y))) \leftarrow even(Y).
\end{aligned}$$

The goal is to obtain a definition for `p`, where the negative occurrence of `even(X)` is replaced by a positive call to its complement, say `non_even(X)`. This involves the *synthesis* of the `non_even` predicate from its positive definition:

$$p(X) \leftarrow \ldots non\_even(X) \ldots$$

$$\begin{aligned}
&non_even(s(0)).\\
&non_even(s(s(Y))) \leftarrow non_even(Y).
\end{aligned}$$

Thus where is our contribution? The problem is that this does not carry immediately over to every computational logics, where the notion of negation normal form may be in itself problematic. The issue was not apparent in the existing literature because of the identification of logic programming with Horn programming. For accident or necessity (though we now lean for the former) Horn logic imposed itself as *the* format in logic programming. Because of its restricted syntax classical and intuitionistic provability coincide in this fragment. This entails that classical equivalences preserve the intended operational semantics of the source program. Thus negation normal forms do work here, as we explain in Chapter 5

Nevertheless, this approach does not scale immediately to more expressive languages. Once we go beyond Horn logic, the intuitionistic (or 'search-like') interpretation becomes crucial to ensure the existence of what is commonly agreed as a reasonable interpreter for a logic programming language. Our endeavor can be paraphrased as the search for a notion of negation normal form for a significantly fragment of higher-order intuitionistic logic which is compatible with a logic programming interpretation.

It must be remarked that the issue of negation in constructivism is by no means new, but it has been considered by many problematic. One sticky point lies in the Heyting semantics of $\neg A$, seen as a short for $A \to \bot$; many have expressed doubts about the epistemological status of a construction which yields the absurdum. A discussion can be found in Wansing's monography [Wan93]. Already in its textbook [Hey56] Heyting mentions Griss' attempt to formalize a notion of negation-less mathematics. The most well-known approach to marry a first-class notion of negation with constructivism is Nelson's *strong negation* [Nel49]. As we will argue in Section 5.3, the interaction between strong negation and implication is inadequate to support the operational interpretation of HHF we are interested in.

Additionally, elimination of negation does not scale immediately to logical frameworks such as HHF, for two other reasons:

1. The simply-typed λ-calculus is not closed under term complement.
2. There is an intrinsic tension between the *Closed World Assumption* (CWA) [Rei78], which is associated with negation, and the *Open World Assumption* (OWA) typical of languages with embedded implication.

Differently from the first-order case, the complement of a lambda term cannot, in general, be described by a pattern, or even by a *finite* set of patterns. We can isolate one basic difficulty: a pattern such as $\lambda x.\ E\ x$ for an existential variable E matches any term of appropriate type, while $\lambda x.\ E$ matches precisely those terms $\lambda x.\ M$ where M does not depend on x. The complement then consists of all terms $\lambda x.\ M$ such that M *does* depend on x. However, this set cannot be described by a pattern, or even a finite set of patterns. This formulation of the problem suggests that we should consider a calculus with an internal notion of *strictness* so that we can directly express that a term must depend on a given variable. We will therefore introduce a *strict* λ-calculus where term complement in the simply typed λ-calculus can be embedded and performed.

The second issue is rooted again in the fundamental difference between Horn and HHF formulae: as well known, a Horn predicate definition can be seen as an inductive definition of the same predicate. The *minimality* condition of inductive definitions excludes anything else which is not allowed by the base and step case(s). This corresponds in Horn logic to the existence of the least model and to the consistency of the CWA and its finitary approximation, the *completion* of a program [Cla78]: every atom which is not provable from a program is assumed to be false. Languages which provide embedded implication and universal quantification are instead *open-ended* and thus require the OWA; in fact, dynamic assumptions may, at run-time, extend the current signature and program in a totally unpredictable way. This makes it in general impossible to talk about the closure of such a program. In the literature (reviewed in detail in Section 5.5) the issue has been addressed in essentially three ways:

1. By enforcing a strict distinction between CWA and OWA predicates and applying *NF* only to the former [Har91b], where the latter would require minimal negation, as in [Mom92].
2. By switching to a modal logic, which is able to take into account *arbitrary extensions* of the program as possible worlds (see the completion construction in [GO98] for N-Prolog and [Bon94] for Hypothetical Datalog).
3. By embracing the idea of *partiality* in inductive definitions and using the rule of *definitional reflection* to incorporate a proof-theoretical notion of closure analogous to the completion [SH93, MM97].

None of those approaches are satisfactory for our purposes: most of the predicates we want to negate are open-ended; similarly, definitional reflection is not well-behaved (for example cut is not eliminable) for that very class of programs we are interested in. Moreover, we need to express the negation of a predicate in the same language where the predicate is formulated. Our solution is to restrict the set of programs we deem deniable in a novel way, so as to enforce a *Regular Word Assumption* (RWA): we define a class of programs whose dynamic assumptions extend the current database in a specific regular way. This constitutes a reasonable middle ground between the CWA which allows no dynamic assumption but is amenable to negation and the OWA, where assumptions are totally unpredictable. The RWA is also a promising tool in the study of the meta-logical frameworks [Sch00]. Technically, this regularity under dynamic extension is calibrated so as to ensure that static and dynamic clauses never *overlap*. This property extends to the negative program; in a sense, we maintain a distinction between static and dynamic information, but at a much finer level, i.e. *inside* the definition of a predicate. The resulting fragment is very rich, as it captures the essence of the usage of hypothetical and parametric judgments in a logical framework; namely, that they are intrinsically combined to represent *scoping* constructs in the object language. This is why we contend that this class of programs is adequate for the practice of logical frameworks.

1.2.1 What is Failure (in a Logical Framework)?

A minimal requirement for a negation operator '$\neg$' is that if a set of assumption Γ is consistent, it is the case that $\Gamma \vdash A$ iff *not* $\Gamma \vdash \neg A$. It is a key issue how to interpret the notion of non-existence of a proof. In the logic programming tradition this has been identified with the idea of *finite failure*: the logic programming interpreter is run by querying a given program $\mathcal{P}$ with a goal G; the halting of the query without a derivation is evidence enough to assert the negation of G. This idea actually traces back to the same principle in the deductive database context, where the decision problem has a positive answer. Indeed, in this setting, the *Closed World Assumption* is a most natural one, since, given the large number of entries in a database, the only reasonable way to encode negation is by absence. The transfer of this idea to full logic programming [She85] has been not exactly worry-free, as the enormous literature on the subject testifies. Luckily, our requirements are somewhat different from general logic programming; in fact, in a logical framework, negation refers not to finite failure but to unprovability tout court, as we refrain from negating programs whose negation is not recursively axiomatizable: the adequacy of the representation will break down, since there would be functions which cannot be captured by the framework. We will therefore deal only with terminating programs; this is why we identify negation with a *complement* operation. This restriction, far from being an easy way out, gives us the additional burden to prove that termination is preserved under every manipulation of programs.

It is clear that elimination of negation makes sense only when negation is *stratified* [ABW88], i.e. the negative predicates ultimately refers (in the call graph) to a positive one. We will informally adopt the generally accepted weaker notion of *local stratification* [AB94], when the positive dependency relies not simply on predicate names, but on ground instantiations of literals. While there may be a place in logic programming for non-stratified negation, as the emerging *answer set programming* paradigm [Lif99] testifies, the latter seems to be circumscribed to solving mainly combinatorial problems. This does not seem to be a concern for a logical framework.

1.2.2 Which Logical Framework?

In this dissertation we work with the pattern fragment of third-order HHF; thus our results apply to the same fragment of L_λ [Mil91], although every design decision has been influenced by the possibility to extend it to the richer language of LF and to its implementation in *Twelf*. We comment on this in the conclusions (Chapter 7). *Twelf* can be seen as a dependently-typed CLP-oriented enhancement of L_λ. Both share unification restricted to the pattern fragment, as well as the lack of predicate quantification. For convenience reasons we take the liberty of decorating HHF clauses with labels that can be thought of as names. This allows us to be more concise when applying program transformations. Even though they resemble the same notation in *Twelf*, they lack any intrinsic meaning and will not be used as proof-terms.

Furthermore, we restrict ourselves to HHF without *local* variables. If we look in the usual logic programming fashion at an implicational clause as a rule where the consequent is the 'head' and the antecedent the 'body', a local variable is an essentially existential one which occurs in the body but not in the head. This restriction is customary in the literature on elimination of negation [ST84, MPRT90a]. For example the following clause for typing application cannot be allowed, in this format.

$$\begin{array}{rcl} ofapp & : & \forall E_1, E_2 : exp. \forall T_1, T_2 : tp. \\ & & of\ (app\ E_1\ E_2)\ T_2 \\ & & \quad \leftarrow of\ E_1\ (arrow\ T_1\ T_2) \\ & & \quad \leftarrow of\ E_2\ T_1. \end{array}$$

The problem is that Horn clauses with local variables are already *not* closed under complementation; in fact, elimination of negation will transform those into *extensionally* universally quantified variables. It is a whole new topic to give an operational reading of universal quantification in this setting and to mingle it with parametric judgments. It is our feeling that the issue of local variables during complementation does not have a simple general solution. Approaches which embrace the extensional nature of universal quantification brought in by the negation of existential quantifiers [BMPT90, ABT90] are not satisfactory and robust enough to carry over to logical frameworks with intensional universal quantification, except when dealing with *finite* domains.

While it is well-known that every computable function can be expressed by a Horn programs *without* local variables, we cannot hide that this is a somewhat severe restriction. We offer some ideas on how to partially overcome it in the conclusion (Subsection 7.1.4).

1.3 From Theorem Proving to Prolog

A legitimate question is to ask is why logic programming does not have a primitive notion of negation. To understand that, we need to say something on how logic programming and Prolog developed. This enterprise has a rather peculiar parabola; logic programming owes its (relative) success to the way it limits and directs generic theorem proving; from then on, ironically, most of the effort has been to extend its boundaries without falling back onto full clausal logic.

Automatic theorem proving, or at least the intuition (and the dream), can be dated back to Leibniz, but become more of a reality in 1965 when Robinson introduced the resolution principle [Rob65]. Briefly, it is a proof procedure which proceeds by contradiction, converting a sentence to clausal form and testing for inconsistency with a version of Gentzen's cut-rule augmented with unification. Yet, this approach has been shown to be in general in-practical. A great deal of research developed after Robinson's breakthrough aimed at restricting the search space, while preserving completeness. This is not the place to give even a short account of these studies: we just sketch those that led to the basis of Prolog as we know it; for references and a chronology see [Apt90]. When building a refutation there are basically two sources of choice:

1. Deciding which clauses to pick as parent clauses.

2. Deciding which literals in those clauses are to be resolved away.

One way to support the first restriction is linear resolution, independently proposed by Loveland and Luckam in 1970, which by fixing one goal at each step, never needs to resolve two input clauses together. As far as the second point is concerned, we may decide, after Hill, Kowalski and Kuehner, to fix the literal to resolve in the center clause ('linear resolution with selection function'). Though we have narrowed the search space considerably, there is still a fair amount of choice, namely conjunctive choice in the side clauses, ancestors tracking and factoring. The winning strategy is to restrict the syntax of the clauses themselves; the choice fell on Horn clauses: definite clauses (that is clauses with exactly one positive literal) are interpreted as input ones, while Horn clauses with empty positive part are taken as goals. Eventually we have arrived at pure Prolog or SLD-resolution.

What has SLD-resolution to do with programming? The answer can be found in the so-called procedural interpretation of Horn logic. Although the origins of Prolog are shrouded in mystery, it is known that in 1972 both Kowalski and Colmerauer came up with the idea that (a subset of) logic could be used as a programming language. A definite clause $A \leftarrow B_1, B_2, \ldots, B_n$ can be viewed as a definition of an Algol-like procedure:

procedure A
begin
call B_1
call B_2
$\vdots$
call B_n
end

Goal invocation corresponds to procedure invocation, and the ordering of the goals in the body of the invoked clause corresponds to sequencing of statements. In logic programs data manipulation is entirely achieved by means of unification, which encompasses parameter passing, multiple assignment, record allocation, data construction and selection.

In spite of its limits, it can be shown that Horn logic has the same computational power of every other programming language [Apt90]. Moreover, Horn logic has some nice model-theoretic properties, namely the minimum model property; it is natural to consider the latter as the declarative meaning or the intended interpretation of a program. Therefore it has been argued that we should be content with Horn logic, which seems to be a complete and reasonably efficient computational logic. However, many have been dissatisfied with the difficulty to express even the easiest logical problems in a language that lacks (explicit) disjunction and negation. We share this complaint up to a certain point. We maintain the logic programming works as far as the logical and the algorithmic parts do not differ too much, and that Kowalski's motto "Programs = Logic + Control" has shown its intrinsic limitations. Yet, we strongly share the idea that especially from a programming point of view it would be advisable to have the possibility of performing negative queries and overall to have a negation operator in the body of clauses instead of simulating it with extra-logical constructions, which make programs less understandable and declarative. It is not a question of expressive power, it is a matter of style and convenience.

There are three ways, in order of increasing complexity, to add negation to Horn logic:

- Negative atomic queries.
- Negative literals in clauses bodies.
- Negative heads.

It is not possible to try to review all the proposed extensions; historically much of the attention has been concentrated on incorporating *NF*; from that, most of first-order expressivity is recovered [LT84].

1.4 Negation-as-Failure

Since negative information is independent from definite programs, a specialized inference rule must be invoked: negation as failure (*NF*), which in logic programming, originated from the confluence of two quite different trends of research: the refinements of resolution based automatic proof procedures and the relational approach to databases. For a nice introduction see [She88, AB94]. The idea of a proof under the *NF* rule is a natural one: suppose you have a set of axioms and some kind of inference mechanism which produces a recursively enumerable set of theorems, and that you are asked to verify the truth of a negative conjecture $\neg C$ under *NF*; then you try to prove C from your theory; if you succeed, then $\neg C$ does not hold, while if you realize (in a finite time) that C is not provable, then you are entitled to assert that $\neg C$ holds. Its basic idea is to state that a goal is false if we are able to prove that it cannot be proved by the program. Actually, *NF* is more a computational than a logical notion; we answer 'no' to a goal because our attempt to say 'yes' failed, so we say 'no' because we cannot say 'yes;. Differently from other kind of negation, *NF* "...does not follow from some constructive knowledge, but from lack of knowledge" ([Gab91] pp. 8). That motivates its intrinsic non-monotonicity: in fact, in dynamic databases every enlargement may cause the meaning of failure to change and so turn success into failure.

In logic programming, *NF* works this way:

> " ... The basic idea is to use *SLD*-resolution augmented by the *NF* rule. When a positive literal is selected, we use essentially *SLD*-derivation to derive a new goal. However, when a ground negative literal is selected, the goal answering process is entered recursively in order to try to establish the negative subgoal ... Having selected ground negative literal $\neg A$ in some goal, an attempt is made to construct a finitely failed *SLDNF* tree with root $\leftarrow A$ before continuing with the remainder of the computation. If such a tree is constructed, then the subgoal $\neg A$ succeeds. Otherwise, if a *SLDNF*-refutation is found for $\neg A$, then the subgoal fails ... " ([Llo93] p. 87).

The operational nature of this rule motivates the lack of a unique semantics and some of its related troublesome features: to begin with, possible unsoundness: without run-time checks on the substitution returned by a negative open query, the final answer substitution may not be a logical consequence of the program. This is the so-called "floundering" phenomenon, the undecidable question of whether the computation will reach a negative open query and abort. Soundness is preserved only for ground queries; the flip side of the medal is that now negation is not a first-class connective, but just a test that cannot return substitutions. We review in Section 1.5.1 how and with what computational cost this can be avoided. And of course, *NF* is in general *incomplete* in general logic programming.

All of the above makes *NF* a suspicious candidate for a negation operator in any logic programming language, but the situation is even worse in logical frameworks. Even if we manage to isolate a well-behaved logical fragment, such as acyclic normal programs [AB90], allowing *NF* in a logical framework carries some additional problems. First, the meta-theory becomes really unwieldy, as both provability and unprovability must now be taken into account. The two systems would be interlinked by rules such as:

$$\frac{\Gamma \not\vdash F}{\Gamma \vdash \neg F}\neg - R+ \qquad \frac{\Gamma \vdash F}{\Gamma \not\vdash \neg F}\neg - R-$$

where $\not\vdash$ denotes a proof system for finite failure. In a type-theoretic logical frameworkthis issue is further exacerbated by the need to deliver evidence of what a proof of a certain judgment is. The most popular way, since the Automath project [dB80], is to to see derivations as lambda terms inhabiting judgments seen as types. Although it is in principle possible to associate proof-terms to a derivation by negation-as-failure – this is implicit in the denial proof system that we present in Chapter 6, Figure 6.1 and 6.2 – the existence of (unique) canonical forms is in general impossible to achieve; and this is pretty much a death sentence for *NF*. In fact, in frameworks with hypothetical judgments, as recognized first by Gabbay [Gab85], the unrestricted combination of *NF* and embedded implication is particularly problematic, since it leads to the failure of basic logic principles such as cut-elimination. We discuss this issue in details in Section 5.5.

At the user level, the presence of *NF* in a logical framework would make adequacy theorems more difficult to establish, again because both provability and unprovability now need to be considered.

In summary the adoption of *NF* in a logical framework seems to be a very risky, if not hopeless road, considering its fragility already in the very simple setting of Horn clauses.

1.5 Extending Horn Logic

As previously mentioned, once Horn logic was isolated as the core of a programming language, a fairly disorderly race was off to get more mileage out of Prolog. To sum up, we can isolate several (slightly overlapping) positions:

- The "tories": for model-theoretic reasons, Horn logic is the best possible world, see the manifest "Why Horn logic matters in computer science" [Mak87].
- The "realists", guided by Apt: logic programming *is* Horn logic with *NF* : what's left to do is logicize the impure features of Prolog.
- The "Making Prolog more expressive" people: divided in two main intertwined sub-tribes: the "logicians", which claim that programming in Horn logic is like living with one hand tied behind your back, and the "compilers" (see Sato and Tamaki. [TS84]: those come from the specification approach

and look at Horn logic as a implementation language which is the target of a long and tiresome travel through derivation and/or transformations from first-order logic. For the logicians, it is a must to conquer any piece of land outside Horn logic, say by adding connectives [PG86], pre-compilation [LT84], change of interpreters (say connection graphs [GR87]) or, more reasonably, switching from classical logic to fragments in the intuitionistic galaxy, reviewed in Section 5.5.

- Finally, there is the proof-theoretic approach of uniform proofs: new connectives are allowed only if we can ascribe a clear meaning in term of search and provide a way of endowing logic programming in a purely logical way with features such as modules, data abstraction and scoping typical of other mature languages.

We now concentrate on how recent research has tried to address some of the problems connected with *NF*.

1.5.1 Constructive Negation

Constructive Negation is an attempt to devise methods capable to provide logically justified answers to non-ground negative queries, in analogy with the witnessing property of constructive logics. Formally, for a suitable derivability relation, this property ensures that from $\vdash \exists x \neg p(x)$ we can infer the existence of a term t such that $\vdash \neg p(t)$. We can roughly distinguish two approaches:

i. Program Transformation: [ST84], [FRTW88], [BMPT90].

ii. Negation by Constraints: [Wal87] for Datalog programs, [Cha88] [Cha89] and extended to CLP in [Stu95]; Fail Substitutions: [She89] [MN89].

Historically, the original attempt to deal with the issue was simply to try avoiding the floundering phenomenon: given that the latter is in general undecidable, one possibility is to try to make sure that when a negative literal is called it has already been grounded: there are basically three possibilities:

1. Satisfy the syntactic, though very restrictive, conditions on allowed computations [She85], which essentially reduces evaluation to *ground* evaluation.

2. Try to achieve grounding by delaying as in *[M]NU-Prolog* [Nai86] or *Sicstus* [AAB+95], where a goal may be declared to be "frozen" and is evaluated only when it reaches a sufficient degree of instantiation. This is obviously only a partial solution, since at run-time there is no guarantee to eventually ground the problematic query. A more complex and historically less successful alternative is offered by the computation rules of *IC-Prolog*, which allow the computation of negative open queries if their positive counterpart does not bound any variable (see [Nai86], for a comprehensive analysis and references).

3. Covering the open negative query with a generator of values for the relevant variables. This is further detailed next.

Static Approaches

If we are dealing with Datalog programs, i.e. with finite Herbrand Universe (U_P), the naive approach would be to instantiate all the rules with potentially troublesome goal with terms from U_P [ABW88], say through propagation in every negative literal in the program. This is clearly infeasible, since it may result in an intractable numbers of rules, especially in an untyped setting.

A sophistication of this idea can be found in [FRTW88]: the proposal is to automatically infer a 'type' for the problematic variables and transform the original program into one where grounding is ensured by coverage from those types. Then useless answers originating from general instantiation would be excluded by the typing discipline. Although it can be shown that the new program is equivalent to the old one, this cannot be extended to full Prolog: function symbols make the type infinite and non-ground facts would undermine the instantiation capability of the type.

Finally, the transformation approach falls in this category and is detailed in Chapter 5.

Dynamic Approaches

Chan [Cha88] is acknowledged to be the inventor of the term 'constructive' negation in this area; his approach can be roughly characterized as mixing *NF* with a constraints-solving attitude. In essence it consists in evaluating a negative goal by executing its positive version and by negating the answer obtained. As in the *CLP* family of languages, unification and disunification are kept explicit and returned as solutions. Of course, we need to keep the (in)equalities in normal form and there are some obvious problems when dealing with computations that have infinite answers; those are addressed in a following paper ([Cha89]), by quantifying over the answer substitutions. No proof of completeness is offered. We can offer the following rational reconstruction: the key observation is that if G is a goal and we consider the answer substitutions $\theta_1, ..., \theta_n$ as equations, $G \leftrightarrow \exists(\theta_1 \vee \ldots \vee \theta_n)$ is a logical consequence of the completed database. Therefore the constructive negation rule is simply $\neg G \leftrightarrow \neg\forall(\theta_1 \vee \ldots \vee \theta_n)$, where the right-hand side can be simplified by disunification. For instance given the query $not(even(X))$, its positive version yields the answer $X = 0 \vee \exists Y : X = s(s(Y))$, whose negation is $X \neq 0 \wedge \forall Y : X \neq s(s(Y))$: its solved form is hence $X = s(0)$, which we can regard as a more informative refinement of the answer constructive negation produces.

A generalization to constraint logic programming over arbitrary structures is given in [Stu95]; it turns out to be sound and complete w.r.t. the three-valued models of the completion. Other development of constructive negation are addressed in [Fag97].

$(E)SLDNF - S$ ([She89]) The finite failure case in the definition of $SLDNF$-resolution is modified as follows: a goal $(\Gamma, \neg A)$ has a descendent $\theta\Gamma$, if there is a finitely failed-tree for θA, where $dom(\theta) \in FV(A)$. So *NF* can instantiate under success, i.e. negative goals may directly return substitutions: given P and G the aim is to look for a (fail) substitution θ such that $P \vdash \theta G$ has a finitely failed tree; then by the soundness of the *NF* rule $\forall\theta\neg G$ is a consequence of $comp(P)$ and thus θ is an answer substitution for the query $\neg G$. This seems very costly, since it entails enumerating (guessing) every fail substitution. I am not aware of any implementation of this proposal.

This is refined in [MN89], where it is shown how to avoid to generate all possible substitutions in lieu of a maximal general fail substitution. Moreover, the improvement w.r.t. Chan's work lies in the feature of always including some positive bindings for the variable in the negated goal. If the SLD-tree is infinite, the method enumerates the set of fail substitutions; this corresponds to the fact that in general negative queries cannot be represented by finite positive information alone (connected to [LM87]).

1.5.2 Non-Failure Driven Negation

During the years ways of incorporating other more logical forms of negation than NF have appeared. Since most of the time this gives back full non clausal-logic, most of them are cataloged as automated theorem provers. In all these accounts, negative information has to be provided explicitly and specific rules are offered to deal with that. Sometimes it is possible to mix "open world" and "closed world" predicates safely. For a more detailed account and bibliography, let me refer to [Mom92].

- *N(Q)Prolog* [GR84], a complete implementation of positive intuitionistic logic. By defining disjunction classically and allowing a restart rule (see *nH Prolog* next), Gabbay shows it to be complete for full classical logic as well.

- *Negation as Inconsistency* ([GS86]). Here we evaluate a query against an ordered pair $\langle P, N\rangle$, where P is a Horn program and N a set of queries that are required *not* to succeed; this is logically equivalent to adding to the program the negation of all the members of N, and permits importing negative facts and rules. Both systems have a very awkward first-order version.

- Stickel's $PTTP$, supplements SLD-resolution with the model elimination rule. This entails keeping track of the ancestors of the goal, loosing one of the key feature of Prolog, namely input resolution.

- Loveland's *nH Prolog* [RL92] incorporates case analysis in SLD-resolution, by demanding the invocation of a restart rule for every disjunctive head, until the stack of the former. Without requiring contrapositives (as in $PTTP$), it simulates case analysis with different runs of essentially the Prolog engine. Unfortunately naive *nH-Prolog* is incomplete and the new versions (*Progressive nH* and *Inheritance nH*) have a less natural and convincing description.

- Another extension goes under the name of *disjunctive logic programming* (see [LMR92] and references therein). It aims to deal with full clausal logic by generalizing Horn clauses to disjunctive heads.

1.5.3 Proof-Theoretic Approaches to Negation and *NF*

In the 90's there has been an attempt to tie LP to proof-theory, where it belongs: and this has brought new insights, particularly on *NF*.

The first step is to view Horn clauses positively as rules and goals as existentially closed conjunctions of atoms to be proved by the former. Historically this can probably be dated back to Gabbay and Reyle [GR84]. It is customary [HSH90] to distinguish among two approaches:

1. Clauses as axioms (programs as theories) and some form of Gentzen sequent calculus to infer goals, i.e. uniform proofs systems.
2. Clauses as rules [HSH90]: Horn (and beyond) programs should be seen as set of inference rules for the derivation of (not necessarily ground) atoms.

This has the following relation with negation:

1. Minimal, intuitionistic and classical negation can be superimposed over uniform proofs [Mil89c], [Har91a], [Mom92]: Minimal negation, being camouflaged implication, is executed through the $AUGMENT$ and *backchain* operations; the evaluation of $\neg D$ consists in the assumption of D and in the attempt to prove $\perp$ from the enlarged theory. The Duns Scoto Law and Reductio ad Absurdum for atoms formalize the latter, preserving the feature of abstract logic programming languages [MNPS91].
2. GCLA [MAK91] is based on the rule-based definitional approach to logic programming: it has intuitionistic negation built-in, applying the *definiens* operator to the left-hand side of a sequent. A discussion can be found in Section 5.5.4.

Stärk [Stä92] has given a sequent calculus reconstruction of *NF* using Clark's equality and freeness axioms, negation (switch) rule and cut rules. Much more is however contained in Stärk's thesis and subsequent research, although not directly applicable to our goals; to quote a few, he shows that a sequent is provable in this calculus iff it is true in all 3-valued model of the completion. Furthermore a completeness result is proved w.r.t $SLDNF$-resolution for program satisfying the cut-property.

1.5.4 Outline

This dissertation is organized in two main parts which address:

- The relative complement problem for higher-order patterns.
- Clause complementation for a fragment of third-order Hereditary Harrop formulae.

We start in Chapter 2 by introducing the relative complement problem; we review the existing solutions to the first-order case in the literature, namely a variant of Lassez & Marriot's original *uncover* algorithm [LM87] (Section 2.1) and *disunification* [Com91] (Section 2.2). We then discuss in Section 2.4 the problems connected to extending those idea to the higher-order case, where we notice the fundamental difference between *fully* and *partially applied terms*. For the latter fragment, the simply-typed λ calculus is not closed under term complement. We remedy this by introducing the *strict* λ-calculus in Chapter 3. We develop the system and mention the existence of canonical forms. Once we have a calculus strong enough to deal with partially applied terms, Section 4.1 introduces a restriction of the language ("simple terms") for which complementation is possible. The algorithm for negation is presented in Section 4.2; in Section 4.3 we give a unification algorithm for the same fragment. This completes our solution to the relative complement problems for higher-order patterns. We conclude this chapter in Section 4.4 by showing how to organize finite sets of simple terms into a boolean algebra. We end up this part of the dissertation reviewing related work on strictness (Section 3.3).

Chapter 5 sets the stage for clause complementation. First, in Section 5.1, we offer a reconstruction of the transformational approach to negation in the Horn case. Then in Section 5.2 and 5.3 we give an informal view of the complement algorithm for HHF and of the restrictions it requires by means of examples. In Section 5.4 we try to motivate the pragmatic adequacy of the fragment of HHF we deal with, while Section 5.5 reviews the state of the art in *NF* and intuitionistic provability.

Chapter 6 is the heart of the thesis; we first introduce the source language and its uniform proofs system in Section 6.1. We then establish the fundamental notion of context schema (Section 6.2). This allows to enforce the *Regular World Assumption* (RWA), on which clause complementation is built. After formalizing the restriction to terminating programs in Section 6.3, we present the clause complementation algorithm and the related notion of *augmentation* (Section 6.5 and 6.6). We then prove the main theorem (Section 6.7 and 6.8). Finally, Section 6.9 discusses how to give an operational semantics to our language.

We conclude the dissertation in Chapter 7 by discussing first how to lift some of the current restrictions (Section 7.1); then we address possible extensions, implementation issues and further future work (Section 7.2, 7.3 and 7.4).

1.6 Contributions and Technical Acknowledgments

The original contribution of the thesis are:

- A relative complement algorithm for higher-order patterns internalized into a strict type theory.
- A complement algorithm for a useful class of third-order hereditary Harrop formulae.

We contend that our approach is the first one to give a realistic analysis of negation in logical frameworks with an emphasis on the development of a practical tool to incorporate this operator in existing languages.

This work has benefited enormously from the large ensemble of research collected in the Elf and offspring projects: not only form the existence of this language and environment, but also from specific contributions which we have used (in a somewhat simplified setting) in this thesis. Let me mention only the most recent ones: schema contexts (Schürmann [Sch00]), linear unification (Cervesato and Pfenning [CP96]), subordination (Virga [Vir99]), mode and termination analysis (Rohwedder and Pfenning [RP96]).

This research has been financially supported for seven semesters by the Department of Philosophy at CMU and by a one-year scholarship from “Consiglio Nazionale delle Ricerche”, Italy.

Chapter 2

The Relative Complement Problem

An open term t in a given signature can be seen as the intensional representation of the set of its ground instances, say $\|t\|$. According to this interpretation, the *complement* of t is the set of ground terms which are *not* instances of t, i.e. are in the set-theoretic complement of $\|t\|$. It is natural to generalize this to the notion of *relative* complement; this corresponds to computing a suitable representation of all the ground instances of a given (finite) set of terms which are not instances of another given one, in symbols:

$$\|t_1, \ldots, t_n\| - \|u_1, \ldots, u_m\|$$

where dots represent (set theoretic) union[1]. More properly:

$$\|t_1, \ldots, t_n\| \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \|t_i\|$$

Let $FV(t_1, \ldots, t_n) = \vec{x}$ disjoint from $FV(u_1, \ldots, u_m) = \vec{y}$. Then the relative complement problem can be also expressed by the following (restricted) form of *equational problem* [Com91], where the z_i's are free variables.

$$\exists \vec{x} \forall \vec{y} : \bigwedge_{i=1}^{n} z_i = t_i \wedge \bigwedge_{i=1}^{m} z_i \neq u_i$$

Example 2.1 *Consider the signature containing the usual declarations for* $0, s, +$. *The following rules define integer addition modulo* 2.

$$\begin{array}{rcl} s(s(0)) & \longmapsto & 0 \\ y + 0 & \longmapsto & y \\ 0 + y & \longmapsto & y \\ y + y & \longmapsto & 0 \end{array}$$

The following relative complement problem expresses the question of sufficient completeness (in this case yielding a positive answer) of the rewrite rules:

$$\|x_1 + x_2\| - \|s(s(0)), 0 + y, y + 0, y + y\|$$

which corresponds to:

$$\exists x_1 x_2 \forall y : (z = x_1 + x_2) \wedge (z \neq s(s(0))) \wedge (z \neq y + 0) \wedge (z \neq 0 + y) \wedge (z \neq y + y)$$

Then, since a variable stands for the universe of discourse, a *complement problem* is representable merely by:

$$\|z\| - \|u_1, \ldots, u_m\|$$

[1]Another equivalent notation found in the literature is $t_1 \vee \cdots \vee t_n \setminus u_1 \vee \cdots \vee u_m$ [LM87], or a mixture of the two.

or a simpler ($\exists$-degenerate) equational problem:

$$\forall \vec{y} : \bigwedge_{i=1}^{m} z \neq u_i$$

Now we turn to review solutions to the relative complement problem in first-order languages.

2.1 The Not Algorithm

We start with the 'Not' algorithm for first-order terms that specializes the prototypical *uncover* algorithm proposed in [LM87] as a first attempt to solve the the problem and is at the heart of Barbuti et al.'s approach [BMPT90]. We present it in a many-sorted framework, differently from the uni-sorted original version. We call (in this chapter) a term *linear* if it does not contain repeated occurrences of a free variable.

Definition 2.2 *Consider a many-sorted signature Σ and a linear term t of type τ. We define* Not(t) *by structural induction, where we suppose that t_i has sort τ_i and the z's are new free variables of appropriate typing:*

$$\begin{array}{ll} \mathrm{Not}(z:\tau) & = \emptyset \\ \mathrm{Not}(f(\overline{t_n}):\tau) & = \\ & \{g(\overline{z_m}):\tau \mid g \in \Sigma, g \not\equiv f, g : \overline{\tau_m} \to \tau\} \cup \\ & \{f(z_1,\ldots,z_{i-1},s,z_{i+1},\ldots,z_n):\tau \mid s \in \mathrm{Not}(t_i), 1 \leq in\} \end{array}$$

The uni-sorted version of this function tends to produce a lot of irrelevant outcomes. For example, $\mathrm{Not}(cons(s(x),nil))$ does not yield only the desired $\{nil, cons(0,nil), cons(y, cons(z,xs))\}$ in the informal signature of lists of numerals but also $\{0, s(x), \ldots\}$. On the other hand, fixing

$$\Sigma = \{0 : nat, nil : nlist, s : nat \to nat, cons : nat * nlist \to nlist\}$$

we get the desired result. This problem may tend to increase dramatically with the size of the signature. It can be argued that the notion of complementation itself without an underlying type discipline makes little sense, not only from a complexity standpoint, but also in intellectual terms. Moreover, more refined type theories, as dependent types and/or sub-typing will further constrain the result of the evaluation of Not.

A complement operator must satisfy the following desiderata:

1. Exclusivity: it is not the case that s is both a ground instance of t and of Not(t).

2. Exhaustivity: s is a ground instance of t or s is a ground instance of Not(t).

That is, the Not algorithm ought to behave as a the complement operation on sets of ground terms. This cannot be achieved in all generality. In other words, intensional representations of terms are not closed under complementation. One canonical example is as follows:

Example 2.3 *Consider the signature $\{a : i, f : (i * i) \to i\}$: intuitively the complement of $f(y,y)$ should be:*

$$\|z\| - \|f(y,y)\| \quad = \quad \{a\} \cup \{f(x,z) \mid x \neq z\}$$

Instead, the Not *algorithm would incorrectly yield:*

$$\mathrm{Not}(f(y,y)) \stackrel{?}{=} \{a\}$$

In fact, Lassez & Marriot [LM87] have been the first to point out that this complement algorithm is correct only for *linear* terms: complement of non-linear ones do not have a straightforward finite representation. More sophisticated representation, such as *constrained terms* [Com88] have been investigated, but are not suitable to our applications.

Moreover, as well known, the restriction to linearity seems to be almost immaterial in logic programming thanks to the idea of *left-linearization* introduced by Plaisted and used first by Stickel [Sti88] to avoid unnecessary occur checks testing. It simply consists of a source-to-source transformation which replaces repeated occurrence of the same variable in a clause head with new variables which are then constrained in the body by a new predicate, say eq, whose definition is simply $eq(x, x)$; unification will then provide the other properties of equality. For example, continuing Example 2.3, a clause such as $\forall y \,.\, p(y, y) \leftarrow G$ would be replaced by $\forall z_1 .\, \forall z_2 \,.\, p(z_1, z_2) \leftarrow eq(z_1, z_2) \wedge G$. As a matter of fact, this approach is less innocent than it looks at first sight, since it opens the road to a CLP attitude; moreover, eq as a predicate is *not* linear in itself and required an ad hoc treatment in the transformational approach to negation [BMPT90]. Miller [Mil89a] has shown how to automatically infer the equality predicate (the `copy` clause, in his terminology) for any type. However, for any order higher than the first, this clauses are not Horn and their negation is itself problematic. One of the result in this dissertation is to apply elimination of negation to predicates such as `copy`.

Once we have a way to solve complement problems, it is easy to pair it to intersection, seen as unification [Plo71], to have a solution to relative complements as well, i.e.

$$\|t_1\| - \|u_1, \ldots, u_m\| \quad \stackrel{\text{def}}{=} \quad \|t_1\| \cap \|\mathrm{Not}(u_1)\| \cap \ldots \cap \|\mathrm{Not}(u_m)\|$$

Another more general approach is possible. As we have seen in the beginning, it is possible to express the (relative) complement problem on terms as an equational problem. This is the basis to solve complement problems with disunification, as we sketch next.

2.2 Disunification

Disunification is devoted to solving *arbitrary* first order formulae whose only predicate symbol is equality, call them *equational formulae*. The definition of what a *solution* is differs on the application at hand. We may be interested only in the overall validity or in the possible assignments that make the formula valid. As we have seen, complement problems can be seen as systems of dis-equations with universally quantified variables. Thus a disunification algorithm (over first-order terms) will solve these problems, possibly providing values for free variables.

From an historic perspective this field became defined when it was realized by Martelli & Montanari [MM82], if not by Herbrand (see the Appendix in [Sny91]) that first-order unification can be seen as a set of transformations on sets of equations. On the other hand the work of Mal'cev [Mal71] on the decidability and the possibility to give complete axiomatization of the theory of equational algebras qualifies as an ancestor. The definition of Prolog-II introduced first-class dis-equations. Indeed Colmerauer [Col84] showed them to have solutions in the algebra of rational trees. Next, Lassez & Marriot [LM87] proposed the seminal (although awkward) *uncover* algorithm for computing relative complements. Kirchner and Lescanne first unified those previous papers in the framework of equational problems and proposed a set of transformational rules, though without a completeness proof [KL87]. Maher introduced the unification community to Mal'cev's results [Mah88]. Comon and Lescanne were the first one to present an adequate set of rules [Com88, CL89] and the former surveyed the field [Com91].

How to go on to derive a disunification procedure can vary from the syntax and semantics we are concerned with, but nevertheless it entails the following steps:

- Provide a set of axioms $\mathcal{T}$ that hold in the model we consider.
- Design a set of rules $\mathcal{R}$ for the transformation of equational formulae that can be proven correct w.r.t. $\mathcal{T}$.
- Design a control C on $\mathcal{R}$ such that the application of rules satisfying C terminates: irreducible formulae are in solved form and have the same set of solutions as the original problem.
- If arbitrary formulae are allowed and solved forms are trivially decidable, this entails the decidability and completeness of $\mathcal{T}$.

The simplest example is unification of finite (first-order) terms:

R	:	$w = t \wedge P[w]$	$\longmapsto$	$w = t \wedge [t/w]P$
M_1	:	$w = t \wedge w \neq u$	$\longmapsto$	$w = t \wedge t \neq u$
UE_1	:	$\forall \vec{y}, y \,.\, P \wedge y \neq t$	$\longmapsto$	$\perp$
$Clh(T)$	:	$f(\overline{t_n}) \neq g(\overline{s_m})$	$\longmapsto$	$\top$
$Clh(F)$	:	$f(\overline{t_n}) = g(\overline{s_m})$	$\longmapsto$	$\perp$
Dec_1	:	$f(\overline{t_n}) \dot{\neq} f(\overline{u_n})$	$\longmapsto$	$\bigvee_{i=1}^{n} t_i \dot{\neq} u_i$
Dec_2	:	$f(\overline{t_n}) = f(\overline{u_n})$	$\longmapsto$	$\bigwedge_{i=1}^{n} t_i = u_i$
E	:	$\forall \vec{y} \,.\, P$	$\longmapsto$	$\bigvee_{f \in \Sigma} (\exists \vec{x} \forall \vec{y} \,.\, P \wedge w = f(\vec{x}))$

In rule R $w \notin Var(t)$, in E, P contains a (dis)equation with LHS w and RHS u, where the latter is not a variable and contains a universally quantified variable.

Figure 2.1: Some disunification rules

- $\mathcal{T}$ is Clark's free equality theory [Cla78].
- $\mathcal{R}$ are, say, the Martelli-Montanari rules; correctness corresponds to the preservation of solutions under rule application.
- Solved forms yield idempotent substitutions and control restricts the application of variable elimination. Completeness (of the theory) is established for example as in [Mah88].

The disunification rewrite rules are divided into three big classes:

- Equality rules, i.e. rules which are correct for any equational algebra
- Rules for finite trees over any signature
- Rules for finite trees over a finite signature.

We will not present the complete set of rules with logical provisos and control. We just mention that the first group contains Replacement, Universal Quantifier Elimination, Existential Quantifier Elimination and Elimination of Disjunctions. The second batch contain Clash, Decomposition, Occur check. The third section would contain rules which are sensitive to the cardinality of the signature. Here we mention only the Explosion rule (E), which is motivated by the domain closure axiom (*DCA*) [MMP88]. We list in Figure 2.1 the rules relevant to the following example in their barest form, i.e. with only soundness and no termination condition:

We now give an example of disunification on the numerals signature, which is required in the synthesis of the `odd` program (see Figure 5.1). It consists in solving:

$$\forall y : z \neq 0 \wedge z \neq s(s(y)) \tag{2.1}$$

The intuitive solution of (2.1) is $z = s(0)$. We will use the rules in Figure 2.1 and gloss over normalization steps as well as elimination of trivial (dis)equations. Branches stemming from the explosion rule are numbered and pursued separately (keeping in mind that they form a disjunction, i.e. a finitely branching tree from a search standpoint). $R(x)$ denote application of the rule R on variable x. The computation is traced in Figure 2.2.

Note that disunification nicely overcomes the difference between *linear* and *non-linear* terms with different notions of solved forms, namely *unification solved form* versus *solved form with dis-equations* [Com91]. This may be interpreted as evidence of the opportunity of rephrasing unrestricted relative complements as disunification problems. We, on the other hand, maintain that this approach is unnecessarily general for this purpose. Implementing disunification entails managing the non-deterministic application of a few

$$
\begin{array}{ll}
\boxed{\forall y : z \neq 0 \wedge z \neq s(s(y))} & \\
\longmapsto_{E(z)(1,2)} & \\
 & (1)\ \forall y : z \neq 0 \wedge z \neq s(s(y)) \wedge z = 0 \\
\longmapsto_{M_1(z)} & \\
 & \forall y : 0 \neq 0 \wedge 0 \neq s(s(y)) \wedge z = 0 \\
\longmapsto^*_{Clh(F)} \boxed{\perp} & \\
 & (2)\ \exists x \forall y : z \neq 0 \wedge z \neq s(s(y)) \wedge z = s(x) \\
\longmapsto_{M_1(z)} & \\
 & \exists x \forall y : s(x) \neq 0 \wedge s(x) \neq s(s(y)) \wedge z = s(x) \\
\longmapsto^*_{Dec,Clh(T)} & \\
 & \exists x \forall y : x \neq s(y) \wedge z = s(x) \\
\longmapsto_{E(x)(2.1,2.2)} & \\
 & (2.1)\ \exists x \forall y : x \neq s(y) \wedge z = s(x) \wedge x = 0 \\
\longmapsto_{M_1(z)} & \\
 & \exists x \forall y : 0 \neq s(y) \wedge z = s(0) \wedge x = 0 \\
\longmapsto^*_{Clh(T)} \boxed{z = s(0)} & \\
 & (2.2)\ \exists x_1 x \forall y : x \neq s(y) \wedge z = s(x) \wedge x = s(x)_1 \\
\longmapsto^*_{M_1(x),Dec_1} & \\
 & \exists x_1 x \forall y : x_1 \neq y \wedge z = s(x) \wedge x = s(x)_1 \\
\longmapsto^*_{UE_1(y)} \boxed{\perp} &
\end{array}
$$

Figure 2.2: Computation of $\forall y : z \neq 0 \wedge z \neq s(s(y))$

dozen rules which eventually turns a given problem into a solved form. Though a reduction to a significant subset of the disunification rules as the one depicted in Figure 2.1 is likely to be attainable for complement problems, control is a major problem. Moreover the higher-order case results in additional complications, such as restrictions on the occurrences of bound variables, which fall outside an otherwise clean framework. As we show in this dissertation, this must not necessarily be the case. We believe that our techniques for the higher-order case can also be applied to analyze disunification, although we have not investigated this possibility at present.

2.3 Other Applications

Complement problems and elimination of negation are not restricted to logic programming, but have some other relevant application in theoretical computer science. Let me refer to [JLLM91] and [Com91] for issues impossible to detail here and for complete references.

In fact, complement problems and variants of the *uncover* algorithm [LM87] as a first attempt to solve the former, have been studied and tentatively applied in several ways:

- In functional programming, to determine, modulo pattern matching, whether the program clauses describing a function are exhaustive and disjoint, even further to produce a non-ambiguous set of patterns. Moreover, it is possible to take advantage of the given sequential application of the rules to provide an improved compiled code. Indeed, if, say, the second rule applies, it means the first one does not: hence the terms reducible by the second rule are in the complement of the LHS of the first.

- The connection of complementation to the notion of *ground reducibility* in term rewriting systems makes it a candidate as a checker for *sufficient completeness* [GH78] of an algebraic (equational) specification. If the latter fails to be complete, the transformation rules my lead to recover the missing cases—hence the motto in [Thi84]:

  "Stop losing sleep over incomplete specifications"

Here we are looking for counter-examples: if a function is not sufficiently complete, there is a term, built from the constructors in the signature, which is different from any LHS, thus irreducible (see Example 2.1).

- In term rewriting systems describing infinite transition systems, the complement of a LHS returns the states from which no transition is achievable, providing thusly a tool for the temporal analysis of communicating processes.
- In machine learning, a concept can be captured by a term with some (finite) exceptions: the computation of this structure, which is a relative complement, coincides with the search for an explicit representation of the cited concept.
- Finally, applications to inductive theorem proving under the slogan *induction-less induction* or *proof by consistency* [Com98] are under scrutiny. This is connected to the idea of the so-called *inductive reducibility* property.

We now switch gears and discuss the extension of the relative complement problem to the higher-order case; as usual, we restrict ourselves to a specific class of λ-terms.

2.4 Complementing Higher-Order Patterns

In most functional and logic programming languages the notion of a pattern, together with the requisite algorithms for matching or unification, play an important role in the operational semantics. And, of course, patterns form the left-hand sides of rewrite rules and are thus critical to the study of rewrite systems. Consequently, analysis of the structure of patterns is an important task in the implementation of programming languages and more abstract studies of rewriting systems and their properties.

Perhaps the most fundamental problems are matching and unification, but other questions such as generalization also arise frequently. Here, we are concerned with the problem of pattern complement in a setting were patterns may contain binding operators, so-called *higher-order patterns* [Mil91, Nip91]. A term possibly containing some existential variables is called a *pattern* if each occurrence of an existential variable has the form $E\ x_1 \ldots x_n$, where the arguments x_i are distinct occurrences of free or bound variables (but not existential variables). Higher-order patterns have found applications in logic programming [Mil91, Pfe91a, MP93], logical frameworks [DPS97], term rewriting [Nip93], and functional logic programming [HP96]. Higher-order patterns inherit many pleasant properties from the first-order case. In particular, most general unifiers [Mil91] and least general generalizations [Pfe91b] exist, even for complex type theories.

In this section we discuss some of the preliminary issues towards a generalization to the complement algorithm to higher-order patterns. We assume the following:

- All terms are linear, i.e. existential variables occurs only once.
- Types do not contain occurrences of the primitive type $\mathbf{o}$. We thus complement only terms with no inner logical structure.

The main difference w.r.t. the first-order case is twofold: first, the second-order (relative) complement problem is not semi-decidable, but higher-order disunification on higher-order patterns is decidable [Lug94].

Secondly, as we will see, the class of patterns is *not* closed under complement, although a special subclass is. We call a canonical pattern $\Gamma \vdash M : A$ *fully applied* if each occurrence of an existential variable E under binders $y_1, \ldots, y_m$ is applied to some permutation of the variables in Γ and $y_1, \ldots, y_m$. This is formally defined in Figure 2.3. Fully applied patterns play an important role in functional logic programming and rewriting [HP96] because any fully applied existential variable $\Gamma \vdash E\ x_1 \ldots x_n$ denotes all canonical terms with free variables from Γ. It is this property which makes complementation particularly simple. In fact, the main difference with the first-order case is that we need to carefully keep track of bound variables: those are collected in a context Γ, so that the complement of a rigid term is taken w.r.t. both the signature and the current context. In the case the term is *not* fully applied, the complement has to take into account whether some of the variables mentioned in a lambda binder do appear in the matrix; we discuss this in the next Section 2.5. We first analyze the simpler case.

$$\frac{\overline{y_n} = \mathrm{dom}(\Gamma)}{\Gamma \vdash_\Sigma E\ \overline{y_n}\ \text{ f.a.}}\ \mathtt{FaPat} \qquad \frac{\Gamma, x{:}A \vdash_\Sigma M\ \text{ f.a.}}{\Gamma \vdash_\Sigma \lambda x{:}A.\, M\ \text{ f.a.}}\ \mathtt{FaLam}$$

$$\frac{h \in \Gamma \cup \Sigma \qquad \Gamma \vdash_\Sigma N_1\ \text{ f.a.}\ \cdots \Gamma \vdash_\Sigma N_n\ \text{ f.a.}}{\Gamma \vdash_\Sigma h\ \overline{N_n}\ \text{ f.a.}}\ \mathtt{FaApp}$$

Figure 2.3: M is a fully applied pattern: $\Gamma \vdash_\Sigma M$ f.a.

The language of the simply-typed λ-calculus is as follows, where we use a for atomic types (different from the type of proposition $\mathbf{o}$), c for term-level constants, and x for term-level variables, while h will stand for a constant *or* a variable.

$$\begin{array}{rrcl}
\textit{Simple Types} & A & ::= & a \mid A_1 \to A_2 \\
\textit{Terms} & M & ::= & c \mid x \mid \lambda x{:}A.\ M \mid M_1\ M_2 \\
\textit{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}\text{type} \mid \Sigma, c{:}A \\
\textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}$$

We require that signatures and contexts declare each constant or variable at most once so that, for example, when we write $\Gamma, x{:}A$, x may not already be declared in Γ. Furthermore, we identify contexts which differ only in their order, in other words, contexts are treated as sets of declarations for distinct variables. We promote "," to denote *disjoint* set union. As usual we identify terms which differ only in the names of their bound variables. We restrict attention to well-typed terms, omitting the standard typing rules.

In applications such a logic programming or logical frameworks, λ-abstraction is used to represent binding operators in some object language. In such a situation the most normal forms are long $\beta\eta$-normal forms (which we call *canonical forms*), since the canonical forms are almost always the terms in bijective correspondence with the objects we are trying to represent. Every well-typed term in the simply-typed λ-calculus has a unique canonical form—a property which persists in the strict λ-calculus introduced in Chapter 3. See that chapter for further discussion and an inductive definition of canonical forms.

We denote existential variables of type A (also called logical variables, meta-variables, or pattern variables) by E_A, although we mostly omit the type A when it is clear from the context. We think of existential variables as syntactically distinct from bound variables or free variables declared in a context.

Semantically, an existential variable E_A stands for all canonical terms M of type A in the empty context with respect to a given signature. We extend this to arbitrary well-typed terms in the usual way, and write $\|M\|$ for the set of canonical ground instances of a term M possibly containing existential variables (formally defined in Figure 4.2). In this setting, unification of two patterns corresponds to an intersection of the set of terms they denote [Mil91, Pfe91b]. This set is always either empty, or can be expressed again as the set of instances of a single pattern. That is, patterns admit most general unifiers.

We now introduce the generalization of the Not algorithm to the fully-applied case:

Definition 2.4 (Fully applied higher-order pattern complement) *Fix a signature Σ. For a fully applied higher-order linear pattern in canonical form M, define $\Gamma \vdash \mathrm{Not}(M : A)$ as:*

$$\begin{array}{lcl}
\Gamma \vdash \mathrm{Not}(E\ \overline{x_n} : a) & = & \emptyset \\
\Gamma \vdash \mathrm{Not}(h\ M_1 \ldots M_m : a) & = & \mathit{diff}_\Gamma(h : \overline{A_m} \to a) \cup \\
 & & \{h\ (Z_1\ \Gamma) \ldots (Z_{i-1}\ \Gamma)\ N\ (Z_{i+1}\ \Gamma) \ldots (Z_m\ \Gamma) \mid \\
 & & N \in \Gamma \vdash \mathrm{Not}(M_i : A_i), 1 \le i \le m\} \\
\Gamma \vdash \mathrm{Not}(\lambda x{:}A.\, M : A \to B) & = & \{\lambda x{:}A.\, N \mid N \in (\Gamma, x : A \vdash \mathrm{Not}(M : B))\}
\end{array}$$

where $m \ge 0$, $(Z\ \ \Gamma)$ denotes that a fresh variable Z of appropriate typing may depend on variables in dom(Γ) and

$$\mathit{diff}_\Gamma(h : \overline{A_m} \to a) \quad = \quad \{g\ (Z_1\ \Gamma) \ldots (Z_n\ \Gamma) \mid g \in \Sigma \cup \Gamma, g : \overline{A_n} \to a, n \ge 0, h \not\equiv g\}$$

Note that the definition makes an essential use of the fact that M is canonical and thus its matrix has atomic type a.

Remark 2.5 *For* $h \in \Gamma \cup \Sigma, \Gamma \vdash \mathrm{Not}(h : a) = \mathit{diff}_\Gamma(h : a)$.

Proof: Consider the 0-ary application case. □

We will suppress mention of the type in $\Gamma \vdash \mathrm{Not}(M : A)$ when it can be inferred from the context.

Example 2.6 *Consider the untyped* λ*-calculus*[2].

$$e \quad ::= \quad x \mid \Lambda x.\, e \mid e_1\, e_2$$

We encode these expressions using the usual techniques of higher-order abstract syntax (see, for example, [MP91]) as canonical forms over the following signature Σ_{lam}.

$$\begin{array}{rcl} \mathit{exp} & : & \mathit{type} \\ \mathit{lam} & : & (\mathit{exp} \to \mathit{exp}) \to \mathit{exp} \\ \mathit{app} & : & \mathit{exp} \to \mathit{exp} \to \mathit{exp} \end{array}$$

The representation function is given by:

$$\begin{array}{rcl} \ulcorner x \urcorner & = & x : \mathit{exp} \\ \ulcorner \Lambda x.\, e \urcorner & = & \mathit{lam}\ (\lambda x\!:\!\mathit{exp}.\ulcorner e \urcorner) \\ \ulcorner e_1\ e_2 \urcorner & = & \mathit{app}\ \ulcorner e_1 \urcorner\ \ulcorner e_2 \urcorner \end{array}$$

As usual with higher-order abstract syntax [PE88], we identify the name of (bound) variables in both languages. The adequacy of the encoding bijectively relates α*-equivalence classes of object-level terms with* $\beta\eta$*-equivalence classes at the meta-level. Now, suppose we want to negate the identity predicate on unary function types:*

$$id(\lambda x\!:\!exp.\, x).$$

The intuitive answer is as follows:

$$\begin{array}{l} \neg id(\lambda x\!:\!exp.\, app\ (E_1\ x)(E_2\ x)). \\ \neg id(\lambda x\!:\!exp.\, lam\ (\lambda y\!:\!exp.\, (E\ x\ y)). \end{array}$$

This follows from the computation of $\cdot \vdash \mathrm{Not}(\lambda x\!:\!exp.\, x)$:

$$\begin{array}{rcl} \cdot \vdash \mathrm{Not}(\lambda x\!:\!exp.\, x) & = & \{\lambda x\!:\!exp.\, Z \mid Z \in (x{:}exp \vdash \mathrm{Not}(x))\} \\ & = & \{\lambda x\!:\!exp.\, Z \mid Z \in \mathit{diff}_{x:exp}(x{:}exp))\} \\ & = & \{\lambda x\!:\!exp.\, Z \mid Z \in \{app\ (E_1 x)(E_2 x), lam\ (\lambda y\!:\!exp.\, (E\ x\ y))\}\} \\ & = & \{\lambda x\!:\!exp.\, app\ (E_1\ x)(E_2\ x), \lambda x\!:\!exp.\, lam\ (\lambda y.\ (E\ x\ y))\} \end{array}$$

For another illustration, consider the representation of an object-language β*-redex:*

$$\ulcorner (\Lambda x.\, e)\ f \urcorner \quad = \quad app\ (lam\ (\lambda x\!:\!exp.\ulcorner e \urcorner))\ \ulcorner f \urcorner$$

where $\ulcorner e \urcorner$ *my have free occurrences of* x. *When written as a pattern with variables* $E_{exp\to exp}$ *and* F_{exp} *ranging over* closed *terms, this is expressed as* $app\ (lam\ (\lambda x\!:\!exp.\, (E\ x))\ F)$. *Consider the predicate:*

$$betardx(app\ (lam\ (\lambda x\!:\!exp.\, E\ x))\ F).$$

The complement of the arguments in the empty context contains every top-level λ*-abstraction plus every application where the first argument is not an abstraction:*

$$\cdot \vdash \mathrm{Not}(app\ (lam\ (\lambda x\!:\!exp.\, (E\ x))\ F)) = \{lam\ (\lambda x\!:\!exp.\, Z\ x), app\ (app\ Z_1\ Z_2)\ Z_3\}$$

[2] We use Λ for lambda abstraction in the object-calculus, not to be confused with λ in the meta-language.

Thus the negation of the `betardx` *predicate is:*

$$\neg betardx(lam\ (\lambda x{:}exp.\, Z\ x)).$$
$$\neg betardx(app\ (app\ Z_1\ Z_2)\ Z_3).$$

If the term to complement is complex, we need to call the Not algorithm recursively as many times as its depth. Let us see another slightly more complicated example.

Example 2.7 *Consider the signature of numerals and the problem* $\cdot \vdash \mathrm{Not}(\lambda f{:}n \to n.\, f\ 0)$.

$$\begin{array}{rcl} \cdot \vdash \mathrm{Not}(\lambda f{:}n \to n.\, f\ 0) & = & \{\lambda f{:}n \to n.\, Z \mid Z \in (f{:}n \to n \vdash \mathrm{Not}(f\ 0))\} \\ & = & \{\lambda f{:}n \to n.\, Z \mid Z \in \mathit{diff}_f(f) \cup \{f\ Z' \mid Z' \in (f{:}n \to n \vdash \mathrm{Not}(0))\}\} \\ & = & \{\lambda f{:}n \to n.\, Z \mid Z \in \{0, s(Z_1\ f)\} \cup \{f\ Z' \mid N' \in \{f(Z_2\ f), s(N_3\ f)\}\} \\ & = & \{\lambda f{:}n \to n.\, 0, \lambda f{:}n \to n.\, s(Z_1\ f), \lambda f{:}n \to n.\, f(f(Z_2\ f)), \lambda f.\, f(s(Z_3\ f))\} \end{array}$$

2.5 Partially Applied Terms

Consider a predicate on the signature Σ_{lam} true if a unary (object-level) function $\ulcorner \Lambda x.\, e \urcorner$ does not depend on its argument. This can be encoded using a pattern variable E_{exp} which does not depend on x.

$$vacous(\lambda x{:}exp.\, E). \tag{2.2}$$

Intuitively, the complement should be a predicate, say `strict`, true when the function *does* use its arguments. Note that there is no finite set of patterns which has as its ground instances exactly those terms M which depend on a given variable x. One way to express it is as follows:

$$\begin{array}{l} strict(\lambda x{:}exp.\, x). \\ strict(\lambda x{:}exp.\, app\ (E_1\ x)(E_2\ x)) \\ \qquad \leftarrow strict(\lambda x{:}exp.\, E_1\ x). \\ strict(\lambda x{:}exp.\, app\ (E_1\ x)(E_2\ x)) \\ \qquad \leftarrow strict(\lambda x{:}exp.\, E_2\ x). \\ strict(\lambda x{:}exp.\, lam\ (\lambda y{:}exp.\, (E\ x\ y))) \\ \qquad \leftarrow (\forall z{:}exp.\, strict(\lambda x{:}exp.\, (E\ x\ z))). \end{array}$$

Hence the complement of a fact, whose arguments are partially applied patterns, may lead to possibly hypothetical and parametric clauses.

Example 2.8 *The encoding of an* η*-redex takes the form:*

$$\ulcorner \Lambda x.\, e\ x \urcorner \quad = \quad lam(\lambda x{:}exp.\, app\ \ulcorner e \urcorner\ x)$$

where $\ulcorner e \urcorner$ *may contain no free occurrence of* x. *The side condition is again expressed in a pattern by introducing an existential variable* E_{exp} *which* does not *depend on* x, *that is* $lam(\lambda x : exp.\, (app\ E)\ x)$. *Hence, its complement with respect to the empty context should contain, among others, also all terms*

$$lam\ (\lambda x{:}exp.\, app\ (F\ x)\ Z)$$

where F must *depend on* x.

More generally, we would have to decorate programs with predicates discriminating when a pattern is fully applied or not. It is clear that the simply typed λ-calculus, or, for that matter, every other intuitionistic type theory is not strong enough to represent the complement of partially applied patterns. This failure of closure under complementation cannot be avoided similarly to the way in which left-linearization bypasses the limitation to linear terms and it needs to be addressed directly.

One approach is taken by Lugiez [Lug95]: he modifies the language of terms to promote constraints to first-class objects, similarly in spirit to explicit substitutions. For example $\lambda xyz.\, M\{1,3\}$ would denote a function which depends on its first and third argument. The technical handling of those objects then becomes awkward as they require specialized rules which are foreign to the issues of complementation.

We can instead internalize the (in)dependence constraints in a type theory which explicitly take into account the occurrence issue. Since our underlying λ-calculus is typed, we use typing to express that a function *must* or *must not* depend on its argument. Following standard terminology, we call such terms *strict in* x and the corresponding function $\lambda x : A.\, M$ a *strict function*. The natural choice is a calculus of strict types and is formalized in Section 3.1. We first give an informal presentation.

We can introduce a strict application primitive constructor, say $F\ {}^{\#}x$, which express the fact that F *must* use an argument x mentioned in the context. Thus for example:

$$\cdot \vdash \mathrm{Not}(\lambda e{:}exp.\, E) = \lambda x{:}exp.\, F\ {}^{\#}x$$

Therefore, the complement of (2.2) would be:

$$\neg vacous(\lambda x{:}exp.\, F\ {}^{\#}x).$$

Conversely, taking the complement of terms where arguments *must* occur yields terms where some argument *must not* occur: for example

$$\begin{aligned}\cdot \vdash \mathrm{Not}(\lambda x{:}exp.\, \lambda y{:}exp.\, E\ {}^{\#}y\ {}^{\#}x) \quad &= \quad \{\lambda x{:}exp.\, \lambda y{:}exp.\, F, \lambda x{:}exp.\, \lambda y{:}exp.\, F_1{}^{\#}x,\\ &\qquad \lambda x{:}exp.\, \lambda y{:}exp.\, F_2{}^{\#}y\}\end{aligned}$$

Let $\overline{x_n}, \overline{y_m}$ be two sets of variables such that $\overline{y_m} \subseteq \overline{x_n}$. Let $\overline{z_p} = \overline{x_n} - \overline{y_m}$. We first treat the intuitionistic application case. The complement of $\lambda\overline{x_n}.\, E\ \overline{y_m}$ is the set of terms that *may* depend on $\overline{y_m}$ but *has* to depend on one of the $\overline{z_p}$'s, that is p terms such that, for an appropriate context Γ:

$$\Gamma \vdash \mathrm{Not}(E\ \overline{y_m}) \quad = \quad \bigcup_{j=1}^{p} \{E_j\ \overline{y_m}\ z_1 \ldots z_{j-1}\ {}^{\#}z_j\ z_{j+1} \ldots z_p\}$$

The complement of terms with strict application is defined as:

$$\Gamma \vdash \mathrm{Not}(E\ {}^{\#}y_1{}^{\#}y_2 \ldots {}^{\#}y_m) \quad = \quad \bigcup_{j=1}^{m} \{E_j\ y_1 \ldots y_{j-1}\ y_{j+1} \ldots y_m\}$$

Example 2.9

$$\begin{aligned}\cdot \vdash \mathrm{Not}(\lambda xyzw.\, E\ y\ w) \quad &= \quad \{\lambda xyzw.\, F_1\ y\ w\ {}^{\#}x\ z, \lambda xyzw.\, F_2\ y\ w\ x\ {}^{\#}z\}\\ \cdot \vdash \mathrm{Not}(\lambda xy.\, E\ {}^{\#}x\ {}^{\#}y) \quad &= \quad \{\lambda xy.\, F_1\ x, \lambda xy.\, F_2\ y\}\end{aligned}$$

Yet, there is a certain asymmetry between strict and intuitionistic application: while the former completely determines the occurrence status of a variable, the latter leaves the status floating indeterminately between the possibility of occurrence or not. We can benefit from a notation which captures the 'non-occurring' condition explicitly. One possibility is to decorate bound variables as well as function types with three *occurrence annotations* $1, 0, u$ with the intended meaning of:

$$\begin{aligned}x^1 &\ :\ x \text{ must occur}\\ x^0 &\ :\ x \text{ must } \textit{not} \text{ occur}\\ x^u &\ :\ x \text{ is undetermined}\end{aligned}$$

Its intended semantics, as a type theory, is explored next.

Chapter 3

A Strict λ-Calculus

In this Chapter we introduce a strict λ-calculus and develop its basic properties, culminating in the existence of canonical forms 3.2. Chapter 4 will introduce a restriction of the language for which complementation is possible.

3.1 Strict Types

As we have seen in the preceding Chapter, the complement of a partially applied pattern in the simply-typed λ-calculus cannot be expressed in a finitary manner within the same calculus. We thus generalize our language to include *strict functions* of type $A \stackrel{1}{\to} B$ (which are guaranteed to depend on their argument) and *invariant functions* of type $A \stackrel{0}{\to} B$ (which are guaranteed **not** to depend on their argument). Of course, any concretely given function either will or will not depend on its argument, but in the presence of existential variables we still need the ability to remain uncommitted. Therefore our calculus also contains the full function space $A \stackrel{u}{\to} B$. A similar calculus has been independently investigated in [Wri91, BF93]: for a comparison see Section 3.3.

$$\begin{array}{rcll} \textit{Labels} & k & ::= & 1 \mid 0 \mid u \\ \textit{Types} & A & ::= & a \mid A_1 \stackrel{k}{\to} A_2 \\ \textit{Terms} & M & ::= & c \mid x \mid \lambda x^k{:}A.\ M \mid (M_1\ M_2)^k \\ \textit{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A \end{array}$$

Note that there are three different forms of abstractions and applications, where the latter are distinguished by different labels on the argument. It is not really necessary to distinguish three forms of application syntactically, since the type of function determines the status of the application, but it is convenient for our purposes. If a label is u it is called *undetermined*, otherwise it is *determined* and denoted with the metavariable d.

We use a formulation of the typing judgment with three zones, containing the unrestricted, irrelevant and strict hypotheses, denoted by Γ, Ω, and Δ, respectively.

$$\Gamma; \Omega; \Delta \vdash M : A$$

We implicitly assume a fixed signature Σ which would otherwise clutter the presentation. Recall that Γ_1, Γ_2 is a union of two contexts which do not declare any common variables. Recall also that we consider contexts as sets, that is, exchange is left implicit.

Our system is biased towards a bottom-up reading of the rules in that variables never disappear, i.e. they are always propagated from the conclusion to the premises, although their status might be changed.

Let us go through the typing rules in detail. The requirement for the strict context Δ to be empty in the Id^u and Id^1 rules expresses that strict variables must be used, while undetermined variables in Γ or irrelevant variables in Ω can be ignored. Note that there is no rule for irrelevant variables, which expresses that they cannot be used.

$$\frac{c{:}A \in \Sigma}{\Gamma;\Omega;\cdot \vdash c : A}\,Con$$

$$\frac{}{(\Gamma, x{:}A);\Omega;\cdot \vdash x : A}\,Id^u \qquad \textit{no } Id^0 \textit{ rule} \qquad \frac{}{\Gamma;\Omega; x{:}A \vdash x : A}\,Id^1$$

$$\frac{(\Gamma, x{:}A);\Omega;\Delta \vdash M : B}{\Gamma;\Omega;\Delta \vdash \lambda x^u{:}A.\ M : A \xrightarrow{u} B}\xrightarrow{u} I$$

$$\frac{\Gamma;(\Omega, x{:}A);\Delta \vdash M : B}{\Gamma;\Omega;\Delta \vdash \lambda x^0{:}A.\ M : A \xrightarrow{0} B}\xrightarrow{0} I$$

$$\frac{\Gamma;\Omega;(\Delta, x{:}A) \vdash M : B}{\Gamma;\Omega;\Delta \vdash \lambda x^1{:}A.\ M : A \xrightarrow{1} B}\xrightarrow{1} I$$

$$\frac{\Gamma;\Omega;\Delta \vdash M : A \xrightarrow{u} B \qquad (\Gamma,\Delta);\Omega;\cdot \vdash N : A}{\Gamma;\Omega;\Delta \vdash M\ N^u : B}\xrightarrow{u} E$$

$$\frac{\Gamma;\Omega;\Delta \vdash M : A \xrightarrow{0} B \qquad (\Gamma,\Omega,\Delta);\cdot;\cdot \vdash N : A}{\Gamma;\Omega;\Delta \vdash M\ N^0 : B}\xrightarrow{0} E$$

$$\frac{(\Gamma,\Delta_N);\Omega;\Delta_M \vdash M : A \xrightarrow{1} B \qquad (\Gamma,\Delta_M);\Omega;\Delta_N \vdash N : A}{\Gamma;\Omega;(\Delta_M,\Delta_N) \vdash M\ N^1 : B}\xrightarrow{1} E$$

Figure 3.1: Typing rules for $\lambda^{\xrightarrow{l}}$

$$
\dfrac{\dfrac{\dfrac{}{y{:}A;\cdot;x{:}A \overset{1}{\to} A \overset{1}{\to} B \vdash x : A \overset{1}{\to} A \overset{1}{\to} B}\,Id^1 \qquad \dfrac{}{x{:}A \overset{1}{\to} A \overset{1}{\to} B;\cdot;y{:}A \vdash y : A}\,Id^1}{\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash xy^1 : A \overset{1}{\to} B}\overset{1}{\to} E \qquad \dfrac{}{(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A);\cdot;\cdot \vdash y : A}\,Id^u}{\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash (xy^1)y^1 : B}\overset{1}{\to} E
$$

Figure 3.2: First derivation of $\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash (xy^1)y^1 : B$

$$
\dfrac{\dfrac{\dfrac{}{y{:}A;\cdot;x{:}A \overset{1}{\to} A \overset{1}{\to} B \vdash x : A \overset{1}{\to} A \overset{1}{\to} B}\,Id^1 \qquad \dfrac{}{(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A);\cdot;\cdot \vdash y : A}\,Id^u}{y{:}A;\cdot;x{:}A \overset{1}{\to} A \overset{1}{\to} B \vdash xy^1 : A \overset{1}{\to} B}\overset{1}{\to} E \qquad \dfrac{}{x{:}A \overset{1}{\to} A \overset{1}{\to} B;\cdot;y{:}A \vdash y : A}\,Id^1}{\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash (xy^1)y^1 : B}\overset{1}{\to} E
$$

Figure 3.3: Second derivation of $\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash (xy^1)y^1 : B$

The introduction rules for undetermined, invariant, and strict functions simply add a variable to the appropriate context and check the body of the function.

The difficult rules are the three elimination rules. First, the undetermined context Γ is always propagated to both premises. This reflects that we place no restriction on the use of these variables.

Next we consider the strict context Δ. Recall that this contains the variables which should occur strictly in a term. An undetermined function $M : A \overset{u}{\to} B$ may or may not use its argument. An occurrence of a variable in the argument to such a function can therefore not be guaranteed to be used. Hence we must require in the rule $\overset{u}{\to} E$ for an application $M\ N^u$ that all variables in Δ occur strictly in M. This ensures at least one strict occurrence in M and no further restrictions on occurrences of strict variables in the argument are necessary. This is reflected in the rule by adding Δ to the undetermined context while checking the argument N. The treatment of the strict variables in the vacuous application $M\ N^0$ is similar.

In the case of a strict application $M\ N^1$ each strict variable should occur strictly in either M or N. We therefore split the context into Δ_M and Δ_N guaranteeing that each variable has at least one strict occurrence in M or N, respectively. However, strict variables can occur more than once, so variables from Δ_N can be used freely in M, and variables from Δ_M can occur freely in N. As before, we reflect this by adding these variables to the undetermined context.

Finally we consider the irrelevant context Ω. Variables declared in Ω cannot be used *except* in the argument to an irrelevant function (which is guaranteed to ignore its argument). We therefore add the irrelevant context Ω to the undetermined context when checking the argument of a vacuous application $M\ N^0$.

We now illustrate how the strict application rule non-deterministically splits contexts. Consider the typing problem $\cdot;\cdot;(x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A) \vdash (xy^1)y^1 : B$. There are four ways to split the strict context:

$$
\begin{array}{ll}
\Delta_M = x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A & \Delta_N = \cdot \\
\Delta_M = x{:}A \overset{1}{\to} A \overset{1}{\to} B & \Delta_N = y{:}A \\
\Delta_M = y{:}A & \Delta_N = x{:}A \overset{1}{\to} A \overset{1}{\to} B \\
\Delta_M = \cdot & \Delta_N = x{:}A \overset{1}{\to} A \overset{1}{\to} B, y{:}A
\end{array}
$$

Only the first two yield a valid derivation (depicted in Figure 3.2 and Figure 3.3), as x needs to be strict in the leftmost branch.

Our strict λ-calculus satisfies the expected properties, culminating in the existence of canonical forms which is critical for the intended applications. We begin with the following:

Remark 3.1 (Inversion) *All rules in $\lambda^{\overset{!}{\to}}$ are invertible.*

We will often use inversion principles tacitly in proofs by structural induction on the typing derivation.

Note that, although typing derivation may not, typing is unique.

Theorem 3.2 (Uniqueness of Typing) *If $\Gamma;\Omega;\Delta \vdash M : A$ and $\Gamma';\Omega';\Delta \vdash M : A'$, then $A \equiv A'$.*

Proof: By induction on the structure of the given derivations, exploiting 'functionality' of signatures and contexts. □

We start addressing the structural properties of the context(s). Exchange is directly built into the formulation and will not be repeated.

Theorem 3.3 (Weakening)

1. *(Weakening^u) If* $\Gamma; \Omega; \Delta \vdash M : A$, *then* $(\Gamma, x{:}C); \Omega; \Delta \vdash M : A$.
2. *(Weakening^0) If* $\Gamma; \Omega; \Delta \vdash M : A$, *then* $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$.

Proof: By induction on the structure of the given derivation. □

The following properties allow us to lose track of strict and vacuous occurrences, if we are so inclined. We use the phrase 'by sub-derivation' to localize the immediate sub-derivation(s) of a given one; 'by rule' means in this Chapter by application of the correct (and unique) typing rule, when not explicitly mentioned.

Theorem 3.4 *If* $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$, *then* $(\Gamma, x{:}C); \Omega; \Delta \vdash M : A$.

Proof: By induction on the structure of $\mathcal{D} :: \Gamma; (\Omega, x{:}C); \Delta \vdash M : A$.

Case:

$$\mathcal{D} = \frac{c{:}A \in \Sigma}{\Gamma; (\Omega, x{:}C); \cdot \vdash c : A}\, Idc$$

Then

$$\mathcal{E} = \frac{c{:}A \in \Sigma}{(\Gamma, x{:}C); \Omega; \cdot \vdash c : A}\, Idc$$

Case:

$$\mathcal{D} = \frac{}{(\Gamma, y{:}A); (\Omega, x{:}C); \cdot \vdash y : A}\, Id^u$$

Then

$$\mathcal{E} = \frac{}{(\Gamma, y{:}A, x{:}C); \Omega; \cdot \vdash y : A}\, Id^u$$

Case:

$$\mathcal{D} = \frac{}{\Gamma; (\Omega, x{:}C); y{:}A \vdash y : A}\, Id^1$$

Then

$$\mathcal{E} = \frac{}{(\Gamma, x{:}C); \Omega; y{:}A \vdash y : A}\, Id^1$$

Case:

$$\mathcal{D} = \frac{\begin{array}{c}\mathcal{D}' \\ \Gamma; (\Omega, x{:}C, y{:}A); \Delta \vdash M : B\end{array}}{\Gamma; (\Omega, x{:}C); \Delta \vdash (\lambda y^0 {:} A.\, M) : A \xrightarrow{0} B}\, \xrightarrow{0} I$$

$\Gamma; (\Omega, x{:}C, y{:}A); \Delta \vdash M : B$	By sub-derivation
$(\Gamma, x{:}C); (\Omega, y{:}A); \Delta \vdash M : B$	By IH on $\mathcal{D}'$
$(\Gamma, x{:}C); \Omega; \Delta \vdash (\lambda y^0 {:} A.\, M) : A \xrightarrow{0} B$	By rule $\xrightarrow{0} I$

Case: $\mathcal{D}$ ends in $\xrightarrow{u} I$ or $\xrightarrow{1} I$: the claim follows from an immediate appeal to the inductive hypothesis, as in the above case.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma;(\Omega,x{:}C);\Delta \vdash M : A \xrightarrow{u} B & (\Gamma,\Delta);(\Omega,x{:}C);\cdot \vdash N : A \end{array}}{\Gamma;(\Omega,x{:}C);\Delta \vdash M\ N^u : B} \xrightarrow{u} E$$

$\mathcal{D}_1 :: \Gamma;(\Omega,x{:}C);\Delta \vdash M : A \xrightarrow{u} B$	By sub-derivation
$(\Gamma,x{:}C);\Omega;\Delta \vdash M : A \xrightarrow{u} B$	By IH on $\mathcal{D}_1$
$\mathcal{D}_2 :: (\Gamma,\Delta);(\Omega,x{:}C);\cdot \vdash N : A$	By sub-derivation
$(\Gamma,\Delta,x{:}C);\Omega;\cdot \vdash N : A$	By IH on $\mathcal{D}_2$
$(\Gamma,x{:}C);\Omega;\Delta \vdash M\ N^u : B$	By rule $\xrightarrow{u} E$

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma;(\Omega,x{:}C);\Delta \vdash M : A \xrightarrow{0} B & (\Gamma,\Omega,\Delta,x{:}C);\cdot;\cdot \vdash N : A \end{array}}{\Gamma;(\Omega,x{:}C);\Delta \vdash M\ N^0 : B} \xrightarrow{0} E$$

$\Gamma;(\Omega,x{:}C);\Delta \vdash M : A \xrightarrow{0} B$	By sub-derivation
$(\Gamma,x{:}C);\Omega;\Delta \vdash M : A \xrightarrow{0} B$	By IH on $\mathcal{D}_1$
$(\Gamma,\Omega,\Delta,x{:}C);\cdot;\cdot \vdash N : A$	By sub-derivation
$(\Gamma,x{:}C);\Omega;\Delta \vdash M\ N^0 : B$	By rule $\xrightarrow{0} E$

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ (\Gamma,\Delta_N);(\Omega,x{:}C);\Delta_M \vdash M : A \xrightarrow{1} B & (\Gamma,\Delta_M);(\Omega,x{:}C);\Delta_N \vdash N : A \end{array}}{\Gamma;(\Omega,x{:}C);(\Delta_N,\Delta_M) \vdash M\ N^1 : B} \xrightarrow{1} E$$

$(\Gamma,\Delta_N);(\Omega,x{:}C);\Delta_M \vdash M : A \xrightarrow{1} B$	By sub-derivation
$(\Gamma,x{:}C,\Delta_N);\Omega;\Delta_M \vdash M : A \xrightarrow{1} B$	By IH on $\mathcal{D}_1$
$(\Gamma,\Delta_M);(\Omega,x{:}C);\Delta_N \vdash N : A$	By sub-derivation
$(\Gamma,x{:}C,\Delta_M);\Omega;\Delta_N \vdash N : A$	By IH on $\mathcal{D}_2$
$(\Gamma,x{:}C);\Omega;(\Delta_N,\Delta_M) \vdash M\ N^1 : B$	By rule $\xrightarrow{1} E$

□

Corollary 3.5 (Loosening^0) *If* $\Gamma;(\Omega,\Phi);\Delta \vdash M : A$, *then* $(\Gamma,\Phi);\Omega;\Delta \vdash M : A$.

Proof: By repeated application of Theorem 3.4. □

Theorem 3.6 *If* $\Gamma;\Omega;(\Delta,x{:}C) \vdash M : A$, *then* $(\Gamma,x{:}C);\Omega;\Delta \vdash M : A$.

Proof: By induction on the structure of $\mathcal{D} :: \Gamma;\Omega;(\Delta,x{:}C) \vdash M : A$.

Case: $\mathcal{D}$ ends in Con, Id^u: vacuously true.

Case:

$$\mathcal{D} = \dfrac{}{\Gamma;\Omega;x{:}A \vdash x : A} Id^1$$

Then

$$\mathcal{E} = \dfrac{}{(\Gamma,x{:}A);\Omega;\cdot \vdash x : A} Id^u$$

Case:

$$\mathcal{D} = \dfrac{\Gamma;\Omega;(\Delta,x{:}C,y{:}A) \vdash M : B}{\Gamma;\Omega;(\Delta,x{:}C) \vdash (\lambda y^1{:}A.\,M) : A \xrightarrow{1} B} \xrightarrow{1} I$$

$\Gamma; \Omega; (\Delta, x{:}C, y{:}A) \vdash M : B$ By sub-derivation
$(\Gamma, x{:}C,); \Omega; (\Delta, y{:}A) \vdash M : B$ By IH
$(\Gamma, x{:}C); \Omega; \Delta \vdash (\lambda y^1 : A.\, M) : A \xrightarrow{1} B$ By rule

Case: $\mathcal{D}$ ends in $\xrightarrow{u} I$ or $\xrightarrow{0} I$: the thesis follow from an immediate appeal to the inductive hypothesis as in the above case.

Case:

$$\mathcal{D} = \frac{\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A \xrightarrow{u} B \qquad (\Gamma, \Delta, x{:}C); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; (\Delta, x{:}C) \vdash M\ N^u : B} \xrightarrow{u} E$$

$\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A \xrightarrow{u} B$ By sub-derivation
$(\Gamma, x{:}C); \Omega; \Delta \vdash M : A \xrightarrow{u} B$ By IH
$(\Gamma, \Delta, x{:}C), \Omega; \cdot \vdash N : A$ By sub-derivation
$(\Gamma, x{:}C); \Omega; \Delta \vdash M\ N^u : B$ By rule

Case:

$$\mathcal{D} = \frac{\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A \xrightarrow{0} B \qquad (\Gamma, \Omega, \Delta, x{:}C); \cdot; \cdot \vdash N : A}{\Gamma; \Omega; (\Delta, x{:}C) \vdash M\ N^0 : B} \xrightarrow{0} E$$

$\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A \xrightarrow{0} B$ By sub-derivation
$(\Gamma, x{:}C); \Omega; \Delta \vdash M : A \xrightarrow{0} B$ By IH
$(\Gamma, \Omega, \Delta, x{:}C); \cdot; \cdot \vdash N : A$ By sub-derivation
$(\Gamma, x{:}C); \Omega; \Delta \vdash M\ N^0 : B$ By rule

Case: $\mathcal{D}$ ends in $\xrightarrow{1}$. There are two sub-cases:

$x \in \Delta_M$ $\mathcal{D} =$

$$\frac{(\Gamma, \Delta_N); \Omega; (\Delta'_M, x{:}C) \vdash M : A \xrightarrow{1} B \qquad (\Gamma, \Delta'_M, x{:}C); \Omega; \Delta_N \vdash N : A}{\Gamma; \Omega; (\Delta'_M, x{:}C, \Delta_N) \vdash M\ N^1 : B} \xrightarrow{1} E$$

$(\Gamma, \Delta_N); \Omega; (\Delta'_M, x{:}C) \vdash M : A \xrightarrow{1} B$ By sub-derivation
$(\Gamma, \Delta_N, x{:}C); \Omega; \Delta'_M \vdash M : A \xrightarrow{1} B$ By IH
$(\Gamma, \Delta'_M, x{:}C); \Omega; \Delta_N \vdash N : A$ By sub-derivation
$(\Gamma, x{:}C); \Omega; (\Delta_N, \Delta'_M) \vdash M\ N^1 : B$ By rule

$x \in \Delta_N$ Symmetrical to the above.

□

Corollary 3.7 (Loosening[1]) *If* $\Gamma; \Omega; \Delta \vdash M : A$, *then* $(\Gamma, \Delta); \Omega; \cdot \vdash M : A$.

Proof: By repeated application of Theorem 3.6. □

Theorem 3.8 (Substitution Properties)

1. *(Substitution^u) If* $(\Gamma, x{:}A); \Omega; \Delta \vdash M : C$ *and* $(\Gamma, \Delta); \Omega; \cdot \vdash N : A$, *then* $\Gamma; \Omega; \Delta \vdash [N/x]M : C$.
2. *(Substitution^0) If* $\Gamma; (\Omega, x{:}A); \Delta \vdash M : C$ *and* $(\Gamma, \Delta, \Omega); \cdot; \cdot \vdash N : A$, *then* $\Gamma; \Omega; \Delta \vdash [N/x]M : C$.
3. *(Substitution^1) If* $(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash M : C$ *and* $(\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A$, *then* $\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash [N/x]M : C$.

Proof: The proof is by mutual induction on the height of the derivation $\mathcal{D}$ of $M : C$. We show the crucial cases.

1. $\boxed{\text{Substitution}^u}$

Case:

$$\mathcal{D} = \frac{c{:}C \in \Sigma}{(\Gamma, x{:}A); \Omega; \cdot \vdash c : C}\, Con$$

As $[N/x]c = c$, then

$$\mathcal{F} = \frac{c{:}C \in \Sigma}{\Gamma; \Omega; \cdot \vdash c : C}\, Con$$

Case:

$$\mathcal{D} = \frac{}{(\Gamma', x{:}A, y{:}C); \Omega; \cdot \vdash y : C}\, Id^u$$

As $[N/x]y = y$, then

$$\mathcal{F} = \frac{}{(\Gamma', y{:}C); \Omega; \cdot \vdash y : C}\, Id^u$$

Case:

$$\mathcal{D} = \frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : C}\, Id^u$$

As $[N/x]x = N$ and $C = A$, then $\mathcal{F} = \mathcal{E} :: \Gamma; \Omega; \cdot \vdash N : C$

Case:

$$\mathcal{D} = \frac{}{(\Gamma, x{:}A); \Omega; y{:}C \vdash y : C}\, Id^1$$

As $[N/x]y = y$, then

$$\mathcal{F} = \frac{}{\Gamma; \Omega; y{:}C \vdash y : C}\, Id^1$$

Case:

$$\mathcal{D} = \frac{(\Gamma, x{:}A, y{:}B); \Omega; \Delta \vdash M : C}{(\Gamma, x{:}A); \Omega; \Delta \vdash \lambda y^u{:}B.\, M : B \overset{u}{\rightarrow} C}\, \overset{u}{\rightarrow} I$$

$(\Gamma, x{:}A, y{:}B); \Omega; \Delta \vdash M : C$	By sub-derivation
$\Gamma; \Omega; \cdot \vdash N : C$	By hypothesis
$(\Gamma, y{:}B); \Omega; \cdot \vdash N : C$	By Weakening^u
$(\Gamma, y{:}B); \Omega; \Delta \vdash [N/x]M : C$	By IH
$\Gamma; \Omega; \Delta \vdash (\lambda y^u{:}B.\, [N/x]M) : A \overset{u}{\rightarrow} B$	By rule
$\Gamma; \Omega; \Delta \vdash [N/x](\lambda y^u{:}B.\, M) : A \overset{u}{\rightarrow} B$	By subst.

Case: $\mathcal{D}$ ends with $\overset{0}{\rightarrow} I, \overset{1}{\rightarrow} I$: similarly.

Case:

$$\mathcal{D} = \frac{\begin{array}{c}\mathcal{D}_1\\(\Gamma, x{:}A); \Omega; \Delta \vdash P : B \overset{u}{\rightarrow} C\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\(\Gamma, \Delta, x{:}A); \Omega; \cdot \vdash Q : B\end{array}}{(\Gamma, x{:}A); \Omega; \Delta \vdash P\ Q^u : C}\, \overset{u}{\rightarrow} E$$

$(\Gamma, x{:}A); \Omega; \Delta \vdash P : B \overset{u}{\rightarrow} C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta); \Omega; \cdot \vdash N : A$	By hypothesis
$\Gamma; \Omega; \Delta \vdash [N/x]P : B \overset{u}{\rightarrow} C$	By IH on $\mathcal{D}_1$ and $\mathcal{E}$
$(\Gamma, x{:}A, \Delta); \Omega; \cdot \vdash Q : B$	By sub-derivation
$(\Gamma, \Delta); \Omega; \cdot \vdash [N/x]Q : B$	By IH on $\mathcal{D}_2$ and $\mathcal{E}$
$\Gamma; \Omega; \Delta \vdash ([N/x]P)([N/x]Q)^u : C$	By rule
$\Gamma; \Omega; \Delta \vdash [N/x](P\ Q^u) : C$	By substitution.

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ (\Gamma, x{:}A); \Omega; \Delta \vdash P : B \xrightarrow{0} C\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ (\Gamma, \Delta, \Omega, x{:}A); \cdot; \cdot \vdash Q : B\end{matrix}}{(\Gamma, x{:}A); \Omega; \Delta \vdash P\ Q^0 : C} \xrightarrow{0} E$$

$(\Gamma, x{:}A); \Omega; \Delta \vdash P : B \xrightarrow{0} C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta); \Omega; \cdot \vdash N : A$	By hypothesis
$\Gamma; \Omega; \Delta \vdash [N/x]P : B \xrightarrow{0} C$	By IH on $\mathcal{D}_1$ and $\mathcal{E}$
$(\Gamma, x{:}A, \Omega, \Delta); \cdot; \cdot \vdash Q : B$	By sub-derivation
$\mathcal{E}' :: (\Gamma, \Delta, \Omega); \cdot; \cdot \vdash N : A$	By Loosening^0 Ω in $\mathcal{E}$
$(\Gamma, \Delta, \Omega); \cdot; \cdot \vdash [N/x]Q : B$	By IH on $\mathcal{D}_2$ and $\mathcal{E}'$
$\Gamma; \Omega; \Delta \vdash [N/x](P\ Q)^0 : C$	By rule

Case: $\mathcal{D} =$

$$\dfrac{\begin{matrix}\mathcal{D}_1 \\ (\Gamma, x{:}A, \Delta_Q); \Omega; \Delta_P \vdash P : B \xrightarrow{1} C\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ (\Gamma, x{:}A, \Delta_P); \Omega; \Delta_Q \vdash Q : B\end{matrix}}{(\Gamma, x{:}A); \Omega; (\Delta_P, \Delta_Q) \vdash P\ Q^1 : C} \xrightarrow{1} E$$

$(\Gamma, x{:}A, \Delta_Q); \Omega; \Delta_P \vdash P : B \xrightarrow{1} C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta_P, \Delta_Q); \Omega; \cdot \vdash N : A$	By hypothesis
$(\Gamma, \Delta_Q); \Omega; \Delta_P \vdash [N/x]P : B \xrightarrow{1} C$	By IH on $\mathcal{D}_1$ and $\mathcal{E}$
$(\Gamma, x{:}A, \Delta_P); \Omega; \Delta_Q \vdash Q : B$	By sub-derivation
$(\Gamma, \Delta_P); \Omega; \Delta_Q \vdash [N/x]Q : B$	By IH on $\mathcal{D}_2$ and $\mathcal{E}$
$\Gamma; \Omega; (\Delta_P, \Delta_Q) \vdash [N/x](P\ Q)^1 : C$	By rule

2. $\boxed{\text{Substitution}^0}$ Similar to the above.

3. $\boxed{\text{Substitution}^1}$

Case: $\mathcal{D}$ ends in Con, Id^u: vacuously true.

Case:

$$\mathcal{D} = \dfrac{}{(\Gamma, \Delta_N); \Omega; x{:}A \vdash x : A} Id^1$$

and $[N/x]x = N$; then $\Delta_M = \cdot$, $\mathcal{F} = \mathcal{E} :: \Gamma; \Omega; \Delta_N \vdash N$.

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ (\Gamma, \Delta_N); (\Omega, y{:}B); (\Delta_M, x{:}A) \vdash Q : C\end{matrix}}{(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash \lambda y^0{:}B.\, Q : B \xrightarrow{0} C} \xrightarrow{0} I$$

$\mathcal{D}_1 :: (\Gamma, \Delta_N); (\Omega, y{:}B); (\Delta_M, x{:}A) \vdash Q : C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A$	By hypothesis
$\mathcal{E}' :: (\Gamma, \Delta_M); (\Omega, y{:}B); \Delta_N \vdash N : A$	By Weakening^0 on $\mathcal{E}$
$\Gamma; (\Omega, y{:}B); (\Delta_M, \Delta_N) \vdash [N/x]Q : C$	By IH on $\mathcal{D}_1$ and $\mathcal{E}'$
$\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash \lambda y^0{:}B.\, [N/x]Q : B \xrightarrow{0} C$	By rule
$\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash [N/x](\lambda y^0{:}B.\, Q) : B \xrightarrow{0} C$	By substitution.

Case: $\mathcal{D}$ ends with $\xrightarrow{u} I$. Proceed similarly using Weakening^u.

Case: $\mathcal{D}$ ends with $\xrightarrow{1} I$. It follows by an immediate appeal to the IH.

Case:

$$\mathcal{D} = \dfrac{(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash P : B \xrightarrow{u} C \qquad (\Gamma, \Delta_M, x{:}A, \Delta_N); \Omega; \cdot \vdash Q : B}{(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash P\ Q^u : C} \xrightarrow{u} E$$

$\mathcal{D}_1 :: (\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}A) \vdash P : B \xrightarrow{u} C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A$	By hypothesis
$\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash [N/x]P : B \xrightarrow{u} C$	By IH on $\mathcal{D}_1$ and $\mathcal{E}$
$\mathcal{D}_2 :: (\Gamma, \Delta_M, x{:}A, \Delta_N); \Omega; \cdot \vdash Q : B$	By sub-derivation
$(\Gamma, \Delta_M, \Delta_N); \Omega; \cdot \vdash [N/x]Q : B$	By IH on $\mathcal{D}_2$ and $\mathcal{E}$
$\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash [N/x](P\ Q)^u : C$	By rule

Case: $\mathcal{D}$ ends in $\xrightarrow{0} E$: similar.

Case: $\mathcal{D}$ ends in $\xrightarrow{1} E$: there are two sub-cases , where $\Delta_M = \Delta_P, \Delta_Q$:

$x \in \Delta_P$ $\quad \mathcal{D} =$

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ (\Gamma, \Delta_N, \Delta_Q); \Omega; (\Delta'_P, x{:}A) \vdash P : B \xrightarrow{1} C\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ (\Gamma, \Delta_N, x{:}A, \Delta'_P); \Omega; \Delta_Q \vdash Q : B\end{array}}{(\Gamma, \Delta_N); \Omega; (\Delta'_P, x{:}A, \Delta_Q) \vdash P\ Q^1 : C} \xrightarrow{1} E$$

$\mathcal{D}_1 :: (\Gamma, \Delta_N, \Delta_Q); \Omega; (\Delta'_P, x{:}A) \vdash P : B \xrightarrow{1} C$	By sub-derivation
$\mathcal{E} :: (\Gamma, \Delta'_P \Delta_Q); \Omega; \Delta_N \vdash N : A$	By hypothesis
$(\Gamma, \Delta_Q); \Omega; (\Delta'_P, \Delta_N) \vdash [N/x]P : B \xrightarrow{1} C$	By IH 3 on $\mathcal{D}_1, \mathcal{E}$
$(\Gamma, \Delta_Q, \Delta_N); \Omega; \Delta'_P \vdash [N/x]P : B \xrightarrow{1} C$	By Loosening^1 Δ_N
$\mathcal{D}_2 :: (\Gamma, \Delta_N, x{:}A, \Delta'_P); \Omega; \Delta_Q \vdash Q : B$	By sub-derivation
$\mathcal{E}' :: (\Gamma, \Delta'_P, \Delta_Q, \Delta_N); \Omega; \cdot \vdash N : A$	By Loosening^1 Δ_N in $\mathcal{E}$
$(\Gamma, \Delta_N, \Delta'_P); \Omega; \Delta_Q \vdash [N/x]Q : B$	By IH 1 on $\mathcal{D}_2, \mathcal{E}'$
$\Gamma; \Omega; (\Delta'_P, \Delta_Q, \Delta_N) \vdash [N/x](P\ Q)^1 : C$	By rule

$x \in \Delta_Q$ Symmetrical to the above.

□

Theorem 3.9 (Contraction) *Let z be a fresh variable of type A:*

1. *(Contraction^u) If $(\Gamma, x{:}A, y{:}A); \Omega; \Delta \vdash M : C$, then $(\Gamma, z{:}A); \Omega; \Delta \vdash [z/y]([z/x]M) : C$.*
2. *(Contraction^0) If $\Gamma; (\Omega, x{:}A, y{:}A); \Delta \vdash M : C$, then $\Gamma; (\Omega, z{:}A); \Delta \vdash [z/y]([z/x]M) : C$.*
3. *(Contraction^1) If $\Gamma; \Omega; (\Delta, x{:}A, y{:}A) \vdash M : C$, then $\Gamma; \Omega; (\Delta, z{:}A) \vdash [z/y]([z/x]M) : C$.*

Proof:

1.

$(\Gamma, x{:}A, y{:}A); \Omega; \Delta \vdash M : C$	By assumption
$(\Gamma, x{:}A, y{:}A, z{:}A); \Omega; \Delta \vdash M : C$	By Weakening^u
$(\Gamma, y{:}A, \Delta, z{:}A); \Omega; \cdot \vdash z : A$	By rule Id^u
$(\Gamma, y{:}A, z{:}A); \Omega; \Delta \vdash [z/x]M : C$	By Substitution^u
$(\Gamma, \Delta, z{:}A); \Omega; \cdot \vdash z : A$	By rule Id^u
$(\Gamma, z{:}A); \Omega; \Delta \vdash [z/y]([z/x]M) : C$	By Substitution^u

2. Similarly, using Substitution^0.

3.

$\Gamma; \Omega; (\Delta, x{:}A, y{:}A) \vdash M : C$	By assumption
$(\Gamma, z{:}A); \Omega; (\Delta, x{:}A, y{:}A) \vdash M : C$	By Weakening^u
$(\Gamma, y{:}A, \Delta); \Omega; z{:}A \vdash z : A$	By rule Id^1
$\Gamma; \Omega; (\Delta, y{:}A, z{:}A) \vdash [z/x]M : C$	By Substitution^1
$(\Gamma, z{:}A); \Omega; (\Delta, y{:}A) \vdash [z/x]M : C$	By Loosening^1
$(\Gamma, \Delta); \Omega; z{:}A \vdash z : A$	By rule Id^1
$\Gamma; \Omega; (\Delta, z{:}A) \vdash [z/y]([z/x]M) : C$	By Substitution^1

□

The notions of reduction and expansion derive directly from the ordinary β and η rules.

$$\begin{array}{rcl} (\lambda x^k{:}A.\ M)\, N^k & \xrightarrow{\beta} & [N/x]M \\ M : A \xrightarrow{k} B & \xrightarrow{\bar\eta} & \lambda x^k{:}A.\ M\ x^k \end{array}$$

Indeed one of the main reason to introduce irrelevant variables, as ones which *may* occur but *must not* be used, is to well-type η-expansion of invariant functions:

$$M : A \xrightarrow{0} B \quad \xrightarrow{\bar\eta} \quad \lambda x^0{:}A.\ M\ x^0$$

The subject reduction and expansion theorems are an immediate consequence of the structural and substitution properties.

Theorem 3.10 (Subject Reduction) *If* $\Gamma;\Omega;\Delta \vdash M : A$ *and* $M \xrightarrow{\beta} M'$ *then* $\Gamma;\Omega;\Delta \vdash M' : A$.

Proof: By cases on k:

1. Let $M \equiv (\lambda x^u{:}B.\,P)Q^u : A$ and $M' \equiv [Q/x]P$.

$\Gamma;\Omega;\Delta \vdash (\lambda x^u{:}B.\,P)Q^u : A$	By hypothesis
$\mathcal{E} :: (\Gamma,\Delta);\Omega;\cdot \vdash Q : B$	By inversion
$\Gamma;\Omega;\Delta \vdash \lambda x^u{:}B.\,P : B \xrightarrow{u} A$	By inversion
$\mathcal{D} :: (\Gamma, x{:}B);\Omega;\Delta \vdash P : A$	By inversion
$\Gamma;\Omega;\Delta \vdash [Q/x]P : A$	By Substitution^u on $\mathcal{D},\mathcal{E}$

2. Let $M \equiv (\lambda x^0{:}B.\,P)Q^0 : A$ and $M' \equiv [Q/x]P$.

$\Gamma;\Omega;\Delta \vdash (\lambda x^0{:}B.\,P)Q^0 : A$	By hypothesis
$\mathcal{E} :: (\Gamma,\Omega,\Delta);\cdot;\cdot \vdash Q : B$	By inversion
$\Gamma;\Omega;\Delta \vdash \lambda x^0{:}B.\,P : B \xrightarrow{0} A$	By inversion
$\mathcal{D} :: \Gamma;(\Omega, x{:}B);\Delta \vdash P : A$	By inversion
$\Gamma;\Omega;\Delta \vdash [Q/x]P : A$	By Substitution^0 on $\mathcal{D},\mathcal{E}$

3. Let $M \equiv (\lambda x^1{:}B.\,P)Q^1 : A$ and $M' \equiv [Q/x]P$.

$\Gamma;\Omega;\Delta \vdash (\lambda x^1{:}B.\,P)Q^1 : A$ and $\Delta = \Delta_P,\Delta_Q$	By hypothesis
$\mathcal{E} :: (\Gamma,\Delta_P);\Omega;\Delta_Q \vdash Q : B$	By inversion
$(\Gamma,\Delta_Q);\Omega;\Delta_P \vdash \lambda x^1{:}B.\,P : B \xrightarrow{1} A$	By inversion
$\mathcal{D} :: (\Gamma,\Delta_Q);\Omega;(\Delta_P, x{:}B) \vdash P : A$	By inversion
$\Gamma;\Omega;(\Delta_P,\Delta_Q) \vdash [Q/x]P : A$	By Substitution^1 on $\mathcal{D},\mathcal{E}$

□

Theorem 3.11 (Subject Expansion) *If* $\Gamma;\Omega;\Delta \vdash M : A \xrightarrow{k} B$ *and* $M \xrightarrow{\bar\eta} M'$ *then* $\Gamma;\Omega;\Delta \vdash M' : A \xrightarrow{k} B$.

Proof: By cases on k:

1.

$\Gamma;\Omega;\Delta \vdash M : A \xrightarrow{u} B$	By hypothesis
$(\Gamma, x{:}A);\Omega;\Delta \vdash M : A \xrightarrow{u} B$	By Weakening^u
$(\Gamma, x{:}A, \Delta);\Omega;\cdot \vdash x : A$	By rule Id^u
$(\Gamma, x{:}A);\Omega;\Delta \vdash M\ x^u : B$	By rule $\xrightarrow{u} E$
$\Gamma;\Omega;\Delta \vdash \lambda x^u{:}A.\,M\ x^u : A \xrightarrow{u} B$	By rule $\xrightarrow{u} I$

$$\frac{}{(\lambda x^k{:}A.\ M)\, N^k \xrightarrow{\beta} [N/x]M} \overset{k}{\mapsto} \beta \qquad \frac{M \overset{k}{\mapsto} M'}{\lambda x^k : A.\, M \overset{k}{\mapsto} \lambda x^k : A.\, M} \overset{k}{\mapsto} \lambda$$

$$\frac{M \overset{k}{\mapsto} Q}{(M\ N)^k \overset{k}{\mapsto} (Q\ N)^k} \overset{k}{\mapsto} \nu \qquad \frac{N \overset{k}{\mapsto} Q}{(M\ N)^k \overset{k}{\mapsto} (M\ Q)^k} \overset{k}{\mapsto} \mu$$

Figure 3.4: Reduction rules for $\lambda^{\overset{l}{\rightarrow}}$

2. $\Gamma; \Omega; \Delta \vdash M : A \overset{0}{\rightarrow} B$ By hypothesis
   $\Gamma; (\Omega, x{:}A); \Delta \vdash M : A \overset{0}{\rightarrow} B$ By Weakening^0
   $(\Gamma, \Omega, x{:}A, \Delta); \cdot; \cdot \vdash x : A$ By rule Id^u
   $(\Gamma, x{:}A); \Omega; \Delta \vdash M\ x^0 : B$ By rule $\overset{0}{\rightarrow} E$
   $\Gamma; \Omega; \Delta \vdash \lambda x^0 : A.\, M\ x^0 : A \overset{0}{\rightarrow} B$ By rule $\overset{0}{\rightarrow} I$

3. $\Gamma; \Omega; \Delta \vdash M : A \overset{1}{\rightarrow} B$ By hypothesis
   $(\Gamma, x{:}A); \Omega; \Delta \vdash M : A \overset{1}{\rightarrow} B$ By Weakening^u
   $(\Gamma, \Delta); \Omega; x{:}A \vdash x : A$ By rule Id^1
   $\Gamma; \Omega; (\Delta, x{:}A) \vdash M\ x^1 : B$ By rule $\overset{1}{\rightarrow} E$
   $\Gamma; \Omega; \Delta \vdash \lambda x^1 : A.\, M\ x^1 : A \overset{1}{\rightarrow} B$ By rule $\overset{1}{\rightarrow} I$

□

We can now give the definition of *reduction* in Figure 3.4: we write $\overset{k}{\mapsto}^*$ for the reflexive and transitive closure of $\overset{k}{\mapsto}$.

Theorem 3.12 (Subject Reduction with Congruences) *If* $\Gamma; \Omega; \Delta \vdash M : A$ *and* $M \overset{k}{\mapsto}^* M'$ *then* $\Gamma; \Omega; \Delta \vdash M' : A$.

Proof: By induction on the derivation of $M \overset{k}{\mapsto}^* M'$, using Subject Reduction (Theorem 3.10) for the base cases. □

The following Lemma establishes a sort of consistency property of the type system, so that a term can be typed only by exclusive contexts. In particular we show that a term M cannot be both strict and vacuous in one variable, say x. This will be central in the proof of disjointness of term complementation (Theorem 4.20):

Lemma 3.13 *It is not the case that both* $\Gamma_1; \Omega_1; (\Delta_1, x{:}C) \vdash M : A$ *and* $\Gamma_2; (\Omega_2, x{:}C); \Delta_2 \vdash M : A$.

Proof: By induction on the structure of $\mathcal{D}_1 :: \Gamma_1; \Omega_1; (\Delta_1, x{:}C) \vdash M : A$ and inversion on $\mathcal{D}_2 :: \Gamma_2; (\Omega_2, x{:}C); \Delta_2 \vdash M : A$.

Case:

$$\mathcal{D}_1 = \frac{}{\Gamma_1; \Omega_1; x{:}A \vdash x : A} Id^1$$

but there can be no proof of $x : A$ from $\Gamma_2; (\Omega_2, x{:}A); \Delta_2$.

No case for Id^u and Con.

Case:

$$\mathcal{D}_1 = \frac{\Gamma_1; \Omega_1; (\Delta_1, x{:}C, y{:}B) \vdash M : A}{\Gamma_1; \Omega_1; (\Delta_1, x{:}C) \vdash \lambda y^1 : B.\, M : B \overset{1}{\rightarrow} A} \overset{1}{\rightarrow} I$$

and

$$\mathcal{D}_2 = \cfrac{\Gamma_2; (\Omega_2, x{:}C); (\Delta_2, y{:}B) \vdash M : A}{\Gamma_2; (\Omega_2, x{:}C); \Delta_2 \vdash \lambda y^1 {:} B.\, M : B \overset{1}{\to} A} \overset{1}{\to} I$$

$\Gamma_1; \Omega_1; (\Delta_1, x{:}C, y{:}B) \vdash M : A$ By sub-derivation of $\mathcal{D}_1$
$\Gamma_2; (\Omega_2, x{:}C); (\Delta_2, y{:}B) \vdash M : A$ By inversion on $\mathcal{D}_2$
$\perp$ By IH

Case: $\mathcal{D}_1$ ends in $\overset{u}{\to} I, \overset{0}{\to} I$. The result follows analogously by IH.

Case: $\mathcal{D}_1$ ends in $\overset{u}{\to} E, \overset{0}{\to} E$. The result follows by IH on the leftmost sub-derivation.

Case: $\mathcal{D}_1$ ends in $\overset{1}{\to} E$: there are two sub-cases:

Subcase: $\Delta^1_M = \Psi^1_M, x{:}C$

$$\mathcal{D}_1 = \cfrac{(\Gamma_1, \Delta^1_N); \Omega_1; (\Psi^1_M, x{:}C) \vdash M : A \overset{1}{\to} B \qquad (\Gamma_1, \Psi^1_M, x{:}C); \Omega_1; \Delta^1_N \vdash N : A}{\Gamma_1; \Omega_1; (\Psi^1_M, x{:}C, \Delta^1_N) \vdash M\ N^1 : B} \overset{1}{\to} E$$

and

$$\mathcal{D}_2 = \cfrac{(\Gamma_2, \Delta^2_N); (\Omega_2, x{:}C); \Delta^2_M \vdash M : A \overset{1}{\to} B \qquad (\Gamma_2, \Delta^2_M); (\Omega_2, x{:}C); \Delta^2_N \vdash N : A}{\Gamma_2; (\Omega_2, x{:}C); (\Delta^2_M, \Delta^2_N) \vdash M\ N^1 : B} \overset{1}{\to} E$$

$(\Gamma_1, \Delta^1_N); \Omega_1; (\Psi^1_M, x{:}C) \vdash M : A \overset{1}{\to} B$ By sub-derivation of $\mathcal{D}_1$
$(\Gamma, \Delta^2_N); (\Omega_2, x{:}C); \Delta^2_M \vdash M : A \overset{1}{\to} B$ By inversion on $\mathcal{D}_2$
$\perp$ By IH

Subcase: $\Delta^1_N = \Psi^1_N, x{:}C$: Symmetrically.

□

Corollary 3.14 (Exclusivity)

1. *It is not the case that both* $\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A$ *and* $(\Gamma, \Omega, \Delta); x{:}C; \cdot \vdash M : A$.
2. *It is not the case that both* $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$ *and* $(\Gamma, \Omega, \Delta); \cdot; x{:}C \vdash M : A$.

Proof:

1. It is not the case that $\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A$ and $\Gamma; (\Omega, x{:}C); \Delta \vdash M : A$ By Lemma 3.13
   It is not the case that $\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A$ and $(\Gamma, \Delta); (\Omega, x{:}C); \cdot \vdash M : A$ By Loosening^1 Δ
   It is not the case that $\Gamma; \Omega; (\Delta, x{:}C) \vdash M : A$ and $(\Gamma, \Omega, \Delta); x{:}C; \cdot \vdash M : A$ By Loosening^0 Ω
2. Analogously.

□

We end this section by checking that our strict calculus is a conservative extension of the simply typed λ-calculus. We therefore define a forgetful functor from $\lambda^{\overset{l}{\to}}$ to $\lambda^{\to}$:

$$\begin{array}{rcl} \mid A \overset{k}{\to} B \mid & = & \mid A \mid \to \mid B \mid \\ \mid a \mid & = & a \\ \\ \mid x \mid & = & x \\ \mid c \mid & = & c \end{array}$$

$$
\begin{array}{rcl}
\mid \lambda x^k : A.\, M \mid & = & \lambda x : \mid A \mid .\ \mid M \mid \\
\mid M\ N^k \mid & = & \mid M \mid \mid N \mid \\
\\
\mid \cdot \mid & = & \cdot \\
\mid \Gamma, x{:}A \mid & = & \mid \Gamma \mid, x{:} \mid A \mid \\
\mid \Sigma, a{:}type \mid & = & \mid \Sigma \mid, a{:}type \\
\mid \Sigma, c{:}A \mid & = & \mid \Sigma \mid, c{:} \mid A \mid
\end{array}
$$

Theorem 3.15 (Conservativity)
If $\Gamma; \Omega; \Delta \vdash_{\lambda^{\overset{l}{\to}}} M : A$,*, then* $\mid \Gamma \mid, \mid \Omega \mid, \mid \Delta \mid \ \vdash_{\lambda^{\to}} \mid M \mid : \mid A \mid$.

Proof: By induction on the structure of the given derivation. □

3.2 The Canonical Form Theorem

In this section we establish the existence of canonical forms for $\lambda^{\overset{l}{\to}}$, i.e. β-normal η-long forms, which is crucial for our intended application. We prove this by Tait's method of *logical relations*; we essentially follow the account in [Pfe92], with a surprisingly little amount of generalization from simple to strict types; we do differ on the account of substitutions.

We start by presenting the inductive definition of *canonical* forms in $\lambda^{\overset{l}{\to}}$. It is realized by the two mutually recursive judgments depicted in Figure 3.5:

1. $\Gamma; \Omega; \Delta \vdash M \downarrow A$ $\quad$ M is atomic of type A.
2. $\Gamma; \Omega; \Delta \vdash M \Uparrow A$ $\quad$ M is canonical of type A.

Lemma 3.16 (Soundndess of Canonical Terms) *If* $\Gamma; \Omega; \Delta \vdash M \Updownarrow A$*, then* $\Gamma; \Omega; \Delta \vdash M : A$.

Proof: By induction on the structure of the derivation of $\Gamma; \Omega; \Delta \vdash M \Updownarrow A$. □

We then introduce *conversion* to canonical form in Figure 3.6. Note that conversion is not required to respect the occurrence constraints, provided that we start with a well-typed term:

1. $\Psi \vdash M \downarrow N : A$ $\quad$ M converts to atomic form N at type A.
2. $\Psi \vdash M \Uparrow N : A$ $\quad$ M converts to canonical form N at type A.

This utilizes weak head reduction, which includes local reduction (β) and partial congruence (ν):

$$
\frac{}{(\lambda x^k : A.\, M) N^k \overset{whr^k}{\longrightarrow} [N/x]M}\ \beta
\qquad
\frac{M \overset{whr^k}{\longrightarrow} Q}{(M\ N)^k \overset{whr^k}{\longrightarrow} (Q\ N)^k}\ \nu
$$

Theorem 3.17 (Conversion Yields Canonical Terms) *If* $(\Gamma, \Omega, \Delta) \vdash M \Updownarrow N : A$ *and* $\Gamma; \Omega; \Delta \vdash M : A$*, then* $\Gamma; \Omega; \Delta \vdash N \Updownarrow A$.

Proof: By induction on the structure of $\mathcal{D} :: (\Gamma, \Omega, \Delta) \vdash M \Updownarrow N : A$ and inversion on the typing derivation; we show some cases:

Case:

$$
\mathcal{D} = \frac{x{:}A \in \Psi}{\Psi \vdash x \downarrow x : A}\ tcIdvar
$$

$$\frac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c \downarrow A}\, cIdc$$

$$\frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x \downarrow A}\, cId^u \qquad \text{no rule for } cId^0 \qquad \frac{}{\Gamma; \Omega; x{:}A \vdash x \downarrow A}\, cId^1$$

$$\frac{\Gamma; \Omega; \Delta \vdash M \downarrow a}{\Gamma; \Omega; \Delta \vdash M \Uparrow a}\, cAt$$

$$\frac{(\Gamma, x{:}A); \Omega; \Delta \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^u{:}A.\, M) \Uparrow A \xrightarrow{u} B}\, c \xrightarrow{u} I$$

$$\frac{\Gamma; \Omega; (\Delta, x{:}A) \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^1{:}A.\, M) \Uparrow A \xrightarrow{1} B}\, c \xrightarrow{1} I$$

$$\frac{\Gamma; (\Omega, x{:}A); \Delta \vdash M \Uparrow B}{\Gamma; \Omega; \Delta \vdash (\lambda x^0{:}A.\, M) \Uparrow A \xrightarrow{0} B}\, c \xrightarrow{0} I$$

$$\frac{\Gamma; \Omega; \Delta \vdash M \downarrow A \xrightarrow{u} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N \Uparrow A}{\Gamma; \Omega; \Delta \vdash M\ N^u \downarrow B}\, c \xrightarrow{u} E$$

$$\frac{\Gamma; \Omega; \Delta \vdash M \downarrow A \xrightarrow{0} B \qquad (\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N \Uparrow A}{\Gamma; \Omega; \Delta \vdash M\ N^0 \downarrow B}\, c \xrightarrow{0} E$$

$$\frac{(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M \downarrow A \xrightarrow{1} B \qquad (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N \Uparrow A}{\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B}\, c \xrightarrow{1} E$$

Figure 3.5: Canonical forms

$$\frac{c{:}A \in \Sigma}{\Psi \vdash c \downarrow c : A}\, tcIdc \qquad \frac{x{:}A \in \Psi}{\Psi \vdash x \downarrow x : A}\, tcIdvar$$

$$\frac{M \xrightarrow{whr^k} M' \qquad \Psi \vdash M' \Uparrow M'' : a}{\Psi \vdash M \Uparrow M'' : a}\, tc \xrightarrow{whr^k} \qquad \frac{\Psi \vdash M \downarrow N : a}{\Psi \vdash M \Uparrow N : a}\, tc\mathrm{At}$$

$$\frac{\Psi \vdash M\ x^k \Uparrow N : B}{\Psi \vdash M \Uparrow (\lambda x^x{:}A.\, N) : A \xrightarrow{k} B}\, tc \xrightarrow{k} I \qquad \frac{\Psi \vdash M \downarrow P : A \xrightarrow{k} B \qquad \Psi \vdash N \Uparrow Q : A}{\Psi \vdash M\ N^k \downarrow P\ Q^k : B}\, tc \xrightarrow{k} E$$

Figure 3.6: Conversion to canonical form

Subcase:

$$\frac{}{\Gamma; \Omega; x{:}A \vdash x : A}\, Id^1$$

Then:

$$\frac{}{\Gamma; \Omega; x{:}A \vdash x \downarrow A}\, cId^1$$

Subcase:

$$\frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : A}\, Id^u$$

Then:

$$\frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x \downarrow A}\, cId^u$$

Case:

$$\mathcal{D} = \frac{M \stackrel{whr^k}{\longrightarrow} M' \qquad \Psi \vdash M' \Uparrow M'' : a}{\Psi \vdash M \Uparrow M'' : a}\, tc \stackrel{whr^k}{\longrightarrow}$$

$\Gamma; \Omega; \Delta \vdash M : a$ By hypothesis
$\Gamma; \Omega; \Delta \vdash M' : a$ By subject reduction (Theorem 3.10)
$\Gamma; \Omega; \Delta \vdash M'' \Uparrow a$ By IH

Case:

$$\mathcal{D} = \frac{\Psi \vdash M \downarrow P : A \stackrel{k}{\to} B \qquad \Psi \vdash N \Uparrow Q : A}{\Psi \vdash M\ N^k \downarrow P\ Q^k : B}\, tc \stackrel{k}{\to} E$$

Subcase:

$$\mathcal{D}' = \frac{\Gamma; \Omega; \Delta \vdash M : A \stackrel{u}{\to} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^u : B}\, E \stackrel{u}{\to} E$$

$\Gamma; \Omega; \Delta \vdash P \downarrow A \stackrel{u}{\to} B$ By IH
$(\Gamma, \Delta); \Omega; \cdot \vdash Q \Uparrow A$ By IH
$\Gamma; \Omega; \Delta \vdash P\ Q^u \downarrow B$ By rule

Subcase: $\mathcal{D}'$ is $E \stackrel{0}{\to} E, E \stackrel{1}{\to} E$: analogously.

□

Since we have to talk about open terms, we will need a notion of context extension:

$$\frac{}{\Psi \geq \Psi} \qquad \frac{\Psi' \geq \Psi}{\Psi', x{:}A \geq \Psi}$$

Lemma 3.18 (Weakening for Conversion to Canonical and Atomic Form)
If $\Psi \vdash M \Updownarrow N : A$ *and* $\Psi' \geq \Psi$, *then* $\Psi \vdash M \Updownarrow N : A$.

Proof: By induction on the structure of the given derivation(s). □

We can now introduce *logical relations*, in complete analogy with the usual definition for the simply-typed λ-calculus:

Definition 3.19 (Logical Relations)

1. $\Psi \vdash M \in [\![a]\!]$ *iff* $\Psi \vdash M \Uparrow N : A$, *for some* N.

2^k. $\Psi \vdash M \in [\![A \stackrel{k}{\to} B]\!]$ *iff for every* $\Psi' \geq \Psi$ *and every* N, *if* $\Psi' \vdash N \in [\![A]\!]$, *then* $\Psi' \vdash M\ N^k \in [\![B]\!]$.

Lemma 3.20

1. *If* $\Psi \vdash M \in [\![A]\!]$, *then* $\Psi \vdash M \Uparrow N : A$.
2. *If* $\Psi \vdash M \downarrow N : A$, *then* $\Psi \vdash M \in [\![A]\!]$.

Proof: By induction on A.

Case: $A = a$. Immediate from definition.

Case: $A = A \xrightarrow{k} B$.

1. $\Psi \vdash M \in [\![A \xrightarrow{k} B]\!]$ — By hypothesis
   $\Psi, x{:}A \geq \Psi$ — By rule
   $(\Psi, x{:}A) \vdash x \downarrow x : A$ — By rule
   $(\Psi, x{:}A) \vdash x \in [\![A]\!]$ — By IH 2
   $(\Psi, x{:}A) \vdash M\ x^k \in [\![B]\!]$ — By definition of $[\![\cdot]\!]$
   $(\Psi, x{:}A) \vdash M\ x^k \Uparrow N : B$ — By IH 1
   $\Psi \vdash M \Uparrow \lambda x^k{:}A.\, N : A \xrightarrow{k} B$ — By rule

2. $\Psi \vdash M \downarrow M_+ : A \xrightarrow{k} B$ — By hypothesis
   $\Psi' \geq \Psi$ and $\Psi' \vdash N \in [\![A]\!]$ — By assumption
   $\Psi' \vdash N \Uparrow N_+ : A$ — By IH 1
   $\Psi' \vdash M \downarrow M_+ : A \xrightarrow{k} B$ — By Lemma 3.18
   $\Psi' \vdash M\ N^k \downarrow M_+ N_+^k : B$ — By rule
   $\Psi' \vdash M\ N^k \in [\![B]\!]$ — By IH 2
   $\Psi \vdash M \in [\![A \xrightarrow{k} B]\!]$ — By definition of $[\![\cdot]\!]$

□

Lemma 3.21 (Closure under Head Expansion)
If $\Psi \vdash M' \in [\![A]\!]$ *and* $M \xrightarrow{whr^k} M'$, *then* $\Gamma; \Omega; \Delta \vdash M \in [\![A]\!]$.

Proof: By induction on A:

Case: $A = a$; immediate by definition and rule $tc \xrightarrow{whr^k}$.

Case: $A = A \xrightarrow{k} B$:

$\Psi \vdash M' \in [\![A \xrightarrow{k} B]\!]$ — By hypothesis
$\Psi' \vdash N \in [\![A]\!]$ for $\Psi' \geq \Psi$ — By assumption
$\Psi' \vdash M'\ N^k \in [\![A]\!]$ — By definition
$(M\ N)^k \xrightarrow{whr^k} (M'\ N)^k$ — By rule ν
$\Psi' \vdash M\ N^k \in [\![A]\!]$ — By IH
$\Psi \vdash M \in [\![A \xrightarrow{k} B]\!]$ — By definition

□

Due to the need to β-reduce during conversion to canonical form, we need to introduce *substitutions*. Differently from [Pfe92] and [Cer], we will not require substitutions to be well-typed.

$$\textit{Substitutions} \quad \theta \quad ::= \epsilon \mid \theta, M/x$$

For $\theta = \theta', M/x$, we say that x is *defined* in θ and we write $\theta(x) = M$. We require all variables defined in a substitution to be distinct: we use $\mathrm{dom}(\theta)$ for the set of variables defined in θ and $\mathrm{cod}(\theta)$ for the variables occurring in the substituting terms. We assume them to be disjoint.

Next, we define the *application* of a substitution to a term M, denoted $[\theta]M$. We limit application of substitution to objects whose free variables are in the domain of θ:

$$\begin{array}{rcl}
[\theta]c & = & c \\
[\theta]x & = & \theta(x) \\
[\theta](M\ N^k) & = & ([\theta]M)\ ([\theta]N)^k \\
[\theta](\lambda x^k{:}A.\, M) & = & \lambda x^k{:}A.\, [\theta, x/x]M
\end{array}$$

Note that in the lambda case we can assume with no loss of generality that x does not occur in $\mathrm{dom}(\theta) \cup \mathrm{cod}(\theta)$.

We will also need to mediate between single-step substitutions stemming from β-reduction and simultaneous substitutions. We define how to *compose* single step bindings from a β reduction with simultaneous substitutions:

$$\begin{array}{rcl}
[N/x]\epsilon & = & \epsilon \\
[N/x]\theta & = & [N/x]\theta, ([N/x]M)/y
\end{array}$$

Lemma 3.22 $[[N/x]\theta]M = [N/x]([\theta]M)$.

Proof: By induction on the structure of M. □

Corollary 3.23 *Assume x to be fresh in θ: $[N/x]([\theta, x/x]M) = [\theta, N/x]M$.*

Proof:

$[N/x]([\theta, x/x])M =$	By hypothesis
$[[N/x](\theta, x/x)])M =$	By Lemma 3.22
$[[N/x]\theta, ([N/x]x)/x)])M =$	By composition of substitution
$[\theta, N/x]M$	By application of substitution and x fresh

□

For a context $\Psi = x_1{:}A_1, , \ldots, x_n{:}A_n$, we introduce the *identity* substitution on Ψ as $id_\Psi = x_1/x_1, \ldots, x_n/x_n$.

Lemma 3.24 (Identity Substitution) *If $\Gamma; \Omega; \Delta \vdash M : A$, then $[id_{(\Gamma,\Omega,\Delta)}]M = M$.*

Proof: By induction on M. □

We extend the notion of logical relations to contexts, exactly as in the simply typed case: a substitution θ is in the relation $[\![\Psi]\!]$ if for every binding M/x such that $x{:}A$ is in Ψ, then M is in $[\![A]\!]$.

Definition 3.25

1. $\Phi \vdash \theta \in [\![\cdot]\!]$ *iff* $\theta = \epsilon$.
2. $\Phi \vdash \theta \in [\![(\Psi, x{:}A)]\!]$ *iff* $\theta = \theta', M/x$, $\Phi \vdash M \in [\![A]\!]$ *and* $\Phi \vdash \theta' \in [\![\Psi]\!]$.

We remark that contexts are *not* ordered, hence, for $\Psi = (\Gamma, \Omega, \Delta)$ we will identify, for example, $[\![\Psi, x{:}A]\!]$ with $[\![(\Gamma, x{:}A, \Omega, \Delta)]\!]$.

Lemma 3.26 (Weakening for Logical Relations) *If $\Psi \vdash \theta \in [\![A]\!]$, then $(\Psi, x{:}A) \vdash \theta \in [\![A]\!]$.*

Proof: By induction on the structure of the given derivation. □

Lemma 3.27 (Well-typed Terms are in the Logical Relation) *If $\Gamma; \Omega; \Delta \vdash M : A$, then for every Φ such that $\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$, $\Phi \vdash [\theta]M \in [\![A]\!]$.*

Proof: By induction on the typing derivation $\mathcal{D} :: \Gamma; \Omega; \Delta \vdash M : A$:

Case:

$$\mathcal{D} = \frac{}{(\Gamma, x{:}A); \Omega; \cdot \vdash x : A} Id^u$$

$\Phi \vdash \theta \in [\![(\Gamma, x{:}A, \Omega)]\!]$	By assumption
$\Phi \vdash \theta(x) \in [\![A]\!]$	By definition of $[\![\cdot]\!]$
$\Phi \vdash [\theta]x \in [\![A]\!]$	By definition of substitution

Case:

$$\mathcal{D} = \frac{}{\Gamma; \Omega; x{:}A \vdash x : A} Id^1$$

$\Phi \vdash \theta \in [\![(\Gamma, \Omega, x{:}A)]\!]$	By assumption
$\Phi \vdash \theta(x) \in [\![A]\!]$	By definition of $[\![\cdot]\!]$
$\Phi \vdash [\theta]x \in [\![A]\!]$	By definition of substitution.

Case:

$$\mathcal{D} = \frac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c : A} Con$$

Immediate by Lemma 3.20 and definition of substitution.

Case:

$$\mathcal{D} = \frac{(\Gamma, x{:}A); \Omega; \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^u{:}A.\ M : A \xrightarrow{u} B} \xrightarrow{u} I$$

$(\Gamma, x{:}A); \Omega; \Delta \vdash M : B$	By sub-derivation
$\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	By hypothesis
$\Phi' \geq \Phi$ and $\Phi' \vdash N \in [\![A]\!]$	By assumption
$\Phi' \vdash (\theta, N/x) \in [\![(\Gamma, x{:}A, \Omega, \Delta)]\!]$	By definition of $[\![\cdot]\!]$ and Weakening (Lemma 3.26)
$\Phi' \vdash [\theta, N/x]M \in [\![B]\!]$	By IH
$\Phi' \vdash [N/x]([\theta, x/x]M) \in [\![B]\!]$	By Corollary 3.23
$\Phi' \vdash (\lambda x^u{:}A.\, [\theta, x/x]M) N^u \in [\![B]\!]$	By Lemma 3.21
$\Phi' \vdash ([\theta](\lambda x^u{:}A.\, M)) N^u \in [\![B]\!]$	By definition of substitution
$\Phi \vdash [\theta](\lambda x^u{:}A.\, M) \in [\![A \xrightarrow{u} B]\!]$	By definition of $[\![A \xrightarrow{u} B]\!]$

Case:

$$\mathcal{D} = \frac{\Gamma; (\Omega, x{:}A); \Delta \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^0{:}A.\ M : A \xrightarrow{0} B} \xrightarrow{0} I$$

$\Gamma; (\Omega, x{:}A); \Delta \vdash M : B$	By sub-derivation
$\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	By hypothesis
$\Phi' \geq \Phi$ and $\Phi' \vdash N \in [\![A]\!]$	By assumption
$\Phi' \vdash (\theta, N/x) \in [\![(\Gamma, \Omega, x{:}A, \Delta)]\!]$	By definition of $[\![\cdot]\!]$ and Weakening (Lemma 3.26)
$\Phi' \vdash [\theta, N/x]M \in [\![B]\!]$	By IH
$\Phi' \vdash [N/x]([\theta, x/x]M) \in [\![B]\!]$	By Corollary 3.23
$\Phi' \vdash (\lambda x^0{:}A.\, [\theta, x/x]M) N^0 \in [\![B]\!]$	By Lemma 3.21
$\Phi' \vdash ([\theta](\lambda x^0{:}A.\, M)) N^0 \in [\![B]\!]$	By definition of substitution
$\Phi \vdash [\theta](\lambda x^0{:}A.\, M) \in [\![A \xrightarrow{0} B]\!]$	By definition of $[\![A \xrightarrow{0} B]\!]$

Case:

$$\mathcal{D} = \frac{\Gamma; \Omega; (\Delta, x{:}A) \vdash M : B}{\Gamma; \Omega; \Delta \vdash \lambda x^1{:}A.\ M : A \xrightarrow{1} B} \xrightarrow{1} I$$

$\Gamma; \Omega; (\Delta, x{:}A) \vdash M : B$	By sub-derivation
$\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	By hypothesis
$\Phi' \geq \Phi$ and $\Phi' \vdash N \in [\![A]\!]$	By assumption
$\Phi' \vdash (\theta, N/x) \in [\![(\Gamma, \Omega, \Delta, x{:}A)]\!]$	By definition of $[\![\cdot]\!]$ and Weakening (Lemma 3.26)
$\Phi' \vdash [\theta, N/x]M \in [\![B]\!]$	By IH
$\Phi' \vdash [N/x]([\theta, x/x]M) \in [\![B]\!]$	By Corollary 3.23
$\Phi' \vdash (\lambda x^1 : A. [\theta, x/x]M) N^1 \in [\![B]\!]$	By Lemma 3.21
$\Phi' \vdash ([\theta](\lambda x^1 : A. M)) N^1 \in [\![B]\!]$	By definition of substitution
$\Phi' \vdash [\theta](\lambda x^1 : A. M) \in [\![A \xrightarrow{1} B]\!]$	By definition of $[\![A \xrightarrow{1} B]\!]$

Case:

$$\mathcal{D} = \frac{\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{u} B \qquad (\Gamma, \Delta); \Omega; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^u : B} \xrightarrow{u} E$$

$\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	By hypothesis
$\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{u} B$	By sub-derivation
$\Phi \vdash [\theta]M \in [\![A \xrightarrow{u} B]\!]$	By IH
$(\Gamma, \Delta); \Omega; \cdot \vdash N : A$	By sub-derivation
$\Phi \vdash [\theta]N \in [\![A]\!]$	By IH
$\Phi \geq \Phi$	By rule
$\Phi \vdash ([\theta]M)([\theta]N)^u \in [\![B]\!]$	By definition of $[\![\cdot]\!]$
$\Phi \vdash [\theta](M\ N)^u \in [\![B]\!]$	By definition of substitution

Case:

$$\mathcal{D} = \frac{\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{0} B \qquad (\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N : A}{\Gamma; \Omega; \Delta \vdash M\ N^0 : B} \xrightarrow{0} E$$

$\Phi \vdash \theta \in [\![(\Gamma, \Omega, \Delta)]\!]$	By hypothesis
$\Gamma; \Omega; \Delta \vdash M : A \xrightarrow{0} B$	By sub-derivation
$\Phi \vdash [\theta]M \in [\![A \xrightarrow{0} B]\!]$	By IH
$(\Gamma, \Omega, \Delta); \cdot; \cdot \vdash N : A$	By sub-derivation
$\Phi \vdash [\theta]N \in [\![A]\!]$	By IH
$\Phi \geq \Phi$	By rule
$\Phi \vdash ([\theta]M)([\theta]N)^0 \in [\![B]\!]$	By definition of $[\![\cdot]\!]$
$\Phi \vdash [\theta](M\ N)^0 \in [\![B]\!]$	By definition of substitution

Case:

$$\mathcal{D} = \frac{(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M : A \xrightarrow{1} B \qquad (\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A}{\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 : B} \xrightarrow{1} E$$

$\Phi \vdash \theta \in [\![(\Gamma, \Omega, (\Delta_M, \Delta_N))]\!]$	By hypothesis
$(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M : A \xrightarrow{1} B$	By sub-derivation
$\Phi \vdash [\theta]M \in [\![A \xrightarrow{1} B]\!]$	By IH
$(\Gamma, \Delta_M); \Omega; \Delta_N \vdash N : A$	By sub-derivation
$\Phi \vdash N \in [\![A]\!]$	By IH
$\Phi \geq \Phi$	By rule
$\Phi \vdash ([\theta]M)([\theta]N)^1 \in [\![B]\!]$	By definition of $[\![\cdot]\!]$
$\Phi \vdash [\theta](M\ N)^1 \in [\![B]\!]$	By definition of substitution

□

Lemma 3.28 $\Psi \vdash id_\Psi \in [\![\Psi]\!]$.

Proof: By a straightforward induction on Ψ using Lemma 3.20(2). $\square$

Theorem 3.29 (Canonical Form Theorem) *If* $\Gamma;\Omega;\Delta \vdash M : A$, *then there is* N *such that* $(\Gamma,\Omega,\Delta) \vdash M \Uparrow N : A$ *and* $\Gamma;\Omega;\Delta \vdash N \Uparrow A$.

Proof: Assume $\Gamma;\Omega;\Delta \vdash M : A$: By Lemma 3.28 $(\Gamma,\Omega,\Delta) \vdash id_{(\Gamma,\Omega,\Delta)} \in [\![(\Gamma,\Omega,\Delta)]\!]$, hence, by Lemma 3.27 $(\Gamma,\Omega,\Delta) \vdash [id_{(\Gamma,\Omega,\Delta)}]M \in [\![A]\!]$ and thus by Lemma 3.24 $(\Gamma,\Omega,\Delta) \vdash M \in [\![A]\!]$; by Lemma 3.20(1), $(\Gamma,\Omega,\Delta) \vdash M \Uparrow N : A$ for some N. Additionally, by Theorem 3.17(2) $\Gamma;\Omega;\Delta \vdash N \Uparrow\!\!\Downarrow A$. $\square$

We shall abbreviate the statement of the canonical form Theorem as $\Gamma;\Omega;\Delta \vdash M \Uparrow\!\!\Downarrow A$.

We will also need the typing and the canonical form rule for existential variables. We use Φ for arrays of (distinct) labeled bound variables; if $x^k \in \Phi$, we set $\Phi(x) = k$. We say that $\Gamma;\Omega;\Delta \vdash \Phi$ *ok* if the following holds:

$$\begin{array}{lcl}
\Phi(x) = u & \leftrightarrow & x \in \mathrm{dom}(\Gamma) \\
\Phi(x) = 0 & \leftrightarrow & x \in \mathrm{dom}(\Omega) \\
\Phi(x) = 1 & \leftrightarrow & x \in \mathrm{dom}(\Delta)
\end{array}$$

Moreover, if $\Gamma;\Omega;\Delta \vdash M : A$ and $\Gamma;\Omega;\Delta \vdash \Phi$ *ok* we may write $\ulcorner\Phi\urcorner \vdash M : A$. We assume that the type A in E_A is well-behaved w.r.t. Φ.

$$\frac{\Gamma;\Omega;\Delta \vdash \Phi\ ok}{\Gamma;\Omega;\Delta \vdash E_A\ \Phi : a}\,Pat \qquad \frac{\Gamma;\Omega;\Delta \vdash E_A\ \Phi : a}{\Gamma;\Omega;\Delta \vdash E_A\ \Phi \downarrow a}\,cPat$$

Remark 3.30 *Exclusivity (Lemma 3.13) holds for open patterns as well.*

Proof: Assume that both $\Gamma;\Omega;(\Delta,x{:}C) \vdash E\ \Phi : A$ and $\Gamma';(\Omega',x{:}C);\Delta' \vdash E\ \Phi : A$. Then $\Gamma;\Omega;(\Delta,x{:}C) \vdash \Phi$ *ok* and $\Gamma';(\Omega',x{:}C);\Delta' \vdash \Phi$ *ok* iff $\Phi(x) = 1$ and $\Phi(x) = 0$, impossible. $\square$

3.3 Related Work on Strictness

Church original definition of the set Λ_I of (untyped) λ-terms [Chu41] has this clause for abstraction:

$$\text{If } M \in \Lambda_I \text{ and } x \in FV(M), \text{ then } \lambda x\,.\,M \in \Lambda_I.$$

i.e. in this language there cannot be any vacuous abstractions. It can be shown that the only difference between Λ_I and Λ – the usual definition of λ-terms – is the lack of the combinator **K**. Indeed, it can be shown that every term in Λ can be defined from Λ_I and **K**. The λI-calculus is the theory of conversion restricted to Λ_I terms. This fragment was favored by Church over the nowadays usual calculus, because, among other issues, it is strong enough to represent every partial recursive function, albeit not in the most efficient way: see [Bar80] Chapter 2.2.2 – 2.2.5 and more extensively in Chapter 9. See [GdQ92] for an historical account.

This would correspond in a simply typed setting to allowing *only* strict types: more formally if we denote with $\Lambda^{\stackrel{k}{\to}}$ the terms typable in a Curry system based on the $\stackrel{k}{\to}$ function space, then $\Lambda^{\stackrel{1}{\to}} = \Lambda^{\stackrel{u}{\to}} \cap \Lambda_I$, as noted for example in [BF93].

The combinatory counterpart of this calculus obviously excludes **K** and consists of $\mathbf{I}, \mathbf{W}, \mathbf{B}, \mathbf{C}$, see [CF58] and [Bar80], Appendix B for an alternative basis. Those are the axioms of what Church called *weak implicational logic* [Chu51], i.e. identity, contraction, prefixing and permutation. This establishes the link with an enterprise born from a very different origin.

The relevance logic project emerged in fact in the early sixties out of Anderson and Belnap's dissatisfaction with the so-called 'paradoxes of implication', let it be material, intuitionistic or strict (in the modal sense of

Lewis and Langford); it was built on the work of Moh, Church, Parry in the fifties[1] and climaxed with the publication of the first volume of *Entailment* [AB75] (the second one was published only in 1992 [AAB92]). Following Girard's and Belnap's [Bel93] suggestion, we will *not* refer to our calculus as *relevant*, but as *strict* logic, as the former may also satisfy other principles such as distributivity of arrow over conjunction.

On an unrelated front, starting with Mycroft's seminal paper [Myc80], compile-time analysis of functional programs concentrated on strictness analysis in order to get the best out of call-by-value and call-by-need evaluation; first in terms of abstract interpretation, later by using non-standard types to represent these 'intensional' information about functions (see [Jen91] for a comparison of these two techniques). However, earlier work as [TMM89] used non-standard primitive type to distinguish strict or non-strict terms, closed only under intuitionistic implication. Not forgetting Wadler's early paper [Wad90] on using linear logic for sharing analysis, Wright [Wri91, Wri92] seems the first one to have extended the Curry-Howard isomorphism to (the implicational fragment of) relevance logic and explicitly connected the two areas, although both [Bel74] and [Hel77] had previously recognized the link between strictness and relevance[2].

In [BF93] the author summarizes the above-mentioned idea of expressing via types the reduction behavior of terms. He characterizes in an operational sense the class of terms which need their argument, the idea being that not only each terms need to be strict, but so does the result of each application. M is not strict if for all N no descendant of N is in the normal form of $M\ N$. This class is then shown to be equivalent to the Curry-typable fragment of Λ_I.

We now discuss the Curry typing system proposed there, which makes available strict, invariant and intuitionistic types: yet, it is biased towards inferring strictness information, which ultimately lead to a difference of expressive power from our calculus. Some rules are presented in Figure 3.7 – we omit the introduction rules which are as expected – transliterated in our notation. There is only one context, where variables carry their occurrence status as a label. Being a term assigment system, there are no different abstractors or applicators, but different rules. Note that there is only one identity rule, the strict one, so that e.g. $\lambda x\,.\,x : A \overset{u}{\to} A$ is not derivable, as it can be given the more stringent type $A \overset{1}{\to} A$. Let us concentrate on the elimination rules: the side condition enforces the information ordering, so that for example $A' \overset{0}{\to} B \leq A \overset{u}{\to} B'$, provided that $A \leq A', B \leq B'$. This allow to infer by strict application $\Gamma, \Gamma' \vdash M\ N : C$ from $\Gamma \vdash M : (A \overset{u}{\to} B) \overset{1}{\to} C$ and $\Gamma' \vdash A \overset{0}{\to} B$. The latter is instead forbidden in our system by the labeled reduction rules. The rationale on the substitution operation on context is that in *app* $\overset{0}{\to}$ A is not relevant to B, so all hypothesis should be deleted. Instead, in order to preserve every variable declaration, their strict label is changed into irrelevant. This would amount to moving the strict variables in the irrelevant context in our system. Note the difference with our rule, where the latter variables are moved in the *undetermined* context. Similarly in *app* $\overset{u}{\to}$ strict labels turn into undetermined. Moreover, having only one context, the author needs a strategy to deal with same binding with different annotations; the solution is that while propagating premises top-down a binding $x^1{:}A$ supersedes $x^u{:}A$ which in turn supersedes $x^0{:}A$.

The author goes on (see also [WBF93]) detailing a system which refines the strict calculus by allowing to *count* usage, motivated by *sharing* analysis; thus $A \overset{i}{\to} B$ denotes a term where A is used i times to infer B. Undetermined usage is then added via dummy variables. This unfortunately leads to an undecidable type checking problem.

In [Wri96] Wright introduces an *Annotation Logic* as a general framework for resource-conscious logics. The annotation logic has formulae/types of the form

$$A ::= X^k \mid A \overset{k}{\to} B$$

for any annotation k and has structural and connective rules as well as annotation ones:

$$\frac{\Gamma \vdash A^i}{\Gamma^k \vdash A^{ik}}\,{*k} \qquad \frac{\Gamma, A^i \vdash B}{\Gamma, A^{i+k} \vdash B}\,{+k}$$

The latter implement rules such as promotion/dereliction. By instantiation with different algebras of annotation, we get systems as linear, strict logic as well as various other usage logics. An abstract normalization

[1]Some early work in the twenties in the Soviet Union was, at the time, not accessible.

[2]Note that we have became aware of this literature only after having fully developed our calculus which was modeled after the two-zoned linear logic calculus.

$$\frac{}{\Gamma[1 := 0], x^1{:}A \vdash x : A}\ var$$

$$\frac{\Gamma \vdash M : A \xrightarrow{1} B \qquad \Gamma' \vdash N : A'}{\Gamma, \Gamma' \vdash M\ N : B}\ app \xrightarrow{1}$$

$$\frac{\Gamma \vdash M : A \xrightarrow{0} B \qquad \Gamma' \vdash N : A'}{\Gamma, \Gamma'[1 := 0] \vdash M\ N : B}\ app \xrightarrow{0}$$

$$\frac{\Gamma \vdash M : A \xrightarrow{u} B \qquad \Gamma' \vdash N : A'}{\Gamma, \Gamma'[1 := u] \vdash M\ N : B}\ app \xrightarrow{u}$$

All elimination rules have the condition $A' \leq A$

Figure 3.7: The system in [BF93]

procedure is sketched, which however needs commutative conversions (e.g. the case contraction/arrow elimination) already in the purely implicational fragment. Another problem is that properties as loosening should in our opinion be admissible rather than primitive rules.

Bunched Implication

In a series of papers Pym et at. (see [Pym99] and references therein) introduce a first-order and its correspondent dependent type calculus which aim to couple multiplicative and addititive implication. The two are distinct by allowing two different constructors for contexts which are called “bunches”. This is different from a zoned calculus as bunches can be nested. As expected, contraction and weakening are allowed for additive assumptions but not for multiplicative ones. The resulting logic can be seen as variant of relevant logic, as there is no requirement that an argument to a multiplicative function must be used only once, but only that it should not share with other variables in the (proof) term. Thus, in out terminology $A \xrightarrow{1} ((A \xrightarrow{u} A \xrightarrow{u} B) \xrightarrow{u} B)$ is derivable, but not $A \xrightarrow{1} ((A \xrightarrow{1} A \xrightarrow{1} B) \xrightarrow{1} B)$. Moreover, the logic allows additive conjunction to distribute on additive disjunction, which is not allowed in multiplicative additive intuitionistic linear logic. Its naturality should follow from its categorical semantics. Its correspondent dependently-typed calculus, dubbed *RLF* is proposed as a resource conscious conservative extension of LF.

Relevant Logic Programming

In [Bol90] the author presents his approach to *relevant*, i.e. in our terminology strict logic programming as part of his dissertation on ‘Conditional Logic Programming’ [Bol88]. He makes a (weak) case for its utility in applications such as planning and diagnosis, whose hypothetical queries should indeed use their premises. The system boils down to a strict version of N-Prolog. Unfortunately the author was only partially aware of Girard’s work on linear logic, and entirely not aware of the notion of uniform proof [MNPS91], although he gives a brave attempt to a mainly proof-theoretic approach: thus, as Gabbay and McCarthy before, he embarks on an awfully complicated and low-level description of an interpreter which enforces the usage requirement, for, I think, the following fragment; note that there are two conjunctions: & is additive and $\wedge$ multiplicative.

$$\begin{array}{rrcl} \textit{Assertions} & A & ::= & P \mid Q \xrightarrow{1} P \mid A_1 \& A_2 \mid \forall x\,.\,A \\ \textit{Queries} & Q & ::= & P \mid Q_1 \wedge Q_2 \mid Q_1 \vee Q_2 \mid A \xrightarrow{1} Q \mid \exists x\,.\,Q \end{array}$$

If we were formulating a strict logical framework in the sense of [Cer96], the former system would therefore be a strict (no pun intended) subset of the latter.

Chapter 4

The Relative Complement Problem for Higher-Order Patterns

We introduce in the next Section 4.1 a restriction of the language for which complementation is possible (Section 4.2). Moreover, in Section 4.3 we will give an unification algorithm for this fragment. Section 4.4 shows how the former operations induce a boolean algebra over finite sets of terms.

4.1 Towards Term Complementation

Now that we have developed a calculus which is potentially strong enough to represent the complement of linear patterns, we need to answer two questions: how do we embed the original λ-calculus, and is the calculus now closed under complement?

We reiterate that we require that our complement operator ought to satisfy the usual boolean rules for negation:

1. (Disjointness) It is not the case that some M is both a ground instance of N and of $\mathrm{Not}(N)$.

2. (Exhaustivity) Every M is a ground instance of N or of $\mathrm{Not}(N)$.

Unfortunately, while the first property follows quite easily from Corollary 3.14, it turns out that exhaustivity does not hold in general in the presence of intuitionistic application. In fact, consider the application $y\ x^u$; while it is clear that $\Gamma \vdash y\ x^u \in \|E\ x^u\ y^u\|$, it is *not* the case that $\Gamma \vdash y\ x^u \in \|E\ x^1\ y^u\|$ or $\Gamma \vdash y\ x^u \in \|E\ x^0\ y^u\|$.

However, the main result of this Chapter is that the complement algorithm presented in Definition 4.13 is sound and complete for the fragment which results from the natural embedding of the original simply-typed λ-calculus; this is sufficient for our intended application. We will proceed in two separate phases:

- Restrict to a class of terms (that we call *simple*) for which the crucial property of *tightening* (Lemma 4.5) can be established, yielding exhaustivity as a corollary.

- Bring simple terms to 'full application'.

4.1.1 Simple Terms

Recall that we have introduced strictness to capture occurrence conditions on variables in canonical forms. This means that first-order constants (and by extension bound variables) should be considered *strict* functions of their argument, since these arguments will indeed occur in the canonical form. On the other hand, if we have a second order constant, we cannot restrict the argument function to be either strict or vacuous, since this would render our representations inadequate.

Example 4.1 *Continuing Example 2.6, consider the representation of the* $\boldsymbol{K}$ *combinator:*

$$\ulcorner \lambda x.\, \lambda y.\, x \urcorner \quad = \quad lam\ (\lambda x\!:\!exp.\, lam\ (\lambda y\!:\!exp.\, x))$$

Notice that the argument to the first occurrence of ' lam *' is a strict function, while the argument to the second occurrence is an invariant function. If we can give only one type to '* lam *' it must therefore be* $(exp \overset{u}{\to} exp) \overset{1}{\to} exp$*.*

Generalizing this observation means that positive occurrence of function types are translated to strict functions, while the negative ones to undetermined functions. We can formalize this as an embedding of the simply-typed λ-calculus into a fragment of the strict calculus via two (overloaded) mutually recursive functions $()^-$ and $()^+$. First, the definition on types:

$$\begin{array}{rcl}
(A \to B)^+ & = & A^- \overset{1}{\to} B^+ \\
(A \to B)^- & = & A^+ \overset{u}{\to} B^- \\
a^- = a^+ & = & a
\end{array}$$

We extend it to canonical terms (including existential variables), signatures, and contexts; we thusly need the usual inductive definition of canonical terms in the simply-typed λ-calculus, which can be obtained by dropping labels (Theorem 3.15) from the definition of canonical form in Figure 3.5. Note that embedding only canonical forms rules out the case of '+-ing' a lambda expression, as well as '−-ing abstractions and non-atomic $h \in \mathrm{dom}(\Gamma \cup \Sigma$.

$$\begin{array}{rcll}
(\lambda x\!:\!A.\, M)^- & = & \lambda x^u\!:\!A^+.\, M^- & \\
M^- & = & M^+ & \text{for } M \text{ of base type} \\
x^+ & = & x & \\
c^+ & = & c & \\
(E_A\ x_1 \ldots x_n)^+ & = & F_{A^-}\ x_1^u \ldots x_n^u & \\
(M\ N)^+ & = & M^+\ (N^-)^1 & \\
 & & & \\
(\cdot)^+ & = & \cdot & \\
(\Gamma, x{:}A)^+ & = & \Gamma^+, x{:}A^+ & \\
(\Sigma, a{:}type)^+ & = & \Sigma^+, a{:}type & \\
(\Sigma, c{:}A)^+ & = & \Sigma^+, c{:}A^+ &
\end{array}$$

Example 4.2 *Coming back to Example 4.1:*

$$(lam\ (\lambda x\!:\!exp.\, lam\ (\lambda y\!:\!exp.\, x)))^+ \quad = \quad lam\ (\lambda x^u\!:\!exp.\, lam\ (\lambda y^u\!:\!exp.\, x)^1)^1$$

The image of the embedding of the canonical forms of the simply-typed λ-calculus gives rise to the following fragment:

$$\textit{Simple Terms} \quad M \quad ::= \quad \lambda x^u{:}A^+.\, M \mid (\ldots(h\ M_1)^1 \ldots M_n)^1 \mid (\ldots(E_{A^-}\ x_1)^u \ldots x_n)^u$$

We often abbreviate $(\ldots(h\ M_1)^1 \ldots M_n)^1$ as $h\ \overline{M_n^1}$; similarly we shall use $F_{A^-}\ \overline{x_n^u}$. Note that, by the use of η-long β-normal forms, such terms, as well as pattern variables, must be of base type.

To prove the correctness of the embedding (Theorem 4.4), we will need the following:

Lemma 4.3 *If* $\Gamma \vdash E_A\ \overline{x_n} : a$*, then* $\Gamma^+; \cdot; \cdot \vdash F_{A^-}\ \overline{x_n^u} : a$*.*

Proof: A straightforward induction on A. □

Theorem 4.4 (Correctness of $()^{\pm}$**)**

1. *If* $\Gamma \vdash M \Uparrow A$*, then* $\Gamma^+; \cdot; \cdot \vdash M^- \Uparrow A^-$.
2. *If* $\Gamma \vdash M \downarrow A$*, then* $\Gamma^+; \cdot; \cdot \vdash M^+ \downarrow A^+$.

Proof: By mutual induction on the proofs of $\mathcal{D}_1 :: \Gamma \vdash M \Uparrow A$ and $\mathcal{D}_2 :: \Gamma \vdash M \downarrow A$.

Case:

$$\mathcal{D}_2 = \frac{c{:}A \in \Sigma}{\Gamma \vdash c \downarrow A}\,\texttt{atmCnst}$$

Since $c^+ = c$ and $\Sigma^+(c) = A^+$ we conclude

$$\mathcal{E} - \frac{}{\Gamma^+; \cdot; \cdot \vdash c^+ \downarrow A^+}\,cIdc$$

Case:

$$\mathcal{D}_2 = \frac{}{(\Gamma, x{:}A) \vdash x \downarrow A}\,\texttt{atmVar}$$

Since $x^+ = x$ and $\Gamma^+(x) = A^+$ we conclude

$$\mathcal{E} = \frac{}{(\Gamma, x{:}A)^+ \vdash x^+ \downarrow A^+}\,cId^u$$

Case:

$$\mathcal{D}_2 = \frac{\Gamma \vdash M \downarrow B \to A \qquad \Gamma \vdash N \Uparrow B}{\Gamma \vdash M\ \ N \downarrow A}\,\texttt{atmApp}$$

$\Gamma \vdash M \downarrow B \to A$	By sub-derivation
$\Gamma^+; \cdot; \cdot \vdash M^+ \downarrow (B \to A)^+$	By IH 2
$\Gamma^+; \cdot; \cdot \vdash M^+ \downarrow B^- \xrightarrow{1} A^+$	By the embedding
$\Gamma \vdash N \Uparrow B$	By sub-derivation
$\Gamma^+; \cdot; \cdot \vdash N^- \Uparrow B^-$	By IH 1
$\Gamma^+; \cdot; \cdot \vdash M^+\ (N^-)^1 \downarrow A^+$	By rule $c \xrightarrow{1} E$
$\Gamma^+; \cdot; \cdot \vdash (M\ \ N)^+ \downarrow A^+$	By the embedding

Case:

$$\mathcal{D}_2 = \frac{\Gamma \vdash E_A\ \overline{x_n} : a}{\Gamma \vdash E_A\ \overline{x_n} \downarrow a}\,\texttt{canPat}$$

$\Gamma \vdash E_A\ \overline{x_n} : a$	By sub-derivation
$\Gamma^+; \cdot; \cdot \vdash F_{A^-}\ \overline{x_n^u} : a$	By Lemma 4.3
$\Gamma^+; \cdot; \cdot \vdash F_{A^-}\ \overline{x_n^u} \downarrow a$	By rule $cPat$
$\Gamma^+; \cdot; \cdot \vdash (E_A\ \overline{x_n})^+ : a^+$	By the embedding

Case:

$$\mathcal{D}_1 = \frac{\Gamma \vdash M \downarrow a}{\Gamma \vdash M \Uparrow a}\,\texttt{canAtm}$$

$\Gamma \vdash M \downarrow a$	By sub-derivation
$\Gamma^+; \cdot; \cdot \vdash M^+ \downarrow a^+$	By IH 2
$\Gamma^+; \cdot; \cdot \vdash M^+ \Uparrow a^+$	By rule cAt
$\Gamma^+; \cdot; \cdot \vdash M^- \Uparrow a^-$	By the embedding

Case:

$$\mathcal{D}_1 = \frac{\Gamma, x{:}A \vdash M \Uparrow B}{\Gamma \vdash \lambda x\,{:}\,A.\, M \Uparrow A \to B}\,\texttt{canLam}$$

$\Gamma, x{:}A \vdash M \Uparrow B$	By sub-derivation
$(\Gamma, x{:}A)^+ \vdash M^- \Uparrow B^-$	By IH 1
$(\Gamma^+, x{:}A^+) \vdash M^- \Uparrow B^-$	By the embedding
$\Gamma^+; \cdot; \cdot \vdash \lambda x^u\,{:}\,A^+.\, M \Uparrow A^+ \xrightarrow{u} B^-$	By rule $c \xrightarrow{u} I$
$\Gamma^+; \cdot; \cdot \vdash (\lambda x\,{:}\,A.\, M)^- \Uparrow (A \to B)^-$	By the embedding

□

From now on we may hide the $()^1$ decoration from strict application of constants in examples. Moreover, for every judgment $\mathcal{J}$ on simple terms, we will shorten $\Gamma; \cdot; \cdot \vdash \mathcal{J}$ into $\Gamma \vdash \mathcal{J}$.

We can now prove the crucial tightening lemma. It expresses the property that every *closed simple term* is either strict or vacuous in a given undetermined variable.

Lemma 4.5 (Tightening) *Let M be a closed simple term:*

1. *If $(\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow A$, then $\Gamma; \Omega; (\Delta, x{:}C) \vdash M \downarrow A$ or $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow A$.*
2. *If $(\Gamma, x{:}C); \Omega; \Delta \vdash M \Uparrow A$, then $\Gamma; \Omega; (\Delta, x{:}C) \vdash M \Uparrow A$ or $\Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow A$.*

Proof: By mutual induction on $\mathcal{D}_1 :: (\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow A$ and $\mathcal{D}_2 :: (\Gamma, x{:}C); \Omega; \Delta \vdash M \Uparrow A$.

Case:

$$\mathcal{D}_1 = \frac{c{:}A \in \Sigma}{(\Gamma, x{:}C); \Omega; \cdot \vdash c \downarrow A}\,cIdc$$

Then

$$\mathcal{E} = \frac{c{:}A \in \Sigma}{\Gamma; (\Omega, x{:}C); \cdot \vdash c \downarrow A}\,cIdc$$

Case:

$$\mathcal{D}_1 = \frac{}{(\Gamma, x{:}C, y{:}A); \Omega; \cdot \vdash y \downarrow A}\,cId^u$$

Then

$$\mathcal{E} = \frac{}{(\Gamma, y{:}A); (\Omega, x{:}C); \cdot \vdash y \downarrow A}\,cId^u$$

Case:

$$\mathcal{D}_1 = \frac{}{(\Gamma, x{:}C); \Omega; \cdot \vdash x \downarrow C}\,cId^u$$

Then

$$\mathcal{E} = \frac{}{\Gamma; \Omega; x{:}C \vdash x \downarrow C}\,cId^1$$

Case:

$$\mathcal{D}_1 = \frac{}{(\Gamma, x{:}C); \Omega; y{:}A \vdash y \downarrow A}\,Id^1$$

Then

$$\mathcal{E} = \frac{}{\Gamma; (\Omega, x{:}C); y{:}A \vdash y \downarrow A}\,Id^1$$

Case:

$$\mathcal{D}_2 = \frac{(\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow a}{(\Gamma, x{:}C); \Omega; \Delta \vdash M \Uparrow a}\,cAt$$

$(\Gamma, x{:}C); \Omega; \Delta \vdash M \downarrow a$ By sub-derivation
$\Gamma; \Omega; (\Delta, x{:}C) \vdash M \downarrow a$ or $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow a$ By IH 1

Subcase: $\Gamma; \Omega; (\Delta, x{:}C) \vdash M \downarrow a$ By assumption
$\Gamma; \Omega; (\Delta, x{:}C) \vdash M \Uparrow a$ By rule cAt

Subcase: $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow a$ By assumption
$\Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow a$ By rule cAt

Case:

$$\mathcal{D}_2 = \cfrac{(\Gamma, x{:}C, y{:}A); \Omega; \Delta \vdash M \Uparrow B}{(\Gamma, x{:}C); \Omega; \Delta \vdash (\lambda y^u : A.\, M) \Uparrow A \xrightarrow{u} B}\, c \xrightarrow{u} I$$

$(\Gamma, x{:}C, y{:}A); \Omega; \Delta \vdash M \Uparrow B$ By sub-derivation
$(\Gamma, y{:}A); \Omega; (\Delta, x{:}C) \vdash M \Uparrow B$ or $(\Gamma, y{:}A); (\Omega, x{:}C); \Delta \vdash M \Uparrow$ By IH 2

Subcase: $(\Gamma, y{:}A); \Omega; (\Delta, x{:}C) \vdash M \Uparrow B$ By assumption
$\Gamma; \Omega; (\Delta, x{:}C) \vdash (\lambda y^u : A.\, M) \Uparrow A \xrightarrow{u} B$ By rule $c \xrightarrow{u} I$

Subcase: $(\Gamma, y{:}A); (\Omega, x{:}C); \Delta \vdash M \Uparrow$: symmetrical

Case:

$$\mathcal{D}_1 = \cfrac{(\Gamma, x{:}C, \Delta_N); \Omega; \Delta_M \vdash M \downarrow A \xrightarrow{1} B \qquad (\Gamma, x{:}C, \Delta_M); \Omega; \Delta_N \vdash N \Uparrow A}{(\Gamma, x{:}C); \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B}\, c \xrightarrow{1} E$$

There are four sub-cases, stemming from IH 1 and 2:

1. $(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}C) \vdash M \downarrow A \xrightarrow{1} B$ By assumption
$(\Gamma \Delta_M); \Omega; (x{:}C, \Delta_N) \vdash N \Uparrow A$ By assumption
$(\Gamma, \Delta_M, x{:}C); \Omega; \Delta_N \vdash N \Uparrow A$ By Loosening^1 x
$\Gamma; \Omega; (\Delta_M, x{:}C, \Delta_N) \vdash M\ N^1 \downarrow B$ By rule

2. $(\Gamma, \Delta_N); (\Omega, x{:}C); \Delta_M \vdash M \downarrow A \xrightarrow{1} B$ By assumption
$(\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A$ By assumption
$\Gamma; (\Omega, x{:}C); (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B$ By rule

3. $(\Gamma, \Delta_N); \Omega; (\Delta_M, x{:}C) \vdash M \downarrow A \xrightarrow{1} B$ By assumption
$(\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A$ By assumption
$(\Gamma, x{:}C, \Delta_N); \Omega; \Delta_M \vdash N \Uparrow A$ By Loosening^0 x
$\Gamma; \Omega; (\Delta_M, x{:}C, \Delta_N) \vdash M\ N^1 \downarrow B$ By rule

4. Symmetrical to 3.

□

We remark that tightening fails to hold once we allow non-simple terms, namely intuitionistic application. For example $y{:}A \xrightarrow{u} B, x{:}A; \cdot; \cdot \vdash y\ x^u : B$ but both $y{:}A \xrightarrow{u} B; \cdot; x{:}A \not\vdash y\ x^u : B$ and $y{:}A \xrightarrow{u} B; x{:}A; \cdot \not\vdash y\ x^u : B$. This suggest that simple terms are not only a useful technical device to achieve term complement in the simply-typed case, but possibly for other more general calculi such as the linear λ-calculus.

Corollary 4.6 *Let M be a closed simple term such that $\Gamma; \Omega; \Delta \vdash M \Updownarrow A$; then there are Δ', Ω' such that $\Gamma = \Delta', \Omega'$ and $\cdot; (\Omega, \Omega'); (\Delta, \Delta') \vdash M \Updownarrow A$.*

Proof: By induction on Γ, using Lemma 4.5. □

$$\frac{\Omega = \mathrm{dom}(\Gamma) \setminus \mid \Phi \mid}{\Gamma \vdash E\ \Phi^u \longleftrightarrow Z\ \Phi^u\ \Omega^0}\ \texttt{TrnPat}$$

$$\frac{\Gamma, x{:}A \vdash M \longleftrightarrow N}{\Gamma \vdash \lambda x^u : A.\, M \longleftrightarrow \lambda x^u : A.\, N}\ \texttt{TrnLam}$$

$$\frac{\Gamma \vdash M_1 \longleftrightarrow N_1 \cdots \Gamma \vdash M_n \longleftrightarrow N_n}{\Gamma \vdash h\ \overline{M_n^1} \longleftrightarrow h\ \overline{N_n^1}}\ \texttt{TrnApp}$$

Figure 4.1: Full application translation: $\Gamma \vdash M \longleftrightarrow N$

4.1.2 Full Application

We can simplify the presentation of the algorithms for complement and later unification if we require any existential variable to be applied to every bound variable in its declaration context. This is possible for any simple *linear* pattern without changing the set of its ground instances. We just insert vacuous applications, which guarantees that the extra variables are not used. In a slight abuse of notation we call the resulting terms *fully applied.*

We describe in Figure 4.1.2 the judgment $\Gamma \vdash M \longleftrightarrow N$ which turns a term M into an equivalent fully applied term N: while we need this translation specialized to simple terms, it is clear how to generalize this judgment to the canonical forms of any strict term.

Example 4.7 *Recall the simple term from Example 2.8:*

$$lam\ (\lambda x^u : exp.\, app\ E\ x)$$

has fully applied form

$$lam\ (\lambda x^u : exp.\, app\ (Z\ x^0)\ x)$$

for a fresh existential variable Z of type $exp \stackrel{0}{\to} exp$.

We may check the the output of the translation is indeed fully applied w.r.t. its definition in Figure 2.3:

Lemma 4.8 *If M is a simple term and $\Gamma \vdash M \longleftrightarrow N$, then $\Gamma \vdash N$ f.a. .*

Proof: A straightforward induction on the structure of the given derivation. □

We have now arrived to the following language, where the labeling on flexible patterns is unrestricted, still called "simple terms":

$$\textit{Simple Terms} \quad M \quad ::= \quad \lambda x^u{:}A^+.\, M \mid (\ldots(h\ M_1)^1 \ldots M_n)^1 \mid (\ldots(E\ x_1^{l_1}) \ldots x_n^{l_n})$$

To prove the set-theoretic adequacy of the translation, we will need the following irrelevance Lemma.

Lemma 4.9 (Irrelevance) *If M is a closed simple term, then:*

1. *If $\Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow A$, then $\Gamma; \Omega; \Delta \vdash M \Uparrow A$.*
2. *If $\Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow A$, then $\Gamma; \Omega; \Delta \vdash M \downarrow A$.*

Proof: By mutual induction on $\mathcal{D}_1 :: \Gamma; (\Omega, x{:}C); \Delta \vdash M \downarrow A$ and $\mathcal{D}_2 :: \Gamma; (\Omega, x{:}C); \Delta \vdash M \Uparrow A$.

Case:

$$\mathcal{D}_1 = \frac{c{:}A \in \Sigma}{\Gamma; (\Omega, x{:}C); \cdot \vdash c \downarrow A}\ cIdc$$

Then

$$\mathcal{E} = \frac{c{:}A \in \Sigma}{\Gamma; \Omega; \cdot \vdash c \downarrow A}\ cIdc$$

Case:

$$\mathcal{D}_1 = \frac{}{(\Gamma, y{:}A); (\Omega, x{:}C); \cdot \vdash y \downarrow A}\, cId^u$$

Then

$$\mathcal{E} = \frac{}{(\Gamma, y{:}A); \Omega; \cdot \vdash y \downarrow A}\, cId^u$$

Case:

$$\mathcal{D}_1 = \frac{}{\Gamma; (\Omega, x{:}C); y{:}A \vdash y \downarrow A}\, cId^1$$

Then

$$\mathcal{E} = \frac{}{\Gamma; \Omega; y{:}A \vdash y \downarrow A}\, cId^1$$

Case: $\mathcal{D}_2$ ends in cAt: by IH 2.

Case:

$$\mathcal{D}_2 = \frac{(\Gamma, y{:}A); (\Omega, x{:}C); \Delta \vdash M \Uparrow B}{\Gamma; (\Omega, x{:}C); \Delta \vdash (\lambda y^u : A.\, M) \Uparrow A \xrightarrow{u} B}\, c \xrightarrow{u} I$$

$(\Gamma, y{:}A); (\Omega, x{:}C); \Delta \vdash M \Uparrow B$	By sub-derivation
$(\Gamma, y{:}A); \Omega; \Delta \vdash M \Uparrow B$	By IH 2
$\Gamma; \Omega; \Delta \vdash (\lambda y^u : A.\, M) \Uparrow A \xrightarrow{u} B$	By rule

Case:

$$\mathcal{D}_2 = \frac{(\Gamma, \Delta_N); (\Omega, x{:}C); \Delta_M \vdash M \downarrow A \xrightarrow{1} B \qquad (\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A}{\Gamma; (\Omega, x{:}C); (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B}\, c \xrightarrow{1} E$$

$(\Gamma, \Delta_N); (\Omega, x{:}C); \Delta_M \vdash M \downarrow A \xrightarrow{1} B$	By sub-derivation
$(\Gamma, \Delta_N); \Omega; \Delta_M \vdash M \downarrow A \xrightarrow{1} B$	By IH 2
$(\Gamma, \Delta_M); (\Omega, x{:}C); \Delta_N \vdash N \Uparrow A$	By sub-derivation
$(\Gamma, \Delta_M); \Omega; \Delta_N \vdash N \Uparrow A$	By IH 1
$\Gamma; \Omega; (\Delta_M, \Delta_N) \vdash M\ N^1 \downarrow B$	By rule

□

Note that irrelevance holds for any strict canonical terms, but it is false for terms containing redeces. For example $\cdot; x{:}A; \cdot \vdash (\lambda y^0 : A.\, c)\ x^0 : B$, as x becomes unbound in the rightmost premise, but $\cdot; \cdot; \cdot \not\vdash (\lambda y^0 : A.\, c)\ x^0 : B$.

Ground Instances

We recall that we assume every type to be inhabited, so that every term can be seen as the intensional representation of the set of its ground instances. The judgment in Figure 4.2 $\Gamma \vdash M \in \|N\| : A$ formalizes conditions for M to be a ground instance of a simple *linear* term N at type A. We then extend the judgment to *sets* of terms of the same type as follows:

$$\frac{\exists i{:}1 \leq i \leq n \quad \Gamma \vdash M \in \|N_i\| : A}{\Gamma \vdash M \in \|N_1 \cdots N_n\| : A}\, \mathtt{gr}_i$$

Remark 4.10 *$\Psi \vdash M \in \|E_A\ \Phi\| : a$ iff $M \downarrow t\ \Phi$ and $\cdot \vdash t : A$ iff $t \equiv \lambda\Phi\,.\,S$ and if $\Gamma; \Omega; \Delta \vdash \Phi$ ok, then $\Gamma; \Omega; \Delta \vdash S : A$.*

This fact will be heavily used in the following.

$$\frac{M \downarrow t\ \Phi \qquad \cdot \vdash t : A}{\Gamma \vdash M \in \| E_A\ \Phi \| : a}\ \texttt{grFlx}$$

$$\frac{(\Gamma, x{:}A) \vdash M \in \| N \| : B}{\Gamma \vdash \lambda x^u : A.\, M \in \| \lambda x^u : A.\, N \| : A \xrightarrow{u} B}\ \texttt{grLam}$$

$$\frac{\Gamma \vdash h : \overline{A_n} \xrightarrow{1} a \qquad \Gamma \vdash M_1 \in \| N_1 \| : A_1 \cdots \Gamma \vdash M_n \in \| A_n \| : A_n}{\Gamma \vdash h\ \overline{M_n^1} \in \| h\ \overline{N_n^1} \| : a}\ \texttt{grApp}$$

Figure 4.2: Ground instance: $\Gamma \vdash M \in \|N\| : A$

Lemma 4.11 (Ground Instance Weakening) *If* $\Gamma \vdash M \in \|N\| : A$ *then* $(\Gamma, x{:}A) \vdash M \in \|N\| : A$.

Proof: By induction on the structure of the given derivation. □

We implicitly use the above lemma to weaken different contexts with common basis to an unique one.

Now we can prove that the the full application translation preserves the set of ground instances.

Theorem 4.12 (Adequacy of the Full Application Translation) *Let* N *be a simple term of type* A*, such that* $\Gamma \vdash N \longleftrightarrow Q$*; then* $\Gamma \vdash M \in \|N\| : A$ *iff* $\Gamma \vdash M \in \|Q\| : A$.

Proof: By induction on the structure of $\mathcal{D} :: \Gamma \vdash M \longleftrightarrow N$ and inversion on $\Gamma \vdash M \in \|N\| : A$ and $\Gamma \vdash M \in \|Q\| : A$.

Case:

$$\mathcal{D} = \frac{\Omega = \mathrm{dom}(\Gamma) \setminus \mid \Phi \mid}{\Gamma \vdash E\ \Phi^u \longleftrightarrow Z\ \Phi^u\ \Omega^0}\ \texttt{TrnPat}$$

$(\rightarrow)$	$\Gamma \vdash M \in \|E_B\ \Phi^u\|$ and $\Omega = \mathrm{dom}(\Gamma) \setminus \mid \Phi \mid$	By assumption
	$M \downarrow t\ \Phi^u$ and $\cdot \vdash t : B$ for $\Phi; \cdot; \cdot \vdash \Phi^u\ ok$	By inversion
	$\Phi; \cdot; \cdot \vdash S : A$	By Remark 4.10
	$\Phi; \Omega; \cdot \vdash S : A$	By Weakening^0
	$M \downarrow t'\ \Phi^u\Omega^0$ for $\cdot \vdash t' : B'$ and $\Phi; \Omega; \cdot \vdash (\Phi^u, \Omega^0)\ ok$	By Remark 4.10
	$\Gamma \vdash M \in \|F_{B'}\ \Phi^u\Omega^0\|$	By rule
$(\leftarrow)$	$\Gamma \vdash M \in \|F_{B'}\ \Phi^u\Omega^0\|$ and $\Omega = \mathrm{dom}(\Gamma) \setminus \mid \Phi \mid$	By assumption
	$M \downarrow t'\ \Phi^u\Omega^0$ for $\cdot \vdash t' : B'$ and $\Phi; \Omega; \cdot \vdash (\Phi^u, \Omega^0)\ ok$	By inversion
	$\Phi; \Omega; \cdot \vdash S : A$	By Remark 4.10
	$\Phi; \Omega; \cdot \vdash S \Uparrow\!\!\Downarrow A$	By the canonical form Theorem
	$\Phi; \cdot; \cdot \vdash S \Uparrow\!\!\Downarrow A$	By Irrelevance (Lemma 4.9)
	$\Phi; \cdot; \cdot \vdash S : A$	By soundness (Lemma 3.16)
	$\Gamma \vdash M \in \|E_B\ \Phi^u\|$	By rule and Remark 4.10

Case: $\mathcal{D}$ ends in `TrnLam` or `TrnApp`: the result follows from a straightforward application of the inductive hypothesis.

□

From now on we tacitly assume that all simple terms are fully applied. We call a term $E\, x_1^{l_1} \ldots x_n^{l_n}$ a *generalized variable*.

4.2 The Complement Algorithm

The idea of complementation for applications and abstractions is quite simple and similar to the first-order case. For generalized variables we consider each argument in turn. If an argument variable is undetermined it does not contribute to the negation. If an argument variable is strict then any term where this variable does not occur contributes to the negation. We therefore complement the corresponding label from 0 to 1 while all other arguments are undetermined. For vacuous argument variables we proceed dually. If $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$, we write $E\ \Gamma^u$ for the application of $E\ {x_1}^u \ldots {x_n}^u$. Such an application represents the set of all terms without existential variables and free variables from Γ.

In preparation for the rules, we observe that the complement operation on terms behaves on labels like negation does on truth-values in Kleene's three-valued logic, in the sense of the following table:

$$\begin{array}{rcl} \mathrm{Not}(1) & = & 0 \\ \mathrm{Not}(0) & = & 1 \\ \mathrm{Not}(u) & = & u \end{array}$$

Note that these labels form a three-valued semi-lattice with the (reverse) partial information ordering $1 \leq u, 0 \leq u$.

Definition 4.13 (Higher-Order Pattern Complement) *Fix a signature Σ. For a linear simple term M such that $\Gamma \vdash M : A$ define $\Gamma \vdash \mathrm{Not}(M) \Rightarrow N : A$ by the following rules:*

$$\frac{\exists i{:}1 \leq i \leq n \quad k \in \{1,0\}}{\Gamma \vdash \mathrm{Not}(E\ x_1^{k_1} \ldots x_{i-1}^{k_{i-1}}\ x_i^{k}\ x_{i+1}^{k_{i+1}} \ldots x_n^{k_n}) \Rightarrow Z\ x_1^u \ldots x_{i-1}^u\ x_i^{\mathrm{Not}(k)}\ x_{i+1}^u \ldots x_n^u : a}\ \mathrm{Not}\texttt{Flx}$$

no rule for $k = u$

$$\frac{\Gamma, x{:}A \vdash \mathrm{Not}(M) \Rightarrow N : B}{\Gamma \vdash \mathrm{Not}(\lambda x^u{:}A.\, M) \Rightarrow \lambda x^u{:}A.\, N : A \overset{u}{\to} B}\ \mathrm{Not}\texttt{Lam}$$

$$\frac{g \in \Sigma \cup \Gamma, g : A_1 \overset{1}{\to} \ldots \overset{1}{\to} A_m \overset{1}{\to} a \quad m \geq 0, h \not\equiv g}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n^1}) \Rightarrow (\ldots(g\ (Z_1\ \Gamma^u))^1 \ldots (Z_m\ \Gamma^u))^1 : a}\ \mathrm{Not}\texttt{App}^1$$

$$\frac{\exists i{:}1 \leq i \leq n \qquad \Gamma \vdash \mathrm{Not}(M_i) \Rightarrow N : A_i}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n^1}) \Rightarrow (\ldots(h\ (Z_1\ \Gamma^u))^1 \ldots (Z_{i-1}\ \Gamma^u)^1\ N^1\ (Z_{i+1}\ \Gamma^u)^1 \ldots (Z_n\ \Gamma^u))^1 : a}\ \mathrm{Not}\texttt{App}^2$$

where the Z's are fresh logic variables of appropriate typing, $h \in \Sigma \cup \Gamma$ and $\Gamma \vdash h : A_1 \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} a$. Note that a given M may be related to several terms N all of which belong to the complement of M. Finally we define $\Gamma \vdash \mathrm{Not}(M) = \mathcal{N} : A$ if $\mathcal{N} = \{N \mid \Gamma \vdash \mathrm{Not}(M) \Rightarrow N : A\}$.

We may drop the type information from the above judgment in examples and proofs; we will write $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$, when $\Gamma \vdash \mathrm{Not}(N) = \mathcal{N}$ and $\Gamma \vdash M \in \|\mathcal{N}\| : A$.

Note that if E_A is a generalized variable considered in the empty context, it has the canonical form $\lambda \overline{x_n^u}.\, E\ \overline{x_n^u}$. Hence $\cdot \vdash \mathrm{Not}(E_A) = \emptyset$ as expected.

Example 4.14 *Let $\Gamma = x{:}B, y{:}C$:*

$$\begin{array}{rcl} \Gamma \vdash \mathrm{Not}(E\ x^u y^1) & = & \{F\ x^u y^0\} \\ \Gamma \vdash \mathrm{Not}(E\ x^0 y^1) & = & \{F\ x^1 y^u, G\ x^u y^0\} \end{array} \tag{4.1}$$

It is worthwhile to observe that the members of a complement set are not mutually disjoint, due to the indeterminacy of u. We can achieve an exclusive 'or' if we resolve this indeterminacy, that is by considering for every x^u the two possibilities x^1, x^0. Thus, for example, equation (4.1) may be made explicit into:

$$\{F\ x^1 y^1, G\ x^1 y^0, H\ x^0 y^0\}$$

It is clear that in the worst case scenario the number of terms in a complement set is bound by 2^n; hence the usefulness of this further step needs to be pragmatically determined.

Example 4.15 *In the signature of numerals:*

$$\begin{aligned}&\mathrm{Not}(\lambda x^u \lambda y^u.\, s(E\ x^1 y^0)^1 = \\ &\quad \{\lambda x^u \lambda y^u.\, x, \lambda x^u \lambda y^u.\, y, \lambda x^u \lambda y^u.\, 0, \lambda x^u \lambda y^u.\, s(Z\ x^0 y^u)^1, \lambda x^u \lambda y^u.\, s(Z'\ x^u y^1)^1\}\end{aligned}$$

We can now revisit[1] Example 2.8:

$$\begin{aligned}&\mathrm{Not}(lam(\lambda x^u{:}exp.\, app\ (E\ x^0)\ x)) = \\ &\quad \{lam(\lambda x^u{:}exp.\, app\ (Z\ x^1)\ (Z'\ x^u)), \\ &\quad lam(\lambda x^u{:}exp.\, app\ (Z\ x^u)\ (app\ (Z'\ x^u)\ (Z'' x^u)), \\ &\quad lam(\lambda x^u{:}exp.\, app\ (Z\ x^u)\ (lam(\lambda y^u{:}exp.\, Z'\ x^u\ y^u)), \\ &\quad lam(\lambda x^u{:}exp.\, lam(\lambda y^u{:}exp.\, Z\ x^u\ y^u)), \\ &\quad lam(\lambda x^u{:}exp.\, x), \\ &\quad app\ Z\ Z'\}\end{aligned}$$

It is easy to show that simple terms are closed under complementation.

Theorem 4.16 *If M is a simple term and $\Gamma \vdash \mathrm{Not}(M) \Rightarrow N : A$, then N is simple.*

Proof: By induction on the structure of $\mathcal{D} :: \Gamma \vdash \mathrm{Not}(M) \Rightarrow N : A$.

Case: $\mathcal{D}$ ends in Not`Flx`: immediate.

Case:

$$\mathcal{D} = \dfrac{\Gamma, x{:}A \vdash \mathrm{Not}(M) \Rightarrow N : B}{\Gamma \vdash \mathrm{Not}(\lambda x^u{:}A.\, M) \Rightarrow \lambda x^u{:}A.\, N : A \xrightarrow{u} B}\ \mathrm{Not}\mathtt{Lam}$$

By sub-derivation $\Gamma, x{:}A \vdash \mathrm{Not}(M) \Rightarrow N : B$, hence by IH N is simple and so is $\lambda x^u{:}A.\, N$.

Case:

$$\mathcal{D} = \dfrac{g \in \Sigma \cup \Gamma, g : A_1 \xrightarrow{1} \dots \xrightarrow{1} A_m \xrightarrow{1} a \quad m \geq 0, h \not\equiv g}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n^1}) \Rightarrow (\dots(g\ (Z_1\ \Gamma^u))^1 \dots (Z_m\ \Gamma^u))^1 : a}\ \mathrm{Not}\mathtt{App}^1$$

Since every $(Z_i\ \Gamma^u)$ is simple, so is $(\dots(g\ (Z_1\ \Gamma^u))^1 \dots (Z_m\ \Gamma^u))^1$.

Case: $\mathcal{D} =$

$$\dfrac{\Gamma \vdash \mathrm{Not}(M_i) \Rightarrow N : A_i \qquad 1 \leq i \leq n}{\Gamma \vdash \mathrm{Not}(h\ \overline{M_n^1}) \Rightarrow (\dots(h\ (Z_1\ \Gamma^u))^1 \dots (Z_{i-1}\ \Gamma^u)^1\ N^1\ (Z_{i+1}\ \Gamma^u)^1 \dots (Z_n\ \Gamma^u))^1 : a}\ \mathrm{Not}\mathtt{App}^2$$

By IH N is simple and as above so is every $(Z_i\ \Gamma^u)$. Thus
$(\dots(h\ (Z_1\ \Gamma^u))^1 \dots (Z_{i-1}\ \Gamma^u)\ N^1\ (Z_{i+1}\ \Gamma^u)^1 \dots (Z_n\ \Gamma^u))^1$ is simple.

□

Corollary 4.17 *If M is a simple term, and $\Gamma \vdash \mathrm{Not}(M) = \mathcal{N}$, then $\mathcal{N}$ is a set of simple terms.*

We address the soundness and completeness of the complement algorithm w.r.t. the set-theoretic semantics: the proof obligation consists in proving that the former does behave as a complement operation on sets of patterns, i.e. it satisfies disjointness and exhaustivity. Termination is obvious as the algorithm is syntax-directed and only finitely branching. We start with soundness: for $\Phi \equiv x_1^{k_1} \dots x_{i-1}^{k_{i-1}}\ x_i^d\ x_{i+1}^{k_{i+1}} \dots x_n^{k_n}$ let $\mathrm{Not}(\Phi) \equiv x_1^u \dots x_{i-1}^u\ x_i^{\mathrm{Not}(d)}\ x_{i+1}^u \dots x_n^u$.

[1] To avoid too many indices on existential variables, we adopt a convention that the scope of existential variables is limited to each member of a complement set.

Theorem 4.18 *Let $\Gamma \vdash N : A$ be a simple (linear) term: for every Q such that $\Gamma \vdash \mathrm{Not}(N) \Rightarrow Q : A$, it is not the case that both $\Gamma \vdash M \in \|N\| : A$ and $\Gamma \vdash M \in \|Q\| : A$.*

Proof: By induction on the structure of $\mathcal{D} :: \Gamma \vdash \mathrm{Not}(N) \Rightarrow Q$.

Case: $\mathcal{D}$ ends in Not`Flx`; assume $\Gamma \vdash M \in \|E\ \Phi\|$ and consider $x_i{:}A \in \Gamma$:

Subcase: $\Phi(x_i) = 1$:

$\Psi; \Omega; (\Delta, x_i{:}A) \vdash \Phi\ ok$	By definition
$\Gamma \vdash M \in \|\mathrm{Not}(E\ \Phi)\|$	By assumption
$\Gamma \vdash M \in \|Z\ \mathrm{Not}(\Phi)\|$	By rule Not`Flx`
$\Psi; \Omega; (\Delta, x_i{:}A) \vdash M : A$ and $(\Psi, \Omega, \Delta); x_i{:}A; \cdot \vdash M : A$	By Remark 4.10
$\perp$	By Corollary 3.14.

Subcase: $\Phi(x_i) = 0$: symmetrically.

Subcase: $\Phi(x_i) = u$: trivially true.

Case: $\mathcal{D}$ ends in Not`App`^{1}. Suppose both $\Gamma \vdash M \in \|h\ \overline{N_n^1}\|$ and $\Gamma \vdash M \in \|g\ (Z_1\ \Gamma^u)^1 \dots (Z_m\ \Gamma^u)^1\|$ for $h \not\equiv g$; but this is immediately impossible by rule `grApp` as the root of M should be both h and g.

Case: $\mathcal{D}$ ends in Not`App`^{2}.

$\Gamma \vdash \mathrm{Not}(h\ \overline{N_n^1}) \Rightarrow (\dots(h\ (Z_1\ \Gamma^u))^1 \dots (Z_{i-1}\ \Gamma^u)^1\ Q^1\ (Z_{i+1}\ \Gamma^u)^1 \dots (Z_n\ \Gamma^u))^1 : a$	By hypothesis
$\Gamma \vdash \mathrm{Not}(N_i) \Rightarrow Q : A_i$, for some $1 \leq i \leq n$	By sub-derivation
$\Gamma \vdash h\ \overline{M_n^1} \in \|h\ \overline{N_n^1}\| : a$	By assumption
$\Gamma \vdash h\ \overline{M_n^1} \in \|h\ (Z_1\ \Gamma^u)^1 \dots (Z_{i-1}\ \Gamma^u)^1\ Q^1\ (Z_{i+1}\ \Gamma^u)^1 \dots (Z_n\ \Gamma^u)^1\| : a$	By assumption
$\Gamma \vdash M_i \in \|N_i\| : A_i$	By inversion
$\Gamma \vdash M_i \in \|Q\| : A_i$	By inversion
$\perp$	By IH

Case: $\mathcal{D}$ ends in Not`Lam`:

$\Gamma \vdash \mathrm{Not}(\lambda x^u{:}A.\,N) \Rightarrow \lambda x^u{:}A.\,Q : A \xrightarrow{u} B$	By hypothesis
$\Gamma, x{:}A \vdash \mathrm{Not}(N) \Rightarrow Q : B$	By sub-derivation
$\Gamma \vdash \lambda x^u{:}A.\,M \in \|\lambda x^u{:}A.\,N\| : A \xrightarrow{u} B$	By hypothesis
$\Gamma \vdash \lambda x^u{:}A.\,M \in \|\lambda x^u{:}A.\,Q\| : A \xrightarrow{u} B$	By assumption
$\Gamma, x{:}A \vdash M \in \|N\| : B$	By inversion
$\Gamma, x{:}A \vdash M \in \|Q\| : B$	By inversion
$\perp$	By IH

□

Note that soundness is based on Corollary 3.14, which holds for *any* strict term: thus disjointness does not require simple terms.

Lemma 4.19 *Assume $\Gamma \vdash E_A\ \Phi : a$; either $\Gamma \vdash M \in \|E_A\ \Phi\| : a$ or there exists $1 \leq i \leq n$ such that $\Gamma \vdash \mathrm{Not}(E_A\ \Phi) \Rightarrow Z\ \mathrm{Not}(\Phi) : a$ and $\Gamma \vdash M \in \|Z\ \mathrm{Not}(\Phi)\| : a$.*

Proof: Let $\Gamma; \cdot; \cdot \vdash M : A$; then by Corollary 4.6 there exists Ω and Δ such that $\Gamma = \Omega, \Delta$ and $\cdot; \Omega; \Delta \vdash M \Uparrow\!\!\Downarrow A$. Fix $x_i^{k_i}$ for $1 \leq i \leq n$:

Case: For every $x \in \mathrm{dom}(\Omega)$ such that $x = x_i^{k_i}$ it holds $k_i \in \{0, u\}$ and for every $x \in \mathrm{dom}(\Delta)$, $k_i \in \{1, u\}$. Then $\Gamma \vdash M \in \|E\ \Phi\|$.

Case: For some $x \in \mathrm{dom}(\Omega)$ such that $x = x_i^{k_i}$ it holds $k_i = 1$. Then $\Gamma \vdash M \in \|Z\ x_1^u \dots x_{i-1}^u\ x_i^1\ x_{i+1}^u \dots x_n^u\|$, that is $\Gamma \vdash M \in \|Z|\mathrm{Not}(\Phi)\|$.

$$\frac{M \downarrow t\ \Phi \qquad \cdot \not\vdash t : A}{\Gamma \vdash M \notin \|E_A\ \Phi\| : a}\ \texttt{ngrFlx}$$

$$\frac{\Gamma, x{:}A \vdash M \notin \|N\| : B}{\Gamma \vdash \lambda x^u{:}A.\, M \notin \|\lambda x^u{:}A.\, N\| : A \xrightarrow{u} B}\ \texttt{ngrLam}$$

$$\frac{\Gamma \vdash h : \overline{A_n} \xrightarrow{1} a \qquad \exists i : 1 \leq i \leq n.\, \Gamma \vdash M_i \notin \|N_i\| : A_i}{\Gamma \vdash h\ \overline{M_n^1} \notin \|h\ \overline{N_n^1}\| : a}\ \texttt{ngrApp}$$

$$\frac{g \not\equiv h}{\Gamma \vdash g\ \overline{M_m} \notin \|h\ \overline{N_n^1}\| : a}\ \texttt{ngrAppCls}$$

Figure 4.3: Not a ground instance: $\Gamma \vdash M \notin \|N\| : A$

Case: For some $x \in \mathrm{dom}(\Delta)$ such that $x = x_i^{k_i}$ it holds $k_i = 0$. Then $\Gamma \vdash M \in \|Z\ x_1^u \ldots x_{i-1}^u\ x_i^0\ x_{i+1}^u \ldots x_n^u\|$, that is $\Gamma \vdash M \in \|Z\ \mathrm{Not}(\Phi)\|$.

□

For technical reasons, we need the rules complementary to Figure 4.2, which are depicted in Figure 4.3: We are now ready to prove exhaustivity of complementation.

Theorem 4.20 *Assume* $\Gamma \vdash N : A$ *is a simple (linear) term; then if* $\Gamma \vdash M \notin \|N\| : A$*, then there is a* Q *such that* $\Gamma \vdash \mathrm{Not}(N) \Rightarrow Q : A$ *and* $\Gamma \vdash M \in \|Q\| : A$*.*

Proof: By induction on the structure of $\mathcal{D} :: \Gamma \vdash M \notin \|N\| : A$.

Case: $\mathcal{D}$ ends in `ngrFlx`: by Lemma 4.19.

Case: $\mathcal{D}$ ends in `ngrAppCls`.

$\Gamma \vdash g\ \overline{M_m} \notin \|h\ \overline{N_n^1}\| : a$	By hypothesis
$\Gamma \vdash \mathrm{Not}(h\ \overline{N_n^1}) \Rightarrow (\ldots(g\ (Z_1\ \Gamma^u))^1 \ldots (Z_m\ \Gamma^u))^1 : a$	By rule NotApp[1]
$\Gamma \vdash g\ \overline{M_m} \in \|g\ (Z_1\ \Gamma^u)^1 \ldots (Z_m\ \Gamma^u)^1\| : a$	By rule `grApp`

Case: $\mathcal{D}$ ends in `ngrApp`:

$\Gamma \vdash h\ \overline{M_n^1} \notin \|h\ \overline{N_n^1}\| : a$	By hypothesis
$\Gamma \vdash M_i \notin \|N_i\| : A_i$ for some $1 \leq i \leq n$	By sub-derivation
$\Gamma \vdash \mathrm{Not}(N_i) \Rightarrow Q : A_i$ and $\Gamma \vdash M_i \in \|Q\| : A_i$	By IH
$\Gamma \vdash M_j \in \|(Z_j\ \ \Gamma^u)^1\|$ for all $j \not= i, 1 \leq j \leq n$	By rule `grFlx`
$\Gamma \vdash \mathrm{Not}(h\ \overline{M_n^1}) \Rightarrow (\ldots(h\ (Z_1\ \Gamma^u))^1 \ldots (Z_{i-1}\ \Gamma^u)^1\ N^1\ (Z_{i+1}\ \Gamma^u)^1 \ldots (Z_n\ \Gamma^u))^1 : a$	By rule Not`Lam`
$\Gamma \vdash h\ \overline{M_n^1} \in \|h\ (Z_1\ \Gamma^u)^1 \ldots (Z_{i-1}\ \Gamma^u)^1\ Q^1\ (Z_{i+1}\ \Gamma^u)^1 \ldots (Z_m\ \Gamma^u)^1\| : a$	By rule `grApp`.

Case: $\mathcal{D}$ ends in `ngrLam`.

$\Gamma \vdash \lambda x^u{:}A.\, M \notin \|\lambda x^u{:}A.\, N\| : A \xrightarrow{u} B$	By hypothesis
$\Gamma, x{:}A \vdash M \notin \|N\| : B$	By sub-derivation
$\Gamma, x{:}A \vdash \mathrm{Not}(N) \Rightarrow Q : B$ and $\Gamma, x{:}A \vdash M \in \|Q\| : B$ for some Q	By IH
$\Gamma \vdash \mathrm{Not}(\lambda x^u{:}A.\, N) \Rightarrow \lambda x^u{:}A.\, Q : A \xrightarrow{u} B$	By rule Not`Lam`
$\Gamma \vdash \lambda x^u{:}A.\, M \in \|\lambda x^u{:}A.\, Q\| : A \xrightarrow{u} B$	By rule `grLam`

□

Corollary 4.21 (Partition Lemma) *For a fixed signature* Σ *let* $\Gamma \vdash N : A$ *be a (linear) simple term:*

1. *(Disjointness) It is not the case that* $\Gamma \vdash M \in \|N\| : A$ *and* $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$.
2. *(Exhaustivity)* $\Gamma \vdash M \in \|N\| : A$ *or* $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$.

Proof: Disjointness is entailed by Theorem 4.18, exhaustivity by Theorem 4.20. □

4.3 Unification of Simple Terms

As we observed earlier, we can solve a relative complement problem pairing complementation with intersection. We thus address now the task of giving an algorithm for unification of (linear) simple terms. We start by determining when two labeling are compatible:

$$\begin{aligned} 1 \cap 1 = u \cap 1 = 1 \cap u &= 1 \\ 0 \cap 0 = u \cap 0 = 0 \cap u &= 0 \\ u \cap u &= u \end{aligned}$$

Recall that Φ is a list of labeled bound variables; we can extend the intersection operations to these contexts.

$$\begin{array}{rcll} \cdot \cap \cdot & = & \cdot & \\ (\Phi, x^k) \cap (\Phi', x^{k'}) & = & (\Phi \cap \Phi', x^{k \cap k'}) & \text{if } k \cap k' \text{ is defined.} \end{array}$$

Remark 4.22 *If* $\Gamma_i; \Omega_i; \Delta_i \vdash \Phi_i$ *ok,* $1 \leq i \leq 2$*, then* $(\Gamma_1 \cap \Gamma_2); (\Omega_1, \Omega_2); (\Delta_1, \Delta_2) \vdash (\Phi_1 \cap \Phi_2)$ *ok, where* $\Gamma_1 \cap \Gamma_2$ *denotes set-theoretic intersection and* $(\Phi_1 \cap \Phi_2)(x) \stackrel{def}{=} \Phi_1(x) \cap \Phi_2(x)$*. Indeed,* $(\Phi_1 \cap \Phi_2)(x) = u$ *iff* $x \in dom(\Gamma_1)$ *and* $x \in dom(\Gamma_2)$*; moreover* $(\Phi_1 \cap \Phi_2)(x) = 0$ *iff either* $x \in dom(\Omega_1 \cup \Gamma_2)$ *or* $x \in dom(\Omega_2 \cup \Gamma_1)$*. Analogously for* $(\Phi_1 \cap \Phi_2)(x) = 1$*. From that, as before, it follows that* $\Psi \vdash M \in \|E_A \; \Phi_1 \cap \Phi_2\|$ *iff* $M \downarrow \lambda\Phi_1 \cap \Phi_2 \,.\, S$ *such that* $(\Gamma_1 \cap \Gamma_2); (\Omega_1, \Omega_2); (\Delta_1, \Delta_2) \vdash S : A$*.*

Following standard terminology with call atomic terms whose head is a free or bound variable *rigid*, while terms whose head is an existential variable is called *flexible*.

Definition 4.23 (Higher-Order Pattern Intersection) *Fix a signature* Σ*. For simple (linear) terms* M *and* N *without shared variables such that* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$*, define* $\Gamma \vdash M \cap N \Rightarrow Q : A$ *by the following rules:*

$$\frac{}{\Gamma \vdash (E_1 \; \Phi_1) \cap (E_2 \; \Phi_2) \Rightarrow H \; (\Phi_1 \cap \Phi_2) : a} \cap FF$$

no rule for flex-flex same

$$\frac{c \in \Sigma \quad \Gamma \vdash (H_1 \; \Phi_1) \cap M_1 \Rightarrow N_1 : A_1 \cdots \Gamma \vdash (H_n \; \Phi_n) \cap M_n \Rightarrow N_n : A_n}{\Gamma \vdash (E \; \Phi) \cap (c \; \overline{M_n^1}) \Rightarrow c \; \overline{N_n^1} : a} \cap FR^c$$

$$\frac{y \in \Gamma \qquad \Gamma \vdash (H_1 \; \Phi_1) \cap M_1 \Rightarrow N_1 : A_1 \cdots \Gamma \vdash (H_n \; \Phi_n) \cap M_n \Rightarrow N_n : A_n}{\Gamma \vdash (E \; \Phi) \cap (y \; \overline{M_n^1}) \Rightarrow y \; \overline{N_n^1} : a} \cap FR^y$$

$$\frac{h \in \Gamma \cup \Sigma \quad \Gamma \vdash M_1 \cap N_1 \Rightarrow Q_1 : A_1 \cdots \Gamma \vdash M_n \cap N_n \Rightarrow Q_n : A_n}{\Gamma \vdash h \; \overline{M_n^1} \cap h \; \overline{N_n^1} \Rightarrow h \; \overline{Q_n^1} : a} \cap RR$$

$$\frac{\Gamma, x{:}A \vdash M \cap N \Rightarrow Q : B}{\Gamma \vdash \lambda x^u{:}A.\, M \cap \lambda x^u{:}A.\, N \Rightarrow \lambda x^u{:}A.\, Q : A \stackrel{u}{\rightarrow} B} \cap L$$

where the H's are fresh variables of appropriate typing and $n \geq 0$. We omit two rules $\cap RF^c$ and $\cap RF^y$ which are symmetric to $\cap FR^c$ and $\cap FR^y$. The rules $\cap FR^c$ and $\cap RF^c$ have the following proviso: for all $x \in \Phi$ and $1 \leq i, j \leq n$:

$$\begin{aligned}
\forall x.\Phi(x) = 0 &\to \forall i\,.\,\Phi_i(x) = 0 \\
\forall x.\Phi(x) = u &\to \forall i\,.\,\Phi_i(x) = u \\
\forall x.\Phi(x) = 1 &\to \exists i\,.\,\Phi_i(x) = 1 \wedge \forall j, j \neq i.\Phi_j(x) = u
\end{aligned}$$

The rules $\cap FR^y$ and $\cap RF^y$ are subject to the proviso:

$$\begin{aligned}
\forall x.\Phi(x) = 0 &\to \forall i\,.\,\Phi_i(x) = 0 \\
\forall x.\Phi(x) = u &\to \forall i\,.\,\Phi_i(x) = u \\
\forall x.x \neq y \wedge \Phi(x) = 1 &\to \exists i\,.\,\Phi_i(x) = 1 \wedge \forall j, j \neq i.\Phi_j(x) = u \\
\Phi(y) = u &\vee (\Phi(y) = 1 \wedge \forall i.\Phi_i(y) = u)
\end{aligned}$$

Finally define $\Gamma \vdash M \cap N : A = \mathcal{Q}$ if $\mathcal{Q} = \{Q \mid \Gamma \vdash M \cap N \Rightarrow Q : A\}$.

Some remarks are in order:

- In rule $\cap FF$ we can assume that the same list of variables, though with different labeling, is the argument of E, F and H, since simple terms are fully-applied and due to linearity we can always reorder the context to the same list.
- Since patterns are linear and M and N share no pattern variables, the flex-flex case arises only with distinct variables. This also means we do not have to apply substitutions or perform the customary occurs-check.
- In the flex/rigid and rigid/flex rules, the proviso enforces the typing discipline since each strict variable x must be strict in some premise. If instead y is the projected variable, the modified condition on y takes into account that the head of an application constitutes a strict occurrence; moreover, since y did occur, it is set to u in the rest of the computation, as there are no more requirements on that variable.
- The symmetric rules take the place of an explicit exchange rule that is problematic w.r.t. termination.

The following example illustrates how the Flex-Rigid rules, in this case $\cap FR^c$, make unification on simple terms finitary.

Example 4.24 *Consider the unification problem*

$$x{:}A \vdash E\ x^1 \cap\ c\ (F\ x^u)^1\ (F'\ x^u)^1$$

Since x is strict in the LHS, there are two ways in which Φ can be 'split' leading to the following sub-problems:

$$\begin{array}{lll}
1. & x{:}A \vdash E'\ x^1 \cap\ F\ x^u \Rightarrow H\ x^1 & x{:}A \vdash E''\ x^u \cap\ F'\ x^u \Rightarrow H'\ x^u \\
2. & x{:}A \vdash E'\ x^u \cap\ F\ x^u \Rightarrow H\ x^u & x{:}A \vdash E''\ x^1 \cap\ F'\ x^u \Rightarrow H'\ x^1
\end{array}$$

Hence the result:

$$x{:}A \vdash E\ x^1 \cap\ c\ (F\ x^u)^1\ (F'\ x^u)^1 = \{c\ (H\ x^1)^1\ (H'\ x^u)^1, c\ (H\ x^u)^1\ (H'\ x^1)^1\}$$

Note that, similarly to complementation, intersection return a solution with some 'overlapping' possible; again it is possible, in a post-processing phase to make the result exclusive: for example the above problem can be made explicit in:

$$\begin{aligned}
&x{:}A \vdash E\ x^1 \cap\ c\ (F\ x^u)^1\ (F'\ x^u)^1 = \\
&\quad \{c\ (H\ x^1)^1\ (H'\ x^0)^1, c\ (H\ x^0)^1\ (H'\ x^1)^1, c\ (H\ x^1)^1\ (H'\ x^1)^1\}
\end{aligned}$$

However, differently from complementation, we must remark that the latter is not the most general solution w.r.t. the subsumption ordering on terms based on the (reverse) partial information ordering on labels. Indeed, a member of the intersection, e.g. $c\ (H\ x^1)^1\ (H'\ x^0)^1$ is a lower bound of both terms above, i.e.

$$\begin{array}{rcl} c\ (H\ x^1)^1\ (H'\ x^0)^1 & \leq & E\ x^1 \\ c\ (H\ x^1)^1\ (H'\ x^0)^1 & \leq & c\ (H\ x^u)^1\ (H'\ x^u)^1 \end{array}$$

but it is not the greatest upper bound:

$$c\ (H\ x^1)^1\ (H'\ x^0)^1 \leq c\ (H\ x^1)^1\ (H'\ x^u)^1$$

The following example illustrates the additional proviso on $\cap FR^y$:

Example 4.25 *The unification problem* $y{:}A \vdash (E\ y^0) \cap (y\ (F\ y^u)^1\ (F'\ y^u)^1)$ *has no solution, whereas* $y{:}A \vdash (E\ y^1) \cap (y\ (F\ y^1)^1\ (F'\ y^0)^1) = \{y\ (H\ y^1)^1\ (H'\ y^0)^1\}$.

Lemma 4.26 *Let* M *be a closed simple term such that* $\Gamma_1;\Omega_1;\Delta_1 \vdash M : A$ *and* $\Gamma_2;\Omega_2;\Delta_2 \vdash M : A$ *; then* $\Gamma_1;\Omega_1;\Delta_1 \vdash M : A$ *and* $\Gamma_2;\Omega_2;\Delta_2 \vdash M : A$ *iff* $(\Gamma_1 \cap \Gamma_2);(\Omega_1,\Omega_2);(\Delta_1,\Delta_2) \vdash M : A$.

Proof: $(\rightarrow)$ By induction on the size of $(\Gamma_1 \cup \Gamma_2) \setminus (\Gamma_1 \cap \Gamma_2)$.

Base Let $\Gamma_1 = \Gamma_2$, thus $(\Gamma_1 \cup \Gamma_2) \setminus (\Gamma_1 \cap \Gamma_2) = \emptyset$. Then by Exclusivity (Lemma 3.13) $\Omega_1 = \Omega_2$ and $\Delta_1 = \Delta_2$ and the claim holds.

Step Let $\Gamma_1 = (\Gamma_1', x{:}C)$, where $x \notin \mathrm{dom}(\Gamma_2)$. By Tightening (Lemma 4.5) either $\Gamma_1';(\Omega_1, x{:}C);\Delta_1 \vdash M : A$ or $\Gamma_1';\Omega_1;(\Delta_1, x{:}C) \vdash M : A$:

Subcase:	$\Gamma_1';(\Omega_1, x{:}C);\Delta_1 \vdash M : A$	By assumption
	$(\Gamma_1' \cap \Gamma_2);(\Omega_1, x{:}C, \Omega_2);(\Delta_1,\Delta_2) \vdash M : A$	By IH
	$(\Gamma_1 \cap \Gamma_2);(\Omega_1, x{:}C, \Omega_2);(\Delta_1,\Delta_2) \vdash M : A$	$x \notin \mathrm{dom}(\Gamma_2)$
	$(\Gamma_1 \cap \Gamma_2);(\Omega_1,\Omega_2);(\Delta_1,\Delta_2) \vdash M : A$	$x \in \mathrm{dom}(\Omega_2)$ by Lemma 3.13

Subcase: $\Gamma_1';\Omega_1;(\Delta_1, x{:}C) \vdash M : A$. Analogously.

$(\leftarrow)$

$(\Gamma_1 \cap \Gamma_2);(\Omega_1,\Omega_2);(\Delta_1,\Delta_2) \vdash M : A$	By hypothesis
$(\Gamma_1 \cap \Gamma_2, \Omega_2);\Omega_1;(\Delta_1,\Delta_2) \vdash M : A$	By Loosening^0 on Ω_2
$(\Gamma_1 \cap \Gamma_2, \Omega_2, \Delta_2);\Omega_1;\Delta_1 \vdash M : A$	By Loosening^1 on Δ_2
$\Gamma_1;\Omega_1;\Delta_1 \vdash M : A$	Since $(\Gamma_1 \cap \Gamma_2),\Omega_2,\Delta_2,\Omega_1,\Delta_1 = \Gamma_1,\Omega_1,\Delta_1$
$\Gamma_2;\Omega_2;\Delta_2 \vdash M : A$	Analogously

□

We introduce two n-ary strict application rules, one which correspond to the imitation step and the other to projection, which will be needed in the proof of Theorem 4.33 and 4.34; in the following we shorten $x \in \mathrm{dom}(\Gamma)$ to $x \in \Gamma$.

$$\frac{(\Gamma, \Delta_i^u);\Omega;\Delta_i^1 \vdash M_i : A_i \quad 1 \leq i \leq n}{\Gamma;\Omega;\Delta \vdash c\ M_1^1 \ldots M_n^1 : B} \stackrel{1}{\rightarrow} E_n^c$$

where $\Gamma;\Omega;\cdot \vdash c\ : A_1 \stackrel{1}{\rightarrow} \ldots \stackrel{1}{\rightarrow} A_n \stackrel{1}{\rightarrow} B$ and

1. $\forall x \in \Delta\,.\,\exists i : 1 \leq i \leq n\,.\,x \in \Delta_i^1$.
2. $\forall i : 1 \leq i \leq n\,.\,\Delta_i^u \cup \Delta_i^1 = \Delta$.

$$\frac{(\Gamma, \Delta_i^u); \Omega; \Delta_i^1 \vdash M_i : A_i \quad 1 \leq i \leq n}{\Gamma; \Omega; \Delta \vdash y\ M_1^1 \ldots M_n^1 : B} \overset{1}{\to} E_n^y$$

where $\Gamma_0; \Omega; \Delta_0 \vdash y : A_1 \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$ and

1. $\forall x \in \Delta, x \neq y \,.\, \exists i : 1 \leq i \leq n \,.\, x \in \Delta_i^1$.
2. $\forall i : 1 \leq i \leq n \,.\, \Delta_i^u \cup \Delta_i^1 = \Delta$.
3. $\Delta_0 = \{y\}$ and $\forall i : 1 \leq i \leq n. y \in \Delta_i^u$.

Note that in the latter rule we consider only the case where y occurs strictly, that is $\Delta_0 = \{y\}$; indeed, if $y \in \Gamma_0$, then the rule is just a renaming of the previous rule $\overset{1}{\to} E_n^c$.

We proceed to show that both are derivable and invertible rules.

Lemma 4.27 *Let* $\Gamma; \Omega; \cdot \vdash c : A_1 \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$*: if* $(\Gamma, \Delta_i^u); \Omega; \Delta_i^1 \vdash M_i : A_i$ *for* $1 \leq i \leq j$*, then* $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash c\ M_1^1 \ldots M_j^1 : A_{j+1} \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$*, where*

1. $\forall x \in \Delta^1. \exists i : 1 \leq i \leq j \,.\, x \in \Delta_i^1$.

1'. $\forall x \in \Delta^u. \neg\exists i : 1 \leq i \leq j \,.\, x \in \Delta_i^1$.

2. $\forall i : 1 \leq i \leq j \,.\, \Delta_i^u \cup \Delta_i^1 = \Delta^u \cup \Delta^1$.

Proof: By induction on j.

j = 0 Set $\Delta^1 = \cdot$; then by hypothesis and Weakening^u $(\Gamma, \Delta^u); \Omega; \cdot \vdash c : A_1 \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$.

j + 1

$(\Gamma, \Delta_i^u); \Omega; \Delta_i^1 \vdash M_i : A_i, 1 \leq i \leq j$	By assumption
$(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash c\ M_1^1 \ldots M_j^1 : A_{j+1} \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$	By IH
$(\Gamma, \Delta_{j+1}^u); \Omega; \Delta_{j+1}^1 \vdash M_{j+1} : A_{j+1}$	By hypothesis
$(\Gamma, \Delta_+^u); \Omega; \Delta_+^1 \vdash c\ M_1^1 \ldots M_{j+1}^1 : A_{j+2} \overset{1}{\to} \ldots \overset{1}{\to} A_n \overset{1}{\to} B$	By rule $\overset{1}{\to} E$

such that:

(a) $x \in \Delta_+^1 \leftrightarrow x \in \Delta_{j+1}^1$ or $x \in \Delta^1$.

(b) $x \in \Delta_+^u \leftrightarrow x \in \Delta_{j+1}^u$ and $x \in \Delta^u$.

We now show that the last step satisfy the conditions in the claim.

1. $x \in \Delta_+^1$	By assumption
$x \in \Delta_{j+1}^1 \cup \Delta^1$	By (a)
Subcase: $x \in \Delta_{j+1}^1$	
$\exists i{:}1 \leq i \leq j+1 \,.\, x \in \Delta_i^1$	
Subcase: $x \in \Delta^1$	
$x \in \Delta_i^1$, for some $1 \leq i \leq j$	By IH
$\exists i{:}1 \leq i \leq j+1 \,.\, x \in \Delta_i^1$	A fortiori
1'. $x \in \Delta_+^u$	By hypothesis
$x \in \Delta^u$	By (b)
$\neg\exists i{:}1 \leq i \leq j \,.\, x \in \Delta_i^1$	By IH
$x \in \Delta_{j+1}^u$	By (b)
$x \notin \Delta_{j+1}^1$	By disjointness of contexts
$\neg\exists i{:}1 \leq i \leq j+1 \,.\, x \in \Delta_i^1$	

2. $\forall i{:}1 \le i \le j\,.\,\Delta^1_i \cup \Delta^u_i = \Delta^1 \cup \Delta^u$ — By IH
$\Delta^1_{j+1} \cup \Delta^u_{j+1} = \Delta^1 \cup \Delta^u$ — By hypothesis
$\forall i{:}1 \le i \le j+1\,.\,\Delta^1_i \cup \Delta^u_i = \Delta^1_+ \cup \Delta^u_+$ — By (a) and (b) and set manipulations

□

Lemma 4.28 *Let* $\Gamma; \Omega; \cdot \vdash c : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$*: if* $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash c\ M^1_1 \ldots M^1_j : A_{j+1} \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$*, then for every* $1 \le i \le j$ *there are* Δ^u_i, Δ^1_i *such that* $(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash M_i : A_i$ *and*

1. $\forall x \in \Delta^1 . \exists i : 1 \le i \le j\,.\,x \in \Delta^1_i$.

1'. $\forall x \in \Delta^u . \neg\exists i : 1 \le i \le j\,.\,x \in \Delta^1_i$.

2. $\forall i : 1 \le i \le j\,.\,\Delta^u_i \cup \Delta^1_i = \Delta^u \cup \Delta^1$.

Proof: By induction on j.

j = 0 Set $\Delta^1 = \cdot$; then by hypothesis and Weakening^u $(\Gamma, \Delta^u); \Omega; \cdot \vdash c : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$.

j + 1 $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash c\ M^1_1 \ldots M^1_{j+1} : A_{j+2} \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$ — By assumption
$(\Gamma, \Delta^u_+); \Omega; \Delta^1_+ \vdash c\ M^1_1 \ldots M^1_j : A_{j+1} \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$ and
$(\Gamma, \Delta^u_{j+1}); \Omega; \Delta^1_{j+1} \vdash M_{j+1} : A_{j+1}$ — By inversion on rule $\xrightarrow{1} E$ and
$(x \in \Delta^1_+ \leftrightarrow x \in \Delta^1_{j+1} \vee x \in \Delta^1)$ and $(x \in \Delta^u_+ \leftrightarrow x \in \Delta^u_{j+1} \wedge x \in \Delta^u)$
$(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash M_i : A_i$ for $1 \le i \le j$ — By IH

since the conditions on the claim are satisfied as in the above Lemma 4.27.

□

Corollary 4.29 *Rule* $\xrightarrow{1} E^c_n$ *is derivable and invertible.*

Proof: For derivability, use Lemma 4.27 with $j = n$, $\Delta^u = \cdot, \Delta^1 = \Delta$; conditions 1. and 2. are immediately satisfied. Ditto w.r.t. invertibility, using Lemma 4.28. □

Lemma 4.30 *Let* $\Gamma_0; \Omega; y{:}\ldots \vdash y : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$*. If* $(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash M_i : A_i$ *for* $1 \le i \le j$*, then* $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash y\ M^1_1 \ldots M^1_j : A_{j+1} \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$ *and*

1. $\forall x \in \Delta^1 . \exists i : 1 \le i \le j\,.\,x \in \Delta^1_i$.

1'. $\forall x \in \Delta^u . \neg\exists i : 1 \le i \le j\,.\,x \in \Delta^1_i$.

2. $\forall i : 1 \le i \le j\,.\,\Delta^u_i \cup \Delta^1_i = \Delta^u \cup \Delta^1$.

Proof: By induction on j.

j = 0 Let $\Delta_1 = \{y\}$ and weaken Γ_0 to Γ, Δ^u; by hypothesis $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash yA_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$.

j + 1 Completely analogous to the same case in Lemma 4.27.

□

Lemma 4.31 *Let* $\Gamma_0; \Omega; y{:}\ldots \vdash y : A_1 \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$*: if* $(\Gamma, \Delta^u); \Omega; \Delta^1 \vdash c\ M^1_1 \ldots M^1_j : A_{j+1} \xrightarrow{1} \ldots \xrightarrow{1} A_n \xrightarrow{1} B$*, then for every* $1 \le i \le j$ *there are* Δ^u_i, Δ^1_i *such that* $(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash M_i : A_i$ *and*

1. $\forall x \in \Delta^1 . \exists i : 1 \le i \le j\,.\,x \in \Delta^1_i$.

1'. $\forall x \in \Delta^1 . \neg\exists i : 1 \le i \le j\,.\,x \in \Delta^u_i$.

2. $\forall i : 1 \le i \le j\,.\,\Delta^u_i \cup \Delta^1_i = \Delta^u \cup \Delta^1$.

Proof: By induction on j similarly to Lemma 4.28. □

Corollary 4.32 *Rule $\stackrel{1}{\to} E^y_n$ is derivable and invertible.*

Proof: As in Corollary 4.29, using Lemma 4.30 and 4.31. □

We are now ready to address soundness and completeness of intersection:

Theorem 4.33 *For any simple linear term N_1 and N_2 without shared variables such that $\Psi \vdash N_1 : A$ and $\Psi \vdash N_2 : A$, if $\Psi \vdash M \in \|N_1\| : A$ and $\Psi \vdash M \in \|N_2\| : A$, then there is N such that $\Psi \vdash N_1 \cap N_2 \Rightarrow N : A$ and $\Psi \vdash M \in \|N\| : A$.*

Proof: By simultaneous induction on the structure of $\mathcal{D}_1 :: \Psi \vdash M \in \|N_1\| : A$ and $\mathcal{D}_2 :: \Psi \vdash M \in \|N_2\| : A$.

Case: $\mathcal{D}_1, \mathcal{D}_2$ end in `grFlx`; by Remark 4.22 and the left-to-right direction of Lemma 4.26.

Case: $\mathcal{D}_1$ ends in `grFlx` and $\mathcal{D}_2$ ends in `grApp`: there are two cases depending whether the head of N_2 is a constant or a bound variable:

Imit	$\Psi \vdash M \in \|c\ \overline{Q^1_n}\|$	By hypothesis
	$M \equiv c\ \overline{M^1_n}$ and $\mathcal{D}^2_i :: \Psi \vdash M_i \in \|Q_i\|$ for all $1 \leq i \leq n$	By sub-derivation
	$\Psi \vdash c\ \overline{M^1_n} \in \|E\ \Phi\|$	By hypothesis
	$c\ \overline{M^1_n} \downarrow t\ \Phi$, where $t \equiv \lambda\Phi\, .\, c\ \overline{t^1_n}$	By sub-derivation
	$\Gamma; \Omega; \Delta \vdash c\ \overline{t^1_n} : B$	For $\Gamma; \Omega; \Delta \vdash \Phi$ *ok*
	$(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash t_i : A_i$	For some Δ^u_i, Δ^u_i satisfying 1 and 2, by inversion on rule $\stackrel{1}{\to} E^n_c$ (Corollary 4.29)
	$\mathcal{D}^1_i :: \Psi \vdash M_i \in \|E_i\ \Phi_i\|$	By rule `grFlx` choosing Φ_i such that $(\Gamma, \Delta^u_i); \Omega; \Delta^1_i \vdash \Phi_i$ *ok*
	$\mathcal{D}_i :: \Psi \vdash E_i\ \Phi_i \cap Q_i \Rightarrow N_i$ for $1 \leq i \leq n$ and	
	$\Psi \vdash M_i \in \|N_i\|$	By IH on $\mathcal{D}^1_i, \mathcal{D}^2_i$ since the proviso is satisfied
	$\mathcal{D} :: \Psi \vdash E\ \Phi \cap c\ \overline{Q^1_n} \Rightarrow c\ \overline{N^1_n}$	By rule $\cap FR^c$
	$\Psi \vdash c\ \overline{M^1_n} \in \|c\ \overline{N^1_n}\|$	By rule `grApp`
Proj	Proceed as above, but using inversion on rule $\stackrel{1}{\to} E^n_y$, i.e. Corollary 4.32.	

Case: $\mathcal{D}_2$ ends in `grFlx` and $\mathcal{D}_1$ ends in `grApp`: symmetrical to the above.

Case: $\mathcal{D}_1, \mathcal{D}_2$ end in `grLam`:

$\Psi \vdash \lambda x^u\!:\!A.\, M \in \|\lambda x^u\!:\!A.\, N_1\|$	By hypothesis
$\mathcal{D}'_1 :: \Psi, x{:}A \vdash M \in \|N_1\|$	By sub-derivation
$\Psi \vdash \lambda x^u\!:\!A.\, M \in \|\lambda x^u\!:\!A.\, N_2\|$	By hypothesis
$\mathcal{D}'_2 :: \Psi, x{:}A \vdash M \in \|N_2\|$	By sub-derivation
$\mathcal{D}' :: \Psi, x{:}A \vdash N_1 \cap N_2 \Rightarrow N$ and $\Psi, x{:}A \vdash M \in \|N\|$	By IH on $\mathcal{D}'_1, \mathcal{D}'_2$
$\Psi \vdash \lambda x^u\!:\!A.\, N_1 \cap \lambda x^u\!:\!A.\, N_2 \Rightarrow \lambda x^u\!:\!A.\, N$	By rule
$\Psi \vdash \lambda x^u\!:\!A.\, M \in \|\lambda x^u\!:\!A.\, N\|$	By rule

Case: $\mathcal{D}_1, \mathcal{D}_2$ end in `grApp`: a straightforward appeal to the inductive hypothesis as in the above case.

□

For the other direction, we are going to prove a stronger result:

Theorem 4.34 *For any simple linear term N_1 and N_2 without shared variables such that $\Psi \vdash N_1 : A$ and $\Psi \vdash N_2 : A$, for every N such that $\Psi \vdash N_1 \cap N_2 \Rightarrow N$ if $\Psi \vdash M \in \|N\| : A$, then $\Psi \vdash M \in \|N_1\| : A$ and $\Psi \vdash M \in \|N_2\| : A$.*

Proof: By induction on the structure of $\mathcal{D} :: \Psi \vdash N_1 \cap N_2 \Rightarrow N$ and inversion on $\mathcal{D}' :: \Psi \vdash M \in \|N\| : A$.

Case: $\mathcal{D}$ ends in $\cap FF$; by Remark 4.22 and the right-to-left direction of Lemma 4.26.

Case: $\mathcal{D}$ ends in $\cap FR$.

$\mathcal{D} :: \Psi \vdash E\ \Phi \cap c\ \overline{Q^1_n} \Rightarrow c\ \overline{N^1_n}$ By hypothesis
$\mathcal{D}_i :: \Psi \vdash E\ \Phi_i \cap Q_i \Rightarrow N_i, 1 \leq i \leq n$ By sub-derivation
$\Psi \vdash c\ \overline{M^1_n} \in \|c\ \overline{N^1_n}\|$ By hypothesis
$\Psi \vdash M_i \in \|N_i\|$ By inversion
$\Psi \vdash M_i \in \|Q_i\|$ and $\Psi \vdash M_i \in \|E_i\ \Phi_i\|$ By IH on $\mathcal{D}_i$
$M_i \downarrow t\ \Phi_i$, where $t \equiv \lambda\Phi_i\,.\,t_i$ such that $(\Psi_i, D^u_i); \Omega; \Delta^1_i \vdash t_i : A_i$ By rule `grFlx` for $(\Psi_i, \Delta^u_i); \Omega; \Delta^1_i \vdash \Phi_i$ *ok*
$\Gamma; \Omega; \Delta \vdash c\ \overline{t^1_n} : a$ By rule $\xrightarrow{1} E^n_c$ (Lemma 4.27), since the proviso satisfies 1, 2
$\Psi \vdash c\ \overline{M^1_n} \in \|E\ \Phi\|$ By rule `grFlx`
$\Psi \vdash c\ \overline{M^1_n} \in \|c\ \overline{Q^1_n}\|$ By rule `grApp`

Case: $\mathcal{D}$ ends in $\cap FR^y$. Proceed as above, but using Lemma 4.30.

Case: $\mathcal{D}$ ends in $\cap L$; by IH as in Theorem 4.33

Case: $\mathcal{D}$ ends in $\cap RR$; ditto.

□

Corollary 4.35 (Adequacy of Pattern Intersection) *Fix a signature Σ. For every simple (linear) term N_1 and N_2 without shared variables such that $\Gamma \vdash N_1 : A$ and $\Gamma \vdash N_2 : A$, for every M, $\Gamma \vdash M \in \|N_1\| : A$ and $\Gamma \vdash M \in \|N_2\| : A$ iff $\Gamma \vdash M \in \|N_1 \cap N_2\| : A$.*

Proof: From Theorem 4.33 and 4.34. □

4.4 The Algebra of Strict Terms

An interesting and natural question is wondering whether complementation is involutive. The answer is of course positive, since the latter is a boolean property and the complement operation has been shown to satisfy "tertium non datur" and the principle of non-contradiction. Rather than proving involution in isolation, we will show that every other boolean property is satisfied. As the complement of a term is possibly a finite set of terms we need to extend the intersection and complement operations to *finite* sets of terms. For the sake of readability, we shall define this the empty context. It is clear, although cumbersome, how to generalize it. We also drop the type information and overload the singleton terms notation.

Definition 4.36 *If $\mathcal{M}$ and $\mathcal{N}$ are finite sets of (linear) simple terms of type A, define:*

$$\mathcal{M} \cap \mathcal{N} \stackrel{def}{=} \{Q \mid Q \in M \cap N, M \in \mathcal{M}, N \in \mathcal{N}\}$$

$$\mathrm{Not}(\mathcal{M}) \stackrel{def}{=} \bigcap_{M \in \mathcal{M}} \mathrm{Not}(M)$$

Those operations on set of terms satisfy the same properties that 'singleton' intersection and complementation do.

Corollary 4.37 (Adequacy of Set Intersection) *If $\mathcal{N}_1, \mathcal{N}_2$ are finite sets of (linear) simple terms of type A, then $\Gamma \vdash M \in \|\mathcal{N}_1\| : A$ and $\Gamma \vdash M \in \|\mathcal{N}_2\| : A$ iff $\Gamma \vdash M \in \|\mathcal{N}_1 \cap \mathcal{N}_2\| : A$.*

Proof: $\Gamma \vdash M \in \|\mathcal{N}_1\| : A$ and $\Gamma \vdash M \in \|\mathcal{N}_2\| : A$ iff there is $N_1 \in \mathcal{N}_1$ and $N_2 \in \mathcal{N}_2$ such that $\Gamma \vdash M \in \|N_1\| : A$ and $\Gamma \vdash M \in \|N_2\| : A$ iff, by Corollary 4.35, $\Gamma \vdash M \in \|N_1 \cap N_2\| : A$ iff, by definition, $\Gamma \vdash M \in \|\mathcal{N}_1 \cap \mathcal{N}_2\| : A$. □

Corollary 4.38 (Set Partition Lemma) *Let $\mathcal{N}$ be a finite set of (linear) simple terms of type A:*

1. *(Disjointness) It is not the case that $\Gamma \vdash M \in \|\mathcal{N}\| : A$ and $\Gamma \vdash M \in \|\mathrm{Not}(\mathcal{N})\| : A$.*

2. *(Exhaustivity)* $\Gamma \vdash M \in \|\mathcal{N}\| : A$ *or* $\Gamma \vdash M \in \|\mathrm{Not}(\mathcal{N})\| : A$.

Proof:

1. Assume $\Gamma \vdash M \in \|\mathcal{N}\| : A$ and $\Gamma \vdash M \in \|\mathrm{Not}(\mathcal{N})\| : A$. By rule $\mathtt{gr}_i$, $\Gamma \vdash M \in \|N\| : A$, for some $N \in \mathcal{N}$. By definition, $\Gamma \vdash M \in \| \cap_{N \in \mathcal{N}} \mathrm{Not}(N)\| : A$; by (repeated application of) Corollary 4.35 $\Gamma \vdash M \in \|\mathrm{Not}(N)\| : A$, for every $N \in \mathcal{N}$, impossible by the Partition Lemma.

2. Similarly to the above.

□

It is therefore possible to organize the set of *finite* sets of simple terms over a given signature, call it $\mathcal{T}_F$ in a boolean algebra under set union, patterns intersection and complementation, by taking equality as extensional identity (on sets of ground terms), that is, in symbols:

$$\mathcal{N}_1 \simeq \mathcal{N}_2 \quad \text{iff} \quad \|\mathcal{N}_1\| = \|\mathcal{N}_2\|$$

Theorem 4.39 *Consider the algebra of finite sets of simple terms* $\langle \mathcal{T}_F, \emptyset, \cup, \cap, \mathrm{Not} \rangle$ *under set union, pattern intersection and complementation. Then the following holds:*

1. $\mathcal{M} \cap \mathcal{M} \simeq \mathcal{M}$.
2. $\mathcal{M} \cap \mathcal{N} \simeq \mathcal{N} \cap \mathcal{M}$.
3. $\mathcal{M} \cap (\mathcal{N} \cup \mathcal{P}) \simeq (\mathcal{M} \cap \mathcal{N}) \cup (\mathcal{M} \cap \mathcal{P})$.
4. $\mathcal{M} \cap (\mathcal{N} \cap \mathcal{P}) \simeq (\mathcal{M} \cap \mathcal{N}) \cap \mathcal{P}$.
5. $\mathrm{Not}(\mathrm{Not}(\mathcal{M})) \simeq \mathcal{M}$.
6. $\mathrm{Not}(\mathcal{T}_F) \simeq \emptyset$.
7. $\mathrm{Not}(\emptyset) \simeq \mathcal{T}_F$.

Proof: From Corollary 4.37 and 4.38 and the fact that $\cup$ is set-theoretic. □

Corollary 4.40 *The algebra* $\langle \mathcal{T}_F, \emptyset, \cup, \cap, \mathrm{Not} \rangle$ *of finite sets of simple (linear) terms is boolean.*

Proof: Theorem 4.39 confirms that the above operators satisfy the boolean algebra axioms. □

Corollary 4.40 guarantees that any other boolean operation is definable: indeed complementation and intersection alone allows to define the *relative* complement operation:

Definition 4.41 *Given* $\mathcal{M}$ *and* $\mathcal{N}$ *sets of simple terms of type* A*:*

$$\mathcal{M} - \mathcal{N} \quad \stackrel{def}{=} \quad \mathcal{M} \cap (\mathrm{Not}(\mathcal{N}))$$

The adequacy of this encoding follows immediately from the Partition Lemma and soundness and completeness of intersection.

Corollary 4.42 $\Gamma \vdash M \in \|\mathcal{M}\| - \|\mathcal{N}\|$ *iff* $\Gamma \vdash M \in \|\mathcal{M} - \mathcal{N}\|$.

Proof: $\Gamma \vdash M \in \|\mathcal{M}\| - \|\mathcal{N}\|$ iff $\Gamma \vdash M \in \|\mathcal{M}\|$ and $\Gamma \vdash M \notin \|N\|$ for every $N \in \mathcal{N}$ iff (Corollary 4.38) $\Gamma \vdash M \in \|\mathcal{M}\|$ and $\Gamma \vdash M \in \|\mathrm{Not}(N)\|$ iff (Corollary 4.35) $\Gamma \vdash M \in \|\mathcal{M} \cap (\mathrm{Not}(\mathcal{N}))\|$ iff by definition $\Gamma \vdash M \in \|\mathcal{M} - \mathcal{N}\|$. □

It is notable that the $\cup$ operator must be set-theoretic union rather than anti-unification or generalization, as traditional in lattice-theoretic investigations of the algebra of terms [Plo71]. The problem is the intrinsic classical nature of complementation which is not compatible with the very irregular structure of the lattice of terms where anti-unification is interpreted as the lowest upper bound. Indeed, De Morgan's rules would fail, namely, denoting anti-unification with '$\vee'$':

$$\mathrm{Not}(s(0) \vee s(s(0))) = \mathrm{Not}(s(X)) = 0 \neq \{0, s(s(s(X)))\} = \mathrm{Not}(s(0)) \cap \mathrm{Not}(s(s(0)))$$

We end this chapter with a preview of how term complement will be used as a building block of the clause complement algorithm.

Example 4.43 *We can combine Example 2.6 and 2.8:*

$$\begin{array}{l}
\mathrm{Not}\{(app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F), lam(\lambda x^u\!:\!exp.\, app\ (E\ x^o)\ x)\} = \\
\quad \mathrm{Not}(app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F) \cap \\
\quad \mathrm{Not}(lam(\lambda x^u\!:\!exp.\, app\ (E\ x^0)\ x)) = \\
\\
\qquad \{lam\ (\lambda x^u\!:\!exp.\, (H\ x^u)), \\
\qquad app\ (app\ H\ H')\ H''\} \\
\qquad \cap \\
\\
\qquad \{lam(\lambda x^u\!:\!exp.\, app\ (H\ x^1)\ (H'\ x^u)), \\
\qquad lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H'' x^u))), \\
\qquad lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (lam(\lambda y^u\!:\!exp.\, H'\ x^u\ y^u))), \\
\qquad lam(\lambda x^u\!:\!exp.\, lam(\lambda y^u\!:\!exp.\, H\ x^u\ y^u)), \\
\qquad lam(\lambda x^u\!:\!exp.\, x), \\
\qquad app\ H\ H'\} = \\
\\
\qquad\quad \{lam(\lambda x^u\!:\!exp.\, app\ (H\ x^1)\ (H'\ x^u)), \\
\qquad\quad lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H''\ x^u))), \\
\qquad\quad lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (lam(\lambda y^u\!:\!exp.\, H'\ x^u\ y^u))) \\
\qquad\quad lam(\lambda x^u\!:\!exp.\, lam(\lambda y^u\!:\!exp.\, H'\ x^u\ y^u)), \\
\qquad\quad lam(\lambda x^u\!:\!exp.\, x), \\
\qquad\quad app\ (app\ H\ H')\ H''\}
\end{array}$$

Thus given the 'program':

$$\begin{array}{lcl}
betarx & : & isredx(app\ (lam\ (\lambda x^u\!:\!exp.\, E\ x^u))\ F). \\
etarx & : & isredx(lam(\lambda x^u\!:\!exp.\, app\ (E\ x^0)\ x)).
\end{array}$$

Computing the complement of each head as in Example 4.43 yields the complementary program:

$$\begin{array}{lcl}
nb1 & : & non_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ x^1)\ (H'\ x^u))). \\
nb2 & : & non_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (app\ (H'\ x^u)\ (H''\ x^u)))). \\
nb3 & : & non_isredx(lam(\lambda x^u\!:\!exp.\, app\ (H\ x^u)\ (lam(\lambda y^u\!:\!exp.\, H'\ x^u\ y^u)))). \\
nb4 & : & non_isredx(lam(\lambda x^u\!:\!exp.\, lam(\lambda y^u\!:\!exp.\, H\ x^u\ y^u))). \\
nbr & : & non_isredx(lam(\lambda x^u\!:\!exp.\, x)). \\
nb6 & : & non_isredx(app\ (app\ H\ H')\ H''\ ).
\end{array}$$

4.5 Summary

In this chapter we have been concerned with the relative complement problem in a setting where patterns may contain binding operators, so-called *higher-order patterns*. Higher-order patterns inherit many pleasant properties from the first-order case, even for complex type theories. Unfortunately, the complement operation does not generalize as smoothly. The complement of a partially applied higher-order pattern cannot be described by a pattern, or even a by finite set of patterns. The formulation of the problem suggests that we should consider a λ-calculus with an internal notion of *strictness* so that we can directly express that a term must depend on a given variable. For reasons of symmetry and elegance we have also added the dual concept of *invariance* expressing that a given term does not depend on a given variable. We have developed such a calculus, so that we can show that for a suitable embedding in our calculus simply-typed patterns is such that the complement of a linear pattern is a finite set of linear patterns and unification of two patterns is decidable and leads to a finite set of most general unifiers. Consequently, finite sets of linear patterns in the strict λ-calculus are closed under complement and unification. If we think of finite sets of linear patterns as representing the set of all their ground instances, then they form a boolean algebra under simple union, intersection (implemented via unification) and the complement operation.

Chapter 5

Elimination of Negation in Clauses

The transformational approach to negation in normal programs has a somewhat long history, see [Nai86] for a survey of the early 80's. The idea was to implement negation using inequalities, so that the complement of any predicate occurring negatively in a program is synthesized to obtain an equivalent definite program. This was first proposed in [ST84].

5.1 The Completion

At the risk of being trivial, let us start by asking naively what the complement of a program should be; if we see the latter as a set of (possibly mutually recursive) predicate definitions, its negation would be the set of the negation of those definitions. Thus, let us concentrate on a program definition as our target and consider the simplest case, i.e. that of a single clause $q(0)$ on the signature of numerals. Our first instinct would be to use the Not algorithm and by computing $\mathrm{Not}(0) = s(X)$ assert $\forall X\!:\!nat.\, \neg q(s(X))$; this is indeed the right thing to do, but we need to justify it formally. We can look at the definition $q(0)$ as a degenerate case (that is with trivial condition) of *inductive* definition; an equivalent formulation would be:

$$\forall X\!:\!nat.\, q(X) \leftarrow X \doteq 0.$$

for an object-logic equality symbol '$\doteq$', which simply expresses the condition $X \doteq 0$ for atoms to be in the inductive definition of q. In this Chapter we use just '$=$'. The next step is to enforce the minimality condition by saying that the latter is the only way to belong to the definition. One way to achieve that is by exchanging the $\leftarrow$ connective into a biconditional $\leftrightarrow$:

$$\forall X\!:\!nat.\, q(X) \leftrightarrow X = 0.$$

This is in a nutshell Clark's very fortunate idea of the *completion* of a program [Cla78]. What is left is describing how to interpret the equality relation; this is accomplished by the so-called Clark's equality theory, that is the axioms of free equality: namely, the usual equality axioms including congruence, plus the axioms for finite trees. Indeed, this theory is the axiomatic and proof-theoretic rendering of the unification algorithm, for a proof see for example Stärk's thesis [Stä92]. For instance, the following is free equality over numerals:

$$\begin{array}{rcll}
(Dec) & : & \forall x, y : s(x) = s(y) \rightarrow x = y & \\
(Clh) & : & \forall x : 0 \neq s(x) & \\
(Ock) & : & \forall x : x \neq t[x] & \text{if } x \text{ occurs properly in } t[x] \\
(DCA) & : & \forall x : x = 0 \vee \exists y : x = s(y) &
\end{array}$$

The last axiom is the Domain Closure Axiom [MMP88], which is required to give a complete axiomatization of finite trees over *finite* signatures. Since we will not consider unification so far, we will keep this relation uninterpreted; thus those axioms do not play any role, which is handy, as it allows us to dispense with the issue of the compatibility of DCA with dynamic extensions on the signature.

$$
\begin{array}{l}
\forall x \,.\, even(x) \leftrightarrow x = 0 \lor \exists y \,.\, x = s(s(y)) \land even(y) \\
\quad \leadsto_{cp} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow \neg(x = 0 \lor \exists y \,.\, x = s(s(y)) \land even(y)) \\
\quad \leadsto_{nnf} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow x \neq 0 \land (\forall y \,.\, x \neq s(s(y)) \lor \neg even(y)) \\
\quad \leadsto_{lift} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow x \neq 0 \land \forall y \,.\, x \neq s(s(y)) \lor (\exists y \,.\, x = s(s(y)) \land \neg even(y)) \\
\quad \leadsto_{dnf} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow (x \neq 0 \land \forall y \,.\, x \neq s(s(y))) \lor (x \neq 0 \land \exists y \,.\, x = s(s(y)) \land \neg even(y)) \\
\quad \leadsto_{disunify(x \neq 0 \land \forall y : x \neq s(s(y)))} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow x = s(0) \lor (x \neq 0 \land \exists y \,.\, x = s(s(y)) \land \neg even(y)) \\
\quad \leadsto_{disunify(x \neq 0 \land \exists y : x = s(s(y)))} \\
\quad\quad \forall x \,.\, \neg even(x) \leftrightarrow x = s(0) \lor (\exists y \,.\, x = s(s(y)) \land \neg even(y)) \\
\quad \leadsto_{prettyp}
\end{array}
$$

$$
\begin{array}{l}
odd(s(0)). \\
odd(s(s(Y))) \leftarrow odd(Y).
\end{array}
$$

Figure 5.1: Synthesis of the predicate `odd`

Of course, definitions are usually more interesting than a simple clause, so let us step to next simplest example, even numbers:

$$
\begin{array}{l}
even(0). \\
even(s(s(Y))) \leftarrow even(Y).
\end{array}
$$

To turn this code into a minimal inductive definition, we need to normalize the conjunction of clauses, building what is known in the logic programming jargon as the *completed* definition of a predicate, a process described for example in [AB94]. The net result is the axiom:

$$\forall x.even(x) \leftrightarrow x = 0 \lor \exists y.x = ss(y) \land even(y)$$

One way to obtain the complement definition, that is `odd`, would be to reason classically on the completed program by taking the contrapositive of the completion. Let me offer the following rational reconstruction. We may use rewrite rules to achieve conversion into negation normal form (nnf) and into disjunctive normal form (dnf), plus some more massage to preserve the original positive bindings in clauses. Once this is done, we need a way to solve the possibly universally quantified dis-equalities we have created. A call of the disunification algorithm described in Section 2.2 (`disunify(...)`) is enough to obtain a solved form, from which we can recover the intended negated program. This is best explained in Figure 5.1, which uses the subcomputation of $\forall y : z \neq 0 \land z \neq s(s(y))$, detailed in Example 2.2.

Following this drift, an extensive project started in Pisa under the name of *intensional negation* [BMPT87, MMP88, BMPT90, ABT90, MPRT90b, FBM93]. In particular [BMPT90] computes the set-theoretic complement of the terms in the negative predicate compiling away the inequalities. The authors restrict to a class of left-linear non-stratified program called *flat*, where all predicates are defined by a single clause (and hence we have no disjunction in the completion) and such that if a head contains non-variable terms, the body must be a single literal. With some painful source-to-source transformations, programs can be turned in and out of this format. Thanks to this, the transformation of the completion delineated above yields only disequation of the form $x \neq t$, which the Not algorithm can solve.

If we discard for the moment the problematic issue of local variables, i.e. variables that appear in the body but not in the head of a clause (as they turn out to become universally quantified in an extensional sense during the completion transformation), this seems at first sight fairly convincing. On the other hand, managing control of disunification and rewrite rules requires some ingenuity as the following example shows:

Example 5.1 *Consider the usual program for membership in lists:*

$$\begin{array}{l} member(X, X.XS). \\ member(X, Y.YS) \leftarrow member(X, YS). \end{array}$$

Its completion is

$$\begin{array}{lcl} \forall z, zs(member(z, zs) & \leftrightarrow & \exists x, xs(zs = x.xs \wedge z = x) \vee \\ & & \exists x, y, ys(zs = y.ys \wedge z = x \wedge member(x, ys))) \end{array}$$

The synthesis of its negation is depicted in Figure 5.2. Not only disunification is more complex, as there are a few choices of variables where the Explosion rule can be applied, but propositional transformations are difficult to direct as well.

Working with flat programs is not a real alternative, since some form of partial evaluation is needed to recover some structure in the target program. In fact, the more mature version presented in [FBM93] embraces the constraint logic programming approach. We will instead give a completely deterministic algorithm to compute the negation of programs. This is based on solving the relative complement problem by pairing term complement with unification and is proven correct by Corollary 4.40; that is, we do not need full disunification as we can solve, for example $x \neq 0 \wedge \forall y : x \neq s(s(y))$ by computing $\mathrm{Not}(0) \cap \mathrm{Not}(s(s(Y))$.

There is one further, more basic difficulty with the completion-based approaches; they use transformations that are intrinsically classical and turn out to preserve the operational semantics, only because in Horn logic classical and intuitionistic provability coincide. We discuss this issue further in Section 5.3. We will instead lift the boolean operations we have introduced on (simple) terms to clauses (programs) and we shall prove that they still satisfies the usual boolean rules; in particular we will verify that clause complementation fulfills exclusivity and exhaustivity. This high-level 'boolean' language will eventually be compiled into a version of HHF that is amenable of a complete uniform proof search strategy. We offer an informal discussion next (Section 5.2).

5.2 Introduction to HHF Complementation

Consider the following judgment to check whether a lambda terms is *linear*[1], if every functional sub-term uses each of its arguments exactly once:

$$\dfrac{\Lambda x\,.\,e \text{ linear in } x \qquad \begin{array}{c} \overline{x \text{ linear}}\,u \\ \vdots \\ e \text{ linear} \end{array}}{\Lambda x\,.\,e \text{ linear}}\,linlam^{x,u}$$

$$\dfrac{e_1 \text{ linear} \qquad e_2 \text{ linear}}{(e_1\ e_2) \text{ linear}}\,linapp$$

$$\dfrac{}{\Lambda x\,.\,x \text{ linear in } x}\,linxx \qquad \dfrac{\Lambda x\,.\,e \text{ linear in } x}{\Lambda x\,.\,\Lambda y\,.\,e \text{ linear in } x}\,linxlm^{y}$$

$$\dfrac{\Lambda x\,.\,e_1 \text{ linear in } x}{\Lambda x\,.\,(e_1\ e_2) \text{ linear in } x}\,linxapp_1 \qquad \dfrac{\Lambda x\,.\,e_2 \text{ linear in } x}{\Lambda x\,.\,(e_1\ e_2) \text{ linear in } x}\,linxapp_2$$

[1]Do not confuse this notion with the one which refers to a term not having repeated occurrences of the same existential variable.

$$
\begin{array}{ll}
comp(member) & \\
\rightsquigarrow_{cp} & \\
\quad \forall z, zs(\neg member(z, zs) \leftrightarrow & \forall x, xs(zs \neq x.xs \vee z \neq x) \wedge \\
& \forall x, y, ys(zs \neq y.ys \vee z \neq x \vee \neg member(x, ys)) \\
\rightsquigarrow_{lift} & \\
\quad \forall z, zs(\neg member(z, zs) \leftrightarrow & \forall x, xs(zs \neq x.xs \vee z \neq x) \wedge \\
& \forall x, y, ys(zs \neq y.ys \vee z \neq x) \vee \\
& \exists x, y, ys\,.\,(zs = y.ys \vee z = x) \wedge \neg member(x, ys) \\
\rightsquigarrow_{dnf} & \\
\quad \forall z, zs(\neg member(z, zs) \leftrightarrow & (d1)\ \forall x, y, xs, ys\,.\,(zs \neq x.xs \vee z \neq x) \wedge (zs \neq y.ys \vee z \neq x) \\
& (d2)\ \forall x, xs, (zs \neq x.xs \vee z \neq x) \wedge \\
& \exists x, y, ys\,.\,(zs = y.ys \wedge z = x) \wedge \neg member(x, ys) \\
(d1)\ \longmapsto_{E+R(zs=nil)(1.1)} & \\
& (d1.1)\ \forall(nil \neq x.xs \vee z \neq x) \wedge (nil \neq y.ys \vee z \neq x) \wedge zs = nil \\
\longmapsto_{Cl(T)^{*}} & \\
& zs = nil \\
\rightsquigarrow_{prettyp} & \\
& \mathtt{nonmember(X, nil)} \\
(d1)\ \longmapsto_{E+R(zs=w.ws)(1.2)} & \\
& (d1.2)\ \exists w, ws \forall x, y, xs, ys.(w.ws \neq x.xs \vee z \neq x) \wedge \\
& (w.ws \neq y.ys \vee z \neq x) \wedge zs = w.ws \\
\longmapsto^{*}_{UE2(x,y,xs,ys)} & \\
& \boxed{\exists w(w \neq z)} \\
\rightsquigarrow_{prettyp} & \\
& \langle\rangle \\
(d2)\ \rightsquigarrow_{disunify(\forall x, xs(zs \neq x.xs \vee z \neq x))} & \\
\longmapsto^{*}_{UE2(x,y,xs,ys)} & \\
& x \neq y \wedge \exists y, ys\,.\,zs = y.ys \wedge z = x \\
\longmapsto_{norm} & \\
& \exists y, ys(zs = y.ys \wedge z \neq x) \\
\rightsquigarrow_{prettyp} & \\
& nonmember(X, Y.YS) \leftarrow X \neq Y, nonmember(X, YS).
\end{array}
$$

Figure 5.2: Synthesis of the `nonmember` predicate

Intuitively, we check for linearity of a function making sure that the it is linear in its first argument (judgment '$\Lambda x\,.\,e$ linear in x') and then recurring on the rest of the expression. Note the rule '$linlam\lambda^{x,u}$' is *hypothetical* in u and *parametric* in x; rule $linxlm^{y}$ is instead only parametric in x.

Frameworks based on HHF provide an ideal syntax to represent these judgments; namely, via the usual encoding introduced in Example 2.6:

Example 5.2

$$\begin{array}{rcl}
linlam & : & linear(lam\ \lambda x\,.\,E\ x) \\
& & \quad \leftarrow linx(\lambda x\,.\,E\ x) \\
& & \quad \leftarrow (\forall x\!:\!exp.\,linear(x) \rightarrow linear(E\ x)). \\
linapp & : & linear(app\ E_1\ E_2) \\
& & \quad \leftarrow linear(E_1) \\
& & \quad \leftarrow linear(E_2).
\end{array}$$

$$\begin{array}{rcl}
linxx & : & linx(\lambda x\,.\,x). \\
linxap1 & : & linx(\lambda x\,.\,app\ (E_1\ x)\ E_2) \\
& & \quad \leftarrow linx(\lambda x\,.\,E_1\ x). \\
linxap2 & : & linx(\lambda x\,.\,app\ E_1\ (E_2\ x)) \\
& & \quad \leftarrow linx(\lambda x\,.\,E_2\ x). \\
linxlm & : & linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\
& & \quad \leftarrow (\forall y\!:\!exp.\,linx(\lambda x\,.\,E\ x\ y)).
\end{array}$$

The judgment and its implementation is clearly a decision procedure. It does make sense to ask ourselves what is its complement. An expression is *not* linear if there is some function which either does not use its argument or uses it more than once. We first observe that, since `linear` is a relation defined via exhaustive and exclusive patterns term complementation does not play a role. Then, the complement of `linapp` does not pose any problem, as it is a Horn clause: an application in not linear if either the first element or the second in not linear.

$$\begin{array}{rcl}
\neg linapp & : & \neg linear(app\ E_1\ E_2) \\
& & \quad \leftarrow \neg linear(E_1) \vee \neg linear(E_2).
\end{array}$$

A lambda expression in not linear in two cases: first it is not linear in its first argument:

$$\begin{array}{rcl}
\neg linlam1 & : & \neg linear(lam(\lambda x\,.\,E\ x)) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E\ x).
\end{array}$$

Secondly, if its body is not linear. Now, this poses new problems, as we have to negate a hypothetical and parametric clause. Let us follow our nose and reason by example: suppose we are given, in the empty context a goal $linear(lam(\lambda x\,.\,lam(\lambda y\,.\,x)))$, which is unprovable, since the second lambda term is not linear in y; the proof tree yields the failure leaf $linx(\lambda y\,.\,z)$, for a new parameter z, in the context $z{:}exp; linear(z)$. Our guiding intuition is that we want to mimic a failure derivation so as to provide a successful derivation from the negative definition, i.e. a proof of $\neg linx(\lambda y\,.\,z)$ from $z{:}exp; linear(z)$; this shows one prominent feature of complementation of an HHF formula: negation 'skips' over $\forall$ and $\rightarrow$, since it needs to mirror failure *from* assumptions.

Let us turn to complementing the judgment ' linear in x'. A first point to note in that, by encoding an object expression 'e' with a pattern variable, we must make sure that in clause `linxap1,linxap2` the variable x *does* not occur in the argument which is not checked. We thus embed the clause in the strict λ-calculus and 'E' in the simple term '$E\ x^0$'. For the sake of readability we do this only for the two aforementioned clause and we also hide $()^u$ annotations.

$$\begin{array}{rcl} linxap1 & : & linx(\lambda x\,.\,app\ (F\ x)\ (G\ x^0)) \\ & & \quad \leftarrow linx(\lambda x\,.\,F\ x). \\ linxap2 & : & linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x)) \\ & & \quad \leftarrow linx(\lambda x\,.\,G\ x). \end{array}$$

Via term complementation and intersection in the strict λ-calculus we obtain, among others:

$$\begin{array}{rcl} \neg linxap1 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^1)). \\ \neg linxap2 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x^0)). \end{array}$$

Moreover, similarly the case of top-level application, the complement of $\texttt{linxap}_i$ holds if the body does not hold:

$$\begin{array}{rcl} \neg linxap1 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^0)) \\ & & \quad \leftarrow \neg linx(\lambda x\,.\,F\ x^1) \\ \neg linxap2 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x^1)) \\ & & \quad \leftarrow \neg linx(\lambda x\,.\,G\ x^1). \end{array}$$

Now, let us examine clause `linxlm` and let us reconsider the failure leaf $linx(\lambda y\,.\,z)$ from the context $z{:}exp; linear(z)$. In a first attempt, let us consider what the complement would be according to the idea above:

$$\begin{array}{rcl} \overset{?}{\neg}\, linxlm & : & \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ & & \quad \leftarrow (\forall y\,{:}\,exp.\ \neg linx(\lambda x\,.\,E\ x\ y)). \end{array}$$

However, there is no way to obtain a proof of $\neg linx(\lambda y\,.\,z)$ from the current context. Indeed, the `linxlm` clause does not carry enough information by itself so that its complement can mimic the failure proof. In a sense that we will make precise, the clause, and in turn its predicate definition is not *assumption-complete*: once it has introduced a new parameter, the clause only specifies how to use it in a positive context. It is up to us to synthesize its dynamic negative definition, in this case exactly $\neg linx(\lambda y\,.\,z)$. More generally, it is a characteristic of HHF that the negation of a clause is not enough to determine the behavior of a program under complementation. We will have to insert (via a source-to-source transformation) additional structure in a predicate definition in order to completely determine the provability and failure of goals which mention *parameters*. By observing the structure of all possible assumption that a predicate definition can make, we will *augment* those assumptions with their negative definition. In particular, we first augment the clause `linxlm`:

$$\begin{array}{rcl} aug_{\mathrm{D}}(linxlm) & : & linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ & & \quad \leftarrow (\forall y\,{:}\,exp.\ \neg linx(\lambda x\,.\,y) \rightarrow linx(\lambda x\,.\,E\ x\ y)). \end{array}$$

so that, by complementation, we obtain

$$\begin{array}{rcl} \neg aug_{\mathrm{D}}(linxlm) & : & \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ & & \quad \leftarrow (\forall y\,{:}\,exp.\ \neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)). \end{array}$$

Moreover, we need to do the same with the `linlam` clause, since the `linx` predicate may occur as a subgoal:

$$\begin{array}{rcl} aug_{\mathrm{D}}(linlam) & : & linear(lam(\lambda x\,.\,E\ x)) \\ & & \quad \leftarrow linx(\lambda x\,.\,E\ x) \\ & & \quad \leftarrow (\forall x\,{:}\,exp.\,(\neg linx(\lambda y\,.\,x) \wedge linear(x)) \rightarrow \neg linear(E\ x)). \end{array}$$

In summary the negative program is:

$$
\begin{array}{rcl}
\neg linapp & : & \neg linear(app\ E_1\ E_2) \\
& & \quad \leftarrow \neg linear(E_1) \vee \neg linear(E_2). \\
\neg linlam1 & : & \neg linear(lam(\lambda x\,.\,E\ x)) \\
& & \quad \leftarrow \neg linx(\lambda x\,.\,E\ x) \\
& & \quad \vee (\forall x\!:\!exp.\,(\neg linx(\lambda y\,.\,x) \wedge linear(x)) \rightarrow \neg linear(E\ x)).
\end{array}
$$

$$
\begin{array}{rcl}
\neg linxap1 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^1)). \\
\neg linxap2 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x^0)). \\
\neg linxap3 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^0)) \leftarrow \neg linx(\lambda x\,.\,F\ x^1). \\
\neg linxap4 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x^1)) \leftarrow \neg linx(\lambda x\,.\,G\ x^1). \\
\neg linxlm & : & \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\
& & \quad \leftarrow (\forall y\!:\!exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)).
\end{array}
$$

While it is not impossible[2] to manually come up with this program by writing predicate definitions formalizing when terms are *strict* (that is, variable arguments occur at least once) and *vacuous*(that is, arguments are guaranteed not to occur) and then by merging them in the correct fashion, it would be better to have this done automatically, especially considering changes or extensions of the original program.

Unfortunately the procedure we have outlined is not possible in general. Consider a clause encoding the introduction rule for implication in natural deduction, which can be used to check whether an implicational formula trivially holds:

Example 5.3

$$
\begin{array}{rcl}
form & : & type \\
imp & : & form \rightarrow form \rightarrow form \\
a & : & form \\
b & : & form \\
impi & : & nd(A\ imp\ B) \leftarrow (nd(A) \rightarrow nd(B)).
\end{array}
$$

Following our earlier remark its complement would be:

$$
\begin{array}{rcl}
\neg impi1 & : & \neg nd(a) \\
\neg impi2 & : & \neg nd(b) \\
\overset{?}{\neg}\ impi & : & \neg nd(A\ imp\ B) \leftarrow ((\forall C\!:\!form.\,\neg nd(C) \leftarrow C \not\doteq A) \rightarrow \neg nd(B)).
\end{array}
$$

Apparently, this specification is incorrect since both $nd(a\ imp\ a)$ and $\neg nd(a\ imp\ a)$ are derivable from the empty context. We can isolate one major problem: in clause `impi`, the assumption $nd(A)$ that is dynamically added to the (static) definition of the `nd` predicate *overlaps* with the head of the clause. Thus, a goal such as ¬`nd(a)` can be resolved with both the static *and* the dynamic program, yielding inconsistent solutions. A symmetrical problem can occur when dynamic and static clauses do differ but their complements do not. Suppose we introduce a predicate which checks if a number is even and non zero as follows:

$$e \quad : \quad ev(s(s(N))) \leftarrow (ev(0) \rightarrow ev(N)).$$

Again, our naive algorithm would incorrectly yield:

$$
\begin{array}{rcl}
\neg e1 & : & \neg ev(0). \\
\neg e2 & : & \neg ev(s(0)). \\
\overset{?}{\neg}\ e & : & \neg ev(s(s(N))) \leftarrow (\forall M\!:\!nat.\,\neg ev(s(M)) \rightarrow \neg ev(N)).
\end{array}
$$

[2]For what is worth, the first three versions of such a program I wrote were mistaken.

Thus both $ev(s(s(0)))$ and $\neg ev(s(s(0)))$ are incorrectly provable. The problem here is the overlapping between the of assumption $ev(0)$ and the *complement* of the head of the `e` clause.

We have thus isolated two main issues:

1. Exhaustivity: we need to enrich clauses so that every (ground) goal or its negation is provable.
2. Exclusivity: we need to isolate a significant fragment where it is not the case that both a goal and its negation are provable.

We will describe in Section 6.6 a procedure that we call *augmentation*, which, by enriching the program with the complement of assumptions will, will achieve exhaustivity (Section 6.8); moreover, we will achieve exclusivity with the restriction to *complementable* programs, formally introduced in Figure 6.7. To anticipate the idea, a clause is complementable if every assumption is *parametric* in some eigenvariable. We will try to motivate in Section 5.4 why this fragment is adequate to the practice of logical frameworks. Section 5.5 reviews some related work in the area of *NF* and embedded implication.

5.3 Background

Traditionally (and ideally), a completion construction for the *NF* rule is an extension of a program, say $E(P)$ such that, in a logic L equipped with a provability and finite failure relation, say $\vdash_L$ and $\dashv\vdash_L$, a consequence relation $\models_L$ and a negation sign $\neg$, for any given goal G (ideally) it holds:

1. $\vdash_L G$ iff $E(P) \models_L G$.
2. $\dashv\vdash_L G$ iff $E(P) \models_L \neg G$.

Many such constructions have been proposed for Horn logic, starting from the *Closed World Assumption* (CWA) [Rei78]. The Clark completion [Cla78] is perhaps the most successful proof-theoretic and finitary explanation of *NF*: the main idea is that clauses in a predicate definition should be seen as an *iff-definition*, thus enforcing the minimality condition of its inductive definition; this would correspond, in model-theoretic terms, to the existence of a least intended model. The if-part states the condition to belong to the inductive definition, while the only-if part excludes everything else, thus providing a computable approximation to the CWA. This is well understood and agreed as far as Horn logic is concerned and confirmed by the completeness of finite failure w.r.t. the completion [Apt90].

As observed first by Gabbay [Gab85], the *positive* logic of embedded implication is not classical but intuitionistic (actually minimal). When coupled with negation as failure in all its generality, its meta-logic fails to have some straightforward logic properties, as detailed in Section 5.5. The key difference lies in the constructive interpretation of implication and its delicate interplay with negation. While a completion construction is possible, it may be *not* equivalent to the adjoining of the only-if part of the program. In particular, it not warranted to form the negation of a program by taking the contrapositive of the completed definition; this is intrinsically due to the operational semantic of failure: a goal $D \to G$ fails iff from the (scoped) assumption D we have that G fails. Let us try to mirror this with a logical connective:

$$\neg(D \to G) \quad \overset{?}{\leftrightarrow} \quad D \to \neg G \tag{5.1}$$

Similarly for parametric judgments:

$$\neg(\forall x\!:\!A.\, G) \quad \overset{?}{\leftrightarrow} \quad \forall x\!:\!A.\, \neg G$$

No standard logic of negation satisfies the above rules. In particular, it is erroneous to formulate Clark's completion using Nelson's strong negation [Nel49], which is currently held as the meta-logic of negation elimination in the Horn setting [GL90, Pea90]. Indeed, strong negation brings too much duality to intuitionistic logic as it is pushed in through connectives and quantifiers; in particular if $\sim$ denotes strong negation, the following holds:

$$\begin{aligned} \sim(D \to G) \quad &\leftrightarrow \quad D \wedge \sim G \\ \sim(\forall x\!:\!A.\, G) \quad &\leftrightarrow \quad \exists x\!:\!A.\, \sim G \end{aligned}$$

Strong negation may be "the logic of information structures" [Wan93], as far as Horn logic is concerned, but it is definitely *not* the meta-logic of negation elimination in logical frameworks based on HHF. It is not simply a question to endow intuitionism with a semi-classical notion of negation, while preserving the disjunction and existential property. The hard point is not negation in itself, but its interaction with a more operational Brower-Heyting-Kolmogorov interpretation of implication.

Example 5.4 *Consider the program consisting only of the clause* $a \leftarrow (b \rightarrow c)$*; the standard completion would be*

$$a \leftrightarrow (b \rightarrow c) \wedge \neg b \wedge \neg c.$$

Now, 'a' fails and hence '$\neg a$' should follow from the iff-completion: still 'a' intuitionistically (yet not minimally) follows, while '$\neg a$' is logically independent exactly due to the failure of equivalence (5.1).

We explore in Section 5.5 how this issue has been investigated in the literature. This is relevant to our enterprise because:

- In the Horn setting, the iff-completion has been the preferred way to logically motivate the transformational approach to negation, as we have seen in Section 5.1.
- In [GO98] the authors persuasively argue that the unrestricted addition of *NF* to languages such as N-Prolog requires the switch to a (three-valued) modal logic.

Since we need to express the negation of a predicate in the same language where the predicate is formulated, we choose to restrict the set of programs we deem complementable in a novel and extensive way. This will help to close the gap between the two poles usually associated to classic and intuitionistic logic programming, i.e. the closed versus open world assumption. We will define a class of programs which extend the current database in a specific regular way, by ensuring that static and dynamic clauses never overlap. This property extends w.r.t. the complement program and thus has the side effect of guaranteeing the consistency of the completion. Finally we will require every goal to conform to such a *schema context*. We call this approach the *Regular World Assumption* (RWA). We argue next (Section 5.4) that this class of programs is just what the doctor ordered for logical frameworks.

5.4 Motivation

Embedded implication in intuitionistic logic programming has been successfully used in various areas of logic programming; we can roughly divide those into:

- Meta-programming, namely specifying and implementing (the meta-theory of) deductive systems.
- Hypothetical reasoning in databases [GR84, BMV89, Bon94] or simulating imperative-style programming [BM90].
- Incorporating modules and local predicate definitions [Mil89c, GMR92, KNW93] and state encapsulation in object-oriented programming [HM90].

We now try to motivate why the restriction of complementation to *parametric implication* (formally defined in Section 6.5) is pragmatically adequate for our intended application.

While implementing deductive systems embedded implication is usually coupled with higher-order abstract syntax to represent *scoping constructs*: the latter is typically used to traverse operators like abstractions, quantification and, more pervasively, to represent rules with hypothetical premises. In this application implications are typically 'covered' by an intensional universal quantifier which express the parametricity of the assumption. Dozens of examples can be found for instance in [Pfe92]. One counter-example I am aware of is when using a logic framework to encode derivability in an object logic: here meta-logic contexts are used to manage object logic hypotheses, as in Example 5.1. We feel that this case in not typical; first of all it is questionable (and out of the scope of a logical framework) to complement recursive enumerable predicates such as general provability. Even if we limit ourselves to decision procedures as propositional logic, we are in a sense asking for a self-referential use of implication as we wish to represent implications in the object logic with the same in the meta-logic. In this case, though we have no formal proof of this, we think we have to use ad hoc techniques such as explicit management of hypotheses say as lists; a similar approach is taken in [DM97].

Modules are usually *closed* assumptions and I doubt negation is useful at this level. Local definitions, as an auxiliary `reverse3` procedure hidden inside the naive `reverse` procedure are typical only in higher-order logic programming languages as λProlog, and require full predicate quantification. We conjecture that some of those programs can be complemented if there is no overlap between static and dynamic clauses. As far as object-oriented programming is concerned, more recent research [BDLM00] has now moved on to linear logic programming languages.

The possibility to simulate the availability of global variables in logic programming has been advocated [GO98] as the main motivation of the intertwined and unrestricted use of negation and embedded implication. We contend that sometimes that can be resolved with a refined notion of context as the one available in linear logic programming: this has also the side-effect of by-passing the issue of non-stratification.

We now exemplify how to eliminate negation from non-stratified N-Prolog programs. We cannot say at the moment whether this transformation can be generalized and eventually mechanized. The following example (taken from [GO98]) is a non-stratified program to compute the parity of a relation encoded as n entries of the form $r(X)$, where '\+' denotes negation-as-failure:

Example 5.5 (Parity)

$$\begin{array}{l} even \leftarrow \backslash\!+ odd. \\ odd \leftarrow r(X), \backslash\!+ mark(X), (mark(X) \rightarrow even). \end{array}$$

We can write a Lolli *program [HM94] based on the same algorithm, where the linear context contains the entries of the relation and the initial token* `off`*; let* $\multimap$ *and* $\otimes$ *denote linear implication and conjunction:*

$$\begin{array}{l} even \circ\!- r(X) \otimes off \otimes (on \multimap odd). \\ even \circ\!- off. \\ odd \circ\!- r(X) \otimes on \otimes (off \multimap even). \\ odd \circ\!- on. \end{array}$$

The tensors consume the relation and turn on or off the switch accordingly to the parity of the relation.

We can make a similar remark about the relative pronoun gap parsing example in Categorical Grammars [PM90] which in [BM90] is treated with intuitionistic implication plus *NF*. Hodas [Hod94] has shown how this can be dealt with much more elegantly again with linear implication.

5.5 Related Work

In the 90's there has been some interest in combining *NF* with what is known as intuitionistic logic programming. The underlying languages, with the exception of [Har93], are either versions of N-Prolog or clausal intuitionistic logic. Due to the inherently difficulty with universal quantification mentioned in Subsection 1.5.2, the treatment is deprived of parametric goals. The emphasis is not on the transformational approach but to combine the non-monotonic nature of *NF* with the capability of embedded implication to dynamically update the current program. We will not detail here the proposed semantics (stable models [Dun92], monotonic Kripke-like models [Har89], non-monotonic perfect [BM90] or three-valued [GO98] models).

Gabbay [Gab85] pointed out that *NF* coupled with embedded implication seems to lead to curious paradoxes, at the point of making the whole enterprise not logically sound. These problems arise already when dealing with (propositional) normal clauses. They can be summarized as follows (taken from [NL92]).

1. Failure of transitivity, or, in other words, non-eliminability of the cut rule. Let $\mathcal{P}$ be:

$$\begin{array}{l} a \leftarrow b \wedge \backslash\!+ c. \\ b \leftarrow c. \end{array}$$

   Now, $\mathcal{P} \vdash c \rightarrow b$ and $\mathcal{P} \vdash b \rightarrow a$, as c is undefined, but $c \rightarrow a$ is not provable from $\mathcal{P}$.

2. Failure of weakening. Let $\mathcal{P}$ be:

$$b \leftarrow \backslash + a.$$

   Then $\mathcal{P} \vdash b$ but $a \rightarrow b$ is not provable from $\mathcal{P}$.

3. "Pathology of negative information", i.e. $\mathcal{P} \vdash \backslash + (a \rightarrow b)$ iff $\mathcal{P} \vdash a \rightarrow (\backslash + b)$.

There are several solutions to this riddle:

- A modal completion [GO98].
- A syntactic distinction between implications [BM90].
- A syntactic distinction between predicates [Har93].

We detail the latter next, but let us state, for the record, our position:

1. Provability must take into account the context where the query is attempted: in case 1. the first query is not allowed, since b is a predicate which must be called from an empty context.
2. Similarly, $a \rightarrow b$ is not a legitimate query as b must be queried in the empty context.
3. This is indeed the operational semantics of failure of hypothetical judgments; the issue is giving a logical justification of that and the challenge is to give it without changing the logic underneath.

5.5.1 *NF* in Clausal Intuitionistic Logic

We can regard Clausal Intuitionistic Logic as a close cousin of uniform proofs independently developed by McCarthy in the late eighties [McC88a, McC88b]. *NF* in this setting has been investigated in Bonner's thesis [Bon91] and is summarized in [Bon94]. The framework in [BM90] is hypothetical Datalog (with possibly infinite constants) and embedded implications. They assume a notion of negation-stratification and develop an awkward proof-theory parameterized by strata; a non-monotonic preferred Kripke model theory is presented and adequacy is demonstrated. They offer the following solution to the aforementioned paradoxes: the implication sign is really two distinct connectives, one for clauses and one for goals. Clause implication is interpreted classically and it is transitive, while goal implication is not and must be interpreted non-monotonically. Indeed the latter has a *modal* semantic which takes into account the extension of a context as a shift in worlds (in the Kripke sense). While we favor a syntactic distinction between goals and clauses, the uniform proof approach views implication as logical (intuitionistic) implication whose different behavior is simply dictated by its introduction and elimination rule. Moreover we do not aim to deal with arbitrary extensions. We have hinted in the previous section how linear logic programming can address examples in this paper which are outside the fragment we are able to treat.

5.5.2 *NF* and N-Prolog

This approach has been investigated first at the propositional level in [NL92] and then for first-order N-Prolog in [GO98]:

> "In order to understand N-Prolog computations with *NF*, we must adopt a dynamic view of success and failure [...] we cannot determine what goals succeed or fail from a program unless we determine what goal succeeds or fail from *arbitrary*[3] extension of the program" [NL92] pp. 258.

The main idea is again that a computation of an implicational goal $D \rightarrow G$ from $\mathcal{P}$ entails a shift to another world where $\mathcal{P} \cup D$ holds. The authors propose a completion construction as an explanation of negation-as-failure. The completion is formulated as an infinite theory such that $comp_n(\mathcal{P})$ is used to evaluate a goal G roughly if we need n extensions of the program to compute G: for example for a clause

[3] Emphasis is mine.

$a \leftarrow (b \rightarrow c)$ then a is provable iff $comp(\mathcal{P} \cup b)$ entails c and a is false iff $comp(\mathcal{P} \cup b)$ entails $\neg c$. Note that $comp(\mathcal{P} \cup b)$ is different from $comp(\mathcal{P}) \cup b$ or $comp(\mathcal{P}) \cup comp(b)$. Moreover, due to the possible non-monotonicity of arbitrary extensions $comp(\mathcal{P})$ and $comp(\mathcal{P} \cup b)$ may be jointly inconsistent: the modality represents then the shift of context. The completion construction does not seem to generalize immediately to universally quantified goals. The right kind of logic turns out to be a three-valued version of $K4$. The authors prove soundness and completeness of *ESLDNF* derivability w.r.t. a Kripke-Kleene fixpoint construction for a notion of non-floundering programs. The three-valued approach avoids the restriction to stratified programs.

5.5.3 *NF* in First-Order Uniform Proofs

Harland's thesis [Har91b] is an in-depth analysis of Uniform Proofs and *NF* at the first-order level. Unfortunately, his approach, though sound, is not adequate to our purposes, as we shall argue next. Harland inherits the traditional 'boolean' attitude w.r.t. the closed world assumption: predicates are either *completely defined* (CWA), as in the `append` program, or incomplete (OWA); while this distinction, he maintains, is essentially semantic and therefore enforced by user declarations, the latter coincide with the programs where temporary assumptions are made, i.e. clauses with embedded implications. Thus *NF* would make sense only for the former, since how will you apply the CWA to something that is by definition incomplete? Therefore those programs would require some form of real (say minimal) negation, much as in [Mil89c, Mom92]. Technically, this is achieved by distinguishing, for all predicates occurring in the signature, those which appear negatively versus positively in the body of clauses. The two sets are required to be disjoint. This of course simplifies the later development, but, unfortunately, if a predicate which appears negatively, i.e. it is assumed, cannot occur positively, then it will never be used in the derivation. Its assumption is totally irrelevant to the computation and may be discarded from the program. This fragment therefore collapses to normal programs with extensional quantification. While it is possible to enforce differently the distinction between complete and incomplete predicates, we are indeed interested to apply negation to incomplete predicates, although in a restricted fashion.

Then Harland shows how to formulate the completion as a first-order HHF formulae so as to simulate *NF* as derivability in the uniform proof system. The executable completion is formulated through contraposition. Programs are assumed to be locally stratified, hence the completion is consistent. Thus Harland identifies the completed program with its iff-completion, though he correctly has $\neg(D \rightarrow G) = D \rightarrow \neg G$. This operational interpretation of the *augment* instruction w.r.t. negation is nevertheless inconsistent with its formulation in the completion, where negation is interpreted classically insofar as the negative completion of $Q \leftarrow G$ is seen as $\neg(Q \wedge \neg G)$; again this holds only because *NF* cannot be applied to incomplete predicates. The classical attitude carries over to universal quantification, which is only allowed to be extensional. Harland so does not stress the intrinsic connection between implication and parametric universal quantification. Extensionality is achieved through covering as in [MMP88], though the role of the *DCA* (Domain Closure Axiom) is not made explicit. Therefore Harland is not able to correlate extensional uniform proofs with intuitionistic provability plus *DCA*. This is not surprising since its partition between complete and incomplete predicates restricts the AUGMENT rule so that its operational provability relations are not conservative extensions of standard uniform provability: this makes the usual proof-theoretic adequacy impossible. Moreover no operational semantics for coverings is described: not only is the role of free variables introduced by coverings unclear, but the extensional rules are totally non-deterministic w.r.t. the choice of the correct covering.

An equality/inequality solver on the Herbrand universe is described and used to solve inequalities stemming from the completed definitions: this algorithm is different from the traditional *uncover* algorithm as it does not return the most general solution but a possibly infinite enumeration of the latter; this permits not to left linearize the program.

5.5.4 Partial Inductive Definitions

Partial Inductive Definitions ([Hal91]) (PID) are a generalization of inductive definitions [Acz77] to definiens containing implications and in a finitary version ([Eri93]) of parametric quantification: they also incorporate a proof-theoretical notion of closure somehow as in the CWA, but brought upon by the principle of *definitional reflection*. Not every PID can be given a logical (in the broad sense) reading, see for instance the 'functional' definition of the `plus` predicate in [Kre92], and this may be the source of some misunderstandings. In a logical setting, following [SH93], we can take an intuitionistic sequent calculus parameterized by a set of *definitions*, i.e. a finite set of clauses $b \Leftarrow G$, where b ranges over atomic predicates possibly with free variables. For a definition $\mathcal{D}$, we call $\mathcal{D}(a) = \{\sigma G \mid b \Leftarrow G \in \mathcal{D}, a = \sigma b\}$. In the spirit of the "clauses-as-rules" paradigm [HSH90, HSH91] we add for every defined atom a, the rules

$$\frac{\Gamma \vdash G \quad G \in \mathcal{D}(a)}{\Gamma \vdash a}\,\mathcal{D}-R \qquad \frac{\{\sigma\Gamma, \sigma G \vdash A : \sigma = mgu(a,b), b \Leftarrow G \in \mathcal{D}\}}{\Gamma, a \vdash A}\,\mathcal{D}-L$$

Call this $DR(\mathcal{D})$, for a given set of definitions $\mathcal{D}$. The first rule corresponds to backchaining on the definitional clauses, while the second, in its ω-version, reflect the closure clause of (partial) inductive definitions. They can be seen as right and left introduction rules for the atomic proposition (inductive definition) a. This feature is used in the programming language *GCLA* [MAK91] which allows both functional and logic programming style. Search is conducted on sequents $\Gamma \vdash G$ and it is clearly not amenable of an uniform proof approach, since focusing is impossible as at any time each of the assumption in the antecedent may be expanded via $\mathcal{D} - L$. Indeed, without mentioning the issue of contraction, control in this setting has proven to be a major problem and this has lead to *GCLA II*, which is a middle ground between a logic programming language and a tactic theorem prover [Kre92]; here the user must not only provide the program in the form of a set of definitions, but also to what amount to a tactic (expressed itself as a PID) to direct the search and avoid a large number of meaningless (in logical terms) answer substitutions.

One basic problem is the failure of global cut-elimination: as discussed in [SH93], in order to reduce an atomic cut formula through the $\mathcal{D}-L/\mathcal{D}-R$ reduction, we may generate a cut with a more complex definiens formula. Definitions are called *total* if they enjoy cut-elimination, otherwise they are *partial.* Classes of total definitions include:

- Implication-free programs.
- Contraction-free logics.
- Stratified program w.r.t. negation.
- Stratified program w.r.t. implication [DM97].

In the latter case, predicates are assigned *levels*: the latter are then extended to formulae, so as to forbid recursion through negative occurrences; the level of an implication is essentially the order of its type: definitions are allowed only if the level of the heads is strictly greater than the level of the body: for instance $a \Leftarrow b \rightarrow a$ is allowed while $a \Leftarrow a \rightarrow b$ is not. This excludes every left-recursive definition.

Definitional reflection has been claimed to bring in a proof-theoretic notion of negation, although it would be more accurate to say that it allows a *meta-level* notion of closure. Intuitively, $\mathcal{D} - L$ works as the only-if clause of the completion: a proof of $\vdash \neg a$ is a proof of $a \vdash \bot$. This has been formalized by Schroeder-Heister who has proven [SH93] the equivalence between $comp(P)$ (formulated in a sequent style as a left and a right rule for each completed predicate, plus a formulation of free equality again in sequent style) and $DR(P) \cup (x = x \Leftarrow \top)$, where the latter clause, thanks to definitional reflection, suffices to define free equality. This result holds for any kind of definition and does not depend on cut-elimination. This has been hailed by the *GCLA* group as a major benefit of allowing definitional reflection, in so far as negative goals are simply a special case of implicational ones and therefore should be able to compute answer substitutions. This claim has some validity as far as normal stratified programs and ground goals are concerned: indeed they may yield only sequents such that if the antecedent is non-empty, it must be atomic and the consequent is $\bot$: then $\mathcal{D} - L$ must be used to introduce a definiens from a lower stratum. As mentioned above, search is here cut-free. Thus for example, $\neg even(s(0))$ is provable as follows, suppressing here any reference to the (empty) parameter context:

$$\frac{\overline{even(s(0)) \vdash_{\mathrm{def}(even)} \bot}\,\mathcal{D}-L}{\vdash_{\mathrm{def}(even)} even(s(0)) \rightarrow \bot}\rightarrow R$$

since the definiens of $even(s(0))$ is empty.

The equivalence with the completion holds irrespectively of the presence of local variables. In this remark, then this approach would seem to have an edge on the transformational one since it does not need extensional quantification.

Example 5.6 *Consider a graph* $edge(a, b), edge(b, c)$*: the deduction of* $path(b, a) \vdash \bot$ *leads to the conjunctive goals* $\{(edge(b, a) \vdash \bot), (edge(b, Z) \wedge path(Z, a) \vdash \bot)\}$*, where* Z *is a local logic variable; then there is a computation which reduces to* $\bot \ldots \vdash \bot$.

On the other hand, some of the familiar problems with negative goals arises when concerned with open queries. When logic variables are allowed PID's give the usual unsound reading to existential queries and one is faced again with the question of *failure substitutions*. For example the query $\vdash_{def(even)} \exists x \neg even(x)$ yields $even(X) \vdash_{def(even)} \bot$; the rule $\mathcal{D} - L$ will loop, being unable to compute to what amounts, in our terms, to $\mathrm{Not}(0) \cap \mathrm{Not}(s(s(Y)))$. In other terms, definitional reflection uses unification to consider the given definition as a minimal inductive definition, but this does not allow to compute values that lies outside this very definition. Although it is true that $\vdash_{def(even)} \neg even(s(0))$ and hence $\vdash_{def(even)} \exists x\, .\neg even(s(x))$ by $\exists - I$, it does not mean that the above query will retrieve the appropriate instantiation. This seems to point to an operational incompleteness of definitional reflection w.r.t. open queries. Besides, this is only an intellectual curiosity, since the aforementioned problems with search preclude anyway its adoption as a mechanism to handle negation. Not surprisingly, I am not aware of any implementation of the ω-rule for the purpose of enhancing a logic language with negation, rather than to allow an idea of ω-quantification. As a matter of fact, the rule of definitional reflection used in *GCLA* is the weaker 'logical' version [MAK91] which does not imply in general the completion. As Schroeder-Heister put it:

> "If we want to extend logic programming with definitional reflection then the logical rule is more adequate than the ω one. The idea of successfully computing an answer substitution, which is central to logic programming, is bound to *strong* closure under substitution [lifting lemma], which holds for the [former] but not for the [latter]..." [SH93].

In fairness, Eriksson [Eri92] has proposed a even stronger rule, which encompasses the two other versions "...of course with many algorithmic problems to efficiently compute bindings at application of [such a rule]", ibidem.

Notwithstanding the similarity, PID's give a striking different operational semantics to HHF. Consider for example the definition $a \Leftarrow b \to c$: then the sequent $\vdash \neg a$ has no cut-free proofs, while $\vdash a$ is provable against the operational intuition of HHF:

$$\dfrac{\dfrac{\dfrac{}{b \vdash c}\ \mathcal{D} - L}{\vdash b \to c}\ \to -R}{\vdash a}\ \mathcal{D} - R$$

Here, since b has no definition, it is assumed to be false and hence entails anything. This means that $\mathcal{D} - L$ is not conservative w.r.t. the positive fragment, i.e. the former rule may be used in proving positive atoms. In the complementation approach instead, positive and negative fragments are disjoint and exclusive. This is not surprising since, as mentioned above, the equivalence with the iff completion induces an operational semantics of failure which is distinct from the one typical of HHF. Moreover definitional reflection does not match well with parametric judgments. Consider a goal $\cdot; \cdot \vdash_{def(open)} \neg open\ lam(\lambda x\, .\, x)$, which is clearly unprovable: a naive logic programming interpreter augmented with definitional reflection will eventually yield the goal $\cdot; \forall x : exp.\, open\ x \vdash_{def(open)} \bot$. In the failed derivation of the same positive goal, a goal-oriented strategy would introduce a new parameter and try the goal $y{:}exp; \top \vdash_{def(open)} open\ y$. Yet, the same idea does not work when definitional reflection is allowed as now the universal in on the left and we can only do (inverted) universal elimination. Moreover, consider now the query $w{:}exp; \top \vdash_{def(closed)} closed\ lam(\lambda x\, .\, w)$ which should fail: yet this would yield a sequent $w, y{:}exp; closed\ y \vdash_{def(closed)} closed\ w$ and since $\mathcal{D}(closed\ y)$ is empty (both under the logical and ω-version), we would get the success node $\cdot, \bot \vdash_{def(closed)} closed\ w$!

The main confusion, in general, may lie in attempting to give a naive goal-oriented reading to a sequent calculus augmented with the definitional reflection rule. The latter is instead a meta-level closure operator;

this is indeed how it is used in meta-logics as $\mathcal{M}_2$ [SP98] and $FO\lambda^{\Delta I\!N}$ [MM97]. If we try to use it as a logic programming engine for HHF with negation, i.e. we add introduce logical variables and dynamic assumptions, it breaks down and reveal its meta-theoretic nature. We also remark that adopting our restriction to *complementable* programs (formally introduced in Figure 6.7) would not help, as far as logic programming is concerned. Instead, we conjecture that the proof of cut-elimination given in [MM00] can be extended to those programs, allowing to enhance the range of $FO\lambda^{\Delta I\!N}$ significantly.

Chapter 6

Clause Complementation

In this chapter we introduce the source language and its uniform proofs system. Before formalizing the restriction to terminating programs, we establish the fundamental notion of context schema. This allows to enforce the *Regular World Assumption* (RWA), on which clause complementation is built. Finally, we discuss how to give an operational semantics to our system.

6.1 The Logic

We will use the following slightly unusual source language. Again, we fix a signature Σ in advance which would otherwise clutter the presentation; let $\vec{M}, \vec{N}$ be sequences of *simple* terms:

$$\begin{array}{rcll} \textit{Atoms} & Q & ::= & q\ \vec{M} \mid \neg q\ \vec{M} \\ \textit{Goals} & G & ::= & Q \mid \top \mid \bot \mid \vec{M} \doteq \vec{N} \mid \vec{M} \neq \vec{N} \mid \\ & & & G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \to G \mid \forall x\!:\!A.\, G \mid \\ \textit{Clauses} & D & ::= & \top \mid \bot \mid Q \mid D \leftarrow G \mid D_1 \wedge D_2 \mid D_1 \vee D_2 \mid \forall x\!:\!A.\, D \end{array}$$

First and foremost, we restrict ourselves to clauses of at *most* third-order, that is we allow HHF which only make Horn assumptions. This simplifies the presentation of the complement algorithm, but it is not an unsurmountable obstacle; we comment on this and other restrictions in the Conclusions (Chapter 7). Differently form standard presentation of HHF, we do not allow existential goals. This rules out open queries and local variables, as well as spares us from the heavy machinery of *mixed prefixes* [Mil92]. On the other hand, we allow disjunction between clauses, equality and inequality. While disjunction among clauses will be eliminated, inequalities will survive, although they will always be solvable at run-time by a simple syntactic check, as we will see in due time (Section 6.9). This simplifies their treatment in type-theoretic languages such as *Twelf*, where, in general, (in)equalities should be viewed as types inhabited by appropriate proof terms. The tokens $\top, \bot$ may be decorated with a predicate symbol, as explained in Subsection 6.1.1 and Definition 6.21. Finally, we remark that '$\neg$' is *not* a connective, but a name constructor for atomic formulae.

$$\begin{array}{rcll} \textit{Parameter Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A \\ \textit{Assumptions} & \mathcal{D} & ::= & \top \mid \mathcal{D} \wedge D \end{array}$$

We call a pair of concrete context and assumption $\Gamma; \mathcal{D}$, such that all parameters occurring in $\mathcal{D}$ are mentioned in Γ, a *(run-time) context*. We make the usual conventions on contexts, in particular we avoid mentioning the leading $\cdot$ and $\top$ elements.

We use the notation $D \sqsubseteq \mathcal{P} \wedge \mathcal{D}$ to indicate that D is a top-level conjunct in the $\mathcal{P}$ or in $\mathcal{D}$.

We now introduce the uniform proofs judgments for provability and denial in Figure 6.1 and for immediate implication and denial in Figure 6.2. While the system for denial is introduced for technical reasons, namely the proof of the Exhaustivity Theorem 6.33, it is of independent interest, since it provide evidence (that is proof terms) for non-provability. A similar system was presented in [Har91b], although in a simplified

$$\frac{}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\top}\;\vdash\top \qquad \frac{}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\bot}\;\not\vdash\bot$$

$$\frac{\vec{M}=\vec{N}}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\vec{M}\doteq\vec{N}}\;\vdash\doteq \qquad \frac{\vec{M}\neq\vec{N}}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\vec{M}\doteq\vec{N}}\;\not\vdash\doteq$$

$$\frac{\vec{M}\neq\vec{N}}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\vec{M}\dot{\neq}\vec{N}}\;\vdash\dot{\neq} \qquad \frac{\vec{M}=\vec{N}}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\vec{M}\dot{\neq}\vec{N}}\;\not\vdash\dot{\neq}$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1 \qquad \Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1\wedge G_2}\;\vdash\wedge \qquad \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_1 \qquad \Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_1\vee G_2}\;\not\vdash\vee$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1\vee G_2}\;\vdash\vee_1 \qquad \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_1}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_1\wedge G_2}\;\not\vdash\wedge_1$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G_1\vee G_2}\;\vdash\vee_2 \qquad \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_2}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G_1\wedge G_2}\;\not\vdash\wedge_2$$

$$\frac{\Gamma;(\mathcal{D}\wedge D)\vdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D\to G}\;\vdash\to \qquad \frac{\Gamma;(\mathcal{D}\wedge D)\not\vdash_{\mathcal{P}}G}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}D\to G}\;\not\vdash\to$$

$$\frac{(\Gamma,y{:}A);\mathcal{D}\vdash_{\mathcal{P}}[y/x]G}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\forall x{:}A.\,G}\;\vdash\forall^{y} \qquad \frac{(\Gamma,y{:}A);\mathcal{D}\not\vdash_{\mathcal{P}}[y/x]G}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\forall x{:}A.\,G}\;\not\vdash\forall^{y}$$

$$\frac{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}\mathcal{P}\wedge\mathcal{D}\gg Q}{\Gamma;\mathcal{D}\vdash_{\mathcal{P}}Q}\;\vdash\mathrm{atm} \qquad \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\mathcal{P}\wedge\mathcal{D}\gg Q}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}Q}\;\not\vdash\mathrm{atm}$$

Figure 6.1: Provability and denial

setting without parametric judgments. We remark that the systems for denial will instead not be needed in the proof of exclusivity (Theorem 6.32).

Due to the presence of '$\vee$' as a clause constructor, uniform proofs are *not* complete w.r.t. intuitionistic logic. We will remedy this situation in Section 6.9.2. The rules are depicted (when possible) in two columns where every row displays a positive rule and its negative counterpart.

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}}G$	Program $\mathcal{P}$ and assumption $\mathcal{D}$ uniformly entail G.
$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}G$	Program $\mathcal{P}$ and assumption $\mathcal{D}$ uniformly deny G.
$\Gamma;\mathcal{D}\vdash_{\mathcal{P}}D\gg Q$	Clause D from $\mathcal{P}\wedge\mathcal{D}$ immediately entails atom Q.
$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}D\gg Q$	Clause D from $\mathcal{P}\wedge\mathcal{D}$ immediately denies atom Q.

We briefly comment on the rules: the (in)equalities rules simply mirror the object logic symbols $\doteq,\dot{\neq}$ in meta-level (in)equality. The denial rules for implication and universal quantification reflect the operational semantics of unprovability that we have discussed earlier. Note that $\not\gg\forall$ is an infinitary rule, due to the meta-linguistic extensional universal quantification on all terms. Rule $\not\gg\forall$ must be read as: for every well-typed term n in parameter context Γ, $\forall x{:}A.\,D$ immediately denies Q if so does $[n/x]D$. Rules $\vdash\forall$, $\not\vdash\forall$ are instead parametric in the new eigenvariable, say y; the $()^{y}$ superscript reminds us of the parameter condition on y, e.g. that y does not occur free in $\Gamma;\mathcal{D}$ nor in G. We will use this notation in any other parametric rule.

Differently from the latter, we will also use *global* eigenvariables, say u (see for example rule $\overset{\mathcal{D}}{\Longrightarrow}\forall$ in Figure 6.3); this expresses the fact that in those rules the relation holds for *any* term; those parameters are therefore analogous to logic variables. We will pervasively utilize this notation; however, to avoid notational

$$\frac{}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \bot \gg Q}\ {\gg}\bot \qquad \frac{}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} \top \gg Q}\ {\not\gg}\top$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} [t/x]D \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \forall x{:}A.\, D \gg Q}\ {\gg}\forall$$

$$\frac{\text{for all } n\ \Gamma \vdash n : A \qquad \Gamma;\mathcal{D} \nvdash_{\mathcal{P}} [n/x]D \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} \forall x{:}A.\, D \gg Q}\ {\not\gg}\forall$$

$$\frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q}\ {\gg}\wedge_1 \qquad \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_1 \vee D_2 \gg Q}\ {\not\gg}\vee_1$$

$$\frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q}\ {\gg}\wedge_2 \qquad \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_1 \vee D_2 \gg Q}\ {\not\gg}\vee_2$$

$$\frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \vee D_2 \gg Q}\ {\gg}\vee$$

$$\frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q \qquad \Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q}\ {\not\gg}\wedge$$

$$\frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D \gg Q \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} G}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D \leftarrow G \gg Q}\ {\gg} \rightarrow$$

$$\frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D \leftarrow G \gg Q}\ {\not\gg}{\rightarrow_1} \qquad \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D \leftarrow G \gg Q}\ {\not\gg}{\rightarrow_2}$$

Figure 6.2: Immediate entailment and denial

clutter, when possible we shall not keep an explicit record of global parameters, but we agree to implicitly gather them in a pool Φ, which we can access when needed, in particular to type-check substitutions, see Section 6.2.

For the sake of conciseness we will often use in this thesis the following rule schema, where $\mathcal{J}(X, R)$ is a judgment involving a token X ranging over a set $\mathcal{T}$ and a relation R:

$$\frac{X \in \mathcal{T}}{\mathcal{J}(X, R)}\ R\ X$$

6.1.1 $\top$-Normalization

We show in this section how to put every program in a normalized format w.r.t. assumptions so that every goal in the scope of an universal quantifier depends on some assumption, possibly the trivial clause $\top$. This has also the effect of 'localizing' the trivial assumption to its atom, a property will be very useful while performing augmentation, see Section 6.6.

We first state some basic properties of provability.

Lemma 6.1 (Weakening)

1. *If* $\Gamma; \mathcal{D} \vdash G$*, then:*
   (a) $(\Gamma, x{:}A); \mathcal{D} \vdash G$.
   (b) $\Gamma; (\mathcal{D} \wedge D) \vdash G$.
2. *If* $\Gamma; \mathcal{D} \vdash D \gg Q$*, then:*
   (a) $(\Gamma, x{:}A); \mathcal{D} \vdash D \gg Q$.
   (b) $\Gamma; (\mathcal{D} \wedge D') \vdash D \gg Q$.

Proof: A straightforward mutual induction on the given derivations. □

In the proof of Theorem 6.5 we will need the following form of Strengthening:

Lemma 6.2 ($\top$-Strengthening)

1. *If* $\Gamma; (\mathcal{D} \wedge \top) \vdash G$*, then* $\Gamma; \mathcal{D} \vdash G$.
2. *If* $\Gamma; (\mathcal{D} \wedge \top) \vdash D \gg Q$*, then* $\Gamma; \mathcal{D} \vdash D \gg Q$.

Proof: By an easy mutual induction on the structure of the given derivations. □

Normalization is realized in Figure 6.3 by the two judgments $D \stackrel{D^\top}{\Longrightarrow} D^\top$ and $\Gamma; \mathcal{D} \vdash G \stackrel{G^\top}{\Longrightarrow} G^\top$. The only interesting case is when we reach an atomic goal: if no parameter has been introduced, the clause does not require any normalization. If $\Gamma; \mathcal{D}$ is non-empty, we adopt a brute-force approach and add the trivial clause for *every* predicate in the signature. This is largely unnecessary, but it gives a very simple account of mutually recursion, without an explicit appeal to a call graph. As a matter of fact, we would really need to $\top$-normalize an atom Q with the trivial clause for every predicate that is mutually recursive to Q and such that the type of the parameter in the context is not *subordinate* [Vir99] to the type of Q. On the other hand Theorems 6.4 and 6.5 ensures, as obvious, that this program transformation is harmless, since it preserves provability and denial. Moreover, we will engineer the rule for clause complementation so that the irrelevant $\top_q$ will be inactive. We will remove those irrelevant augmentations in a final pass (Section 6.9.2).

Example 6.3 *Consider the lambda clause in the* `open` *program:*

$$\begin{array}{rcl} oplam & : & \forall E{:}exp \to exp.\ open\ (lam\ E) \\ & & \leftarrow \forall x{:}exp.\ open\ (E\ x). \end{array}$$

oplam $\stackrel{\mathcal{D}^\top}{\Longrightarrow}$ *oplam*$^\top$, *i.e.*

$$\begin{aligned} oplam^\top \quad : \quad & \forall E{:}exp \to exp.\, open\ (lam\ E) \\ & \leftarrow (\forall x{:}exp.\, \top_{open} \to open\ (E\ x)). \end{aligned}$$

Consider this fragment of code involving two mutually recursive predicates:

$$\begin{aligned} qlam \quad : \quad & \forall E{:}exp \to exp.\, q\ (lam\ E) \\ & \leftarrow \forall x{:}exp.\, p\ (E\ x). \\ papp \quad : \quad & \forall E_1{:}exp.\, \forall E_2{:}exp.\, p\ (app\ E_1\ E_2) \\ & \leftarrow q\ E_1 \wedge p\ E_2. \end{aligned}$$

Then:

$$\begin{aligned} qlam^\top \quad : \quad & \forall E{:}exp \to exp.\, q\ (lam\ E) \\ & \leftarrow (\forall x{:}exp.\, \top_p \wedge \top_q \to p\ (E\ x)). \end{aligned}$$

In examples, we will not mention inactive $\top$ clauses.

We start by showing that this transformation preserves run-time provability. We extend the notion of *application* of a substitution, notation $[t/x]F$, to a formula F. More in general, we denote with $[\theta]F$ the application of θ to F. We will also need an operation of composition, say $\sigma_1 \cdot \sigma_2$, such that $[\sigma_1 \cdot \sigma_2]F = [\sigma_2]([\sigma_1]F)$.

Theorem 6.4 *Let* $\Gamma'; \mathcal{D}' \vdash G \stackrel{G^\top}{\Longrightarrow} G^\top$, $D \stackrel{D^\top}{\Longrightarrow} D^\top$, $\mathcal{D} \stackrel{D^\top}{\Longrightarrow} \mathcal{D}^\top$ *and* $\Gamma \vdash \theta : \Phi$.

1. *If* $\Gamma; [\theta]\mathcal{D} \vdash [\theta]G$, *then* $\Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]G^\top$.
2. *If* $\Gamma; [\theta]\mathcal{D} \vdash [\theta]D \gg [\theta]Q$, *then* $\Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]D^\top \gg [\theta]Q$.

Proof: By mutual induction on the structure of the derivation of $\gamma :: \Gamma; [\theta]\mathcal{D} \vdash [\theta]G$ and inversion on $\gamma' :: \Gamma'; \mathcal{D}' \vdash G \stackrel{G^\top}{\Longrightarrow} G^\top$ together with $\delta :: \Gamma; [\theta]\mathcal{D} \vdash [\theta]D \gg [\theta]Q$ and inversion on $\delta' :: D \stackrel{D^\top}{\Longrightarrow} D^\top$. We sketch only some cases:

Case: γ ends in $\vdash$ At and γ' is $\stackrel{G^\top}{\Longrightarrow} X, \stackrel{G^\top}{\Longrightarrow}$ At-emp: trivial.

Case: γ ends in $\vdash$ At and γ' ends in $\stackrel{G^\top}{\Longrightarrow}$ At:

$\Gamma; [\theta]\mathcal{D} \vdash [\theta]Q$	By hypothesis
$\Gamma; [\theta]\mathcal{D} \vdash [\theta](\mathcal{P} \wedge \mathcal{D}) \gg [\theta]Q$	By sub-derivation
$\Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]\mathcal{P} \wedge \mathcal{D}^\top \gg [\theta]Q$	By IH 2
$\Gamma; ([\theta]\mathcal{D}^\top \wedge (\bigwedge_{p \in \Sigma} \top_p)) \vdash [\theta]\mathcal{P} \wedge \mathcal{D}^\top \gg [\theta]Q$	By repeated Weakening
$\Gamma; ([\theta]\mathcal{D}^\top \wedge (\bigwedge_{p \in \Sigma} \top_p)) \vdash [\theta]Q$	By rule $\vdash$ At
$\Gamma; [\theta]\mathcal{D}^\top \vdash (\bigwedge_{p \in \Sigma} \top_p) \to [\theta]Q$	By rule $\vdash\to$
$\Gamma; [\theta]\mathcal{D} \vdash [\theta]Q^\top$	By rule $\stackrel{G^\top}{\Longrightarrow}$ At

The other cases follows immediately by IH 1 and 2.

□

Now, the converse:

Theorem 6.5 *Let* $\Gamma'; \mathcal{D}' \vdash G \stackrel{G^\top}{\Longrightarrow} G^\top$, $D \stackrel{D^\top}{\Longrightarrow} D^\top$, $\mathcal{D} \stackrel{D^\top}{\Longrightarrow} \mathcal{D}^\top$ *and* $\Gamma \vdash \theta : \Phi$.

1. *If* $\Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]G^\top$, *then* $\Gamma; [\theta]\mathcal{D} \vdash [\theta]G$.

$$\frac{X \in \{\top, \bot\} \text{ or (dis)eq}}{\Gamma; \mathcal{D} \vdash X \overset{G^\top}{\Longrightarrow} X} \overset{G^\top}{\Longrightarrow} X$$

$$\frac{}{\cdot; \top \vdash Q \overset{G^\top}{\Longrightarrow} Q} \overset{G^\top}{\Longrightarrow} \text{At-emp} \qquad \frac{}{\Gamma; \mathcal{D} \vdash Q \overset{G^\top}{\Longrightarrow} (\bigwedge_{p \in \Sigma} \top_p \to Q)} \overset{G^\top}{\Longrightarrow} \text{At}$$

$$\frac{\Gamma; (\mathcal{D} \wedge D) \vdash G \overset{G^\top}{\Longrightarrow} G^\top \qquad D \overset{D^\top}{\Longrightarrow} D^\top}{\Gamma; \mathcal{D} \vdash D \to G \overset{G^\top}{\Longrightarrow} D^\top \to G^\top} \overset{G^\top}{\Longrightarrow} \to$$

$$\frac{(\Gamma, y{:}A); \mathcal{D} \vdash [y/x]G \overset{G^\top}{\Longrightarrow} [y/x]G^\top}{\Gamma; \mathcal{D} \vdash \forall x{:}A.\, G \overset{G^\top}{\Longrightarrow} \forall x{:}A.\, G^\top} \overset{G^\top}{\Longrightarrow} \forall^y$$

$$\frac{\Gamma; \mathcal{D} \vdash G_1 \overset{G^\top}{\Longrightarrow} G_1^\top \qquad \Gamma; \mathcal{D} \vdash G_2 \overset{G^\top}{\Longrightarrow} G_2^\top}{\Gamma; \mathcal{D} \vdash G_1 \wedge G_2 \overset{G^\top}{\Longrightarrow} G_1^\top \wedge G_2^\top} \overset{G^\top}{\Longrightarrow} \wedge$$

$$\frac{\Gamma; \mathcal{D} \vdash G_1 \overset{G^\top}{\Longrightarrow} G_1^\top \qquad \Gamma; \mathcal{D} \vdash G_2 \overset{G^\top}{\Longrightarrow} G_2^\top}{\Gamma; \mathcal{D} \vdash G_1 \vee G_2 \overset{G^\top}{\Longrightarrow} G_1^\top \vee G_2^\top} \overset{G^\top}{\Longrightarrow} \vee$$

$$\frac{X \in \{Q, \top, \bot\}}{X \overset{D^\top}{\Longrightarrow} X} \overset{D^\top}{\Longrightarrow} X$$

$$\frac{\cdot; \top \vdash G \overset{G^\top}{\Longrightarrow} G^\top \qquad D \overset{D^\top}{\Longrightarrow} D^\top}{(G \to D) \overset{D^\top}{\Longrightarrow} G^\top \to D^\top} \overset{D^\top}{\Longrightarrow} \to$$

$$\frac{[u/x]D \overset{D^\top}{\Longrightarrow} [u/x]D^\top}{\forall x{:}A.\, D \overset{D^\top}{\Longrightarrow} \forall x{:}A.\, D^\top} \overset{D^\top}{\Longrightarrow} \forall^u$$

$$\frac{D_1 \overset{D^\top}{\Longrightarrow} D_1^\top \qquad D_2 \overset{D^\top}{\Longrightarrow} D_2^\top}{D_1 \wedge D_2 \overset{D^\top}{\Longrightarrow} D_1^\top \wedge D_2^\top} \overset{D^\top}{\Longrightarrow} \wedge \qquad \frac{D_1 \overset{D^\top}{\Longrightarrow} D_1^\top \qquad D_2 \overset{D^\top}{\Longrightarrow} D_2^\top}{D_1 \vee D_2 \overset{D^\top}{\Longrightarrow} D_1^\top \vee D_2^\top} \overset{D^\top}{\Longrightarrow} \vee$$

Figure 6.3: $\top$-Normalization

2. *If* $\Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]D^\top \gg [\theta]Q$*, then* $\Gamma; [\theta]\mathcal{D} \vdash [\theta]D \gg [\theta]Q$.

Proof: By mutual induction on the structure of $\gamma :: \Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]G^\top$ and $\delta :: \Gamma; [\theta]\mathcal{D}^\top \vdash [\theta]D^\top \gg [\theta]Q$, as above but using $\top$-Strengthening (Lemma 6.2) in place of Weakening. □

An analogous result can be proven w.r.t. failure; the latter would require $\top$-Weakening and Strengthening w.r.t. the (immediate) denial relation. On the other hand, the above result will suffice, if we restrict to terminating programs as we will do in Section 6.3. First, we need to establish a theory of *context schemata*

6.2 Context Schemata

In this Section we address the properties of contexts[1]. Hereditary Harrop formulae differ from Horn clauses in that they may dynamically extend the current signature and program; this is reflected in the provability (and denial) relation which is parameterized by a run-time context $\Gamma; \mathcal{D}$. As we have argued before (Section 5.5) we cannot obtain closure under clause complementation for the full logic of HHF, but we have to restrict ourselves to a smaller (but significant) fragment. This in turn entails that we have to make sure that during execution whenever an assumption is made, the latter stays in the fragment we have isolated. Technically, we proceed as follows:

- We extract from the static definition of a predicate the general 'template' of a legal assumption.
- We require dynamic assumptions to conform to this template.

We thus introduce the notion of *schema satisfaction*, for which we will need the following data structure: a *schema context* abstracts over all possible instantiation of a run-time context. To account for that, we introduce a quantifier-like operator, say SOME $\Phi. \mathcal{D}$, which takes a clause and existentially bounds its free variables, i.e. the global parameters occurring in $\mathcal{D}$. We use the following reification of schemata: do not confuse '$\|$', which is used to represent schema alternatives in the object language with '$|$' which does the same in our informal meta-language.

$$\textit{Contexts Schemata} \quad \mathcal{S} \quad ::= \quad \circ \mid \mathcal{S} \| (\Gamma; \mathrm{SOME}\ \Phi. \mathcal{D})$$

Example 6.6 *The schema context grammar induced by the* `linear` *predicate (Example 5.2) is as follows:*

$$\mathcal{S}_{linear} \quad = \quad \circ \| x{:}exp; linear(x) \| x{:}exp; \top_{linx}$$

while a possible run-time context is

$$y_1{:}exp, y_2{:}exp, y_3{:}exp; \top_{linx} \wedge \top_{linx} \wedge linear(y_3)$$

Since every possible assumption is closed, there are no existential bindings. Consider instead the clauses for type-checking in the simply typed calculus:

$$\begin{array}{lcl} ofapp & : & \forall E_1, E_2 : exp. \forall T_1, T_2, T : tp. \\ & & \quad of\ (app\ E_1\ E_2)\ T_2 \\ & & \qquad \leftarrow of\ E_1\ (arrow\ T_1\ T_2) \\ & & \qquad \leftarrow of\ E_2\ T_1. \\ oflam & : & \forall E : exp \to exp. \forall T_1, T_2 : tp. \\ & & \quad of\ lam(E)\ (arrow\ T_1\ T_2) \\ & & \qquad \leftarrow (\forall x : exp. of\ x\ T_1 \to of\ (E\ x)\ T_2). \end{array}$$

Then the schema context is:

$$\mathcal{S}_{of} \quad = \quad \circ \| x{:}exp; SOME\ u : tp. of\ x\ u$$

where 'u' is a global parameter.

[1]This section is inspired by Schürmann's treatment of similar material in his dissertation [Sch00].

We will also need to disambiguate *blocks* in run-time contexts; overlapping may indeed happen when the alternatives in a context schema are not disjoint. For example, suppose we are given the following schema and partial run-time context:

$$\mathcal{S} \;=\; \circ \| x{:}exp; p(x) \| x{:}exp; p(x) \wedge q(x)$$

$$y_1{:}exp, y_2{:}exp; p(y_1) \wedge p(y_2)$$

It is not uniquely determined whether, say, $p(y_2)$ is an instance of the first alternative or a prefix of the second one. To determine that, we simply modify the provability and denial rules $\vdash$ At and $\not\vdash$ At by 'packaging' run-time contexts into blocks with a bracket operator $\lceil \cdot \rceil$. Intuitively, a block is complete when an atomic conclusion is reached during the deduction. It is therefore bracketed when the run-time context is passed to the immediate implication judgment.

$$\frac{D \sqsubseteq \mathcal{P} \wedge \mathcal{D} \qquad \lceil\Gamma\rceil; \lceil\mathcal{D}\rceil \vdash_{\mathcal{P}} D \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} Q} \vdash \mathrm{At} \qquad \frac{\text{for every } D \sqsubseteq \mathcal{P} \wedge \mathcal{D} \qquad \lceil\Gamma\rceil; \lceil\mathcal{D}\rceil \not\vdash_{\mathcal{P}} D \gg Q}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} Q} \not\vdash \mathrm{At}$$

Still, we need to 'flatten' blocks, so that re-bracketing during execution will be immaterial on already delimited blocks; we thus require the bracket operator to satisfy the following absorption-distribution law:

$$\begin{array}{rcl} \lceil \lceil\Gamma\rceil, \Gamma' \rceil & = & \lceil\Gamma\rceil, \lceil\Gamma'\rceil \\ \lceil \lceil\mathcal{D}\rceil \wedge \mathcal{D}' \rceil & = & \lceil\mathcal{D}\rceil \wedge \lceil\mathcal{D}'\rceil \end{array}$$

We are now ready to introduce schema satisfaction; this is accomplished by the following judgments:

- $\models_{\mathcal{S}} D$: clause D satisfies schema $\mathcal{S}$.
- $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$: (partial) run-time context $\Gamma; \mathcal{D}$ together with goal G satisfies schema $\mathcal{S}$.
- $\Gamma; \mathcal{D} < \mathcal{S}$: (completed) run-time context $\Gamma; \mathcal{D}$ satisfies schema $\mathcal{S}$.
- $\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S}$: block $\Gamma'; \mathcal{D}'$ in context Γ occurs in schema $\mathcal{S}$.

First, we say that a completed block *occurs* in a schema when the block is an alphabetic variant of some instantiation of one of the alternatives of the schema:

$$\frac{\Gamma \vdash \theta : \Phi \qquad (\Gamma'; \mathcal{D}') \equiv_\alpha (\Gamma''; [\theta]\mathcal{D})}{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \| (\Gamma''; \mathrm{SOME}\ \Phi. \mathcal{D})} \in_1 \qquad \frac{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S}}{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \| (\Gamma''; \mathcal{D})} \in_2$$

The judgment $\Gamma \vdash \theta : \Phi$ formalizes that θ is a valid well-typed substitution w.r.t. Γ.

$$\frac{}{\Gamma \vdash \epsilon : \cdot} \epsilon\cdot \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash \theta : \Phi}{\Gamma \vdash \theta, t/x : (\Phi, x : A)} \theta\Phi$$

Let us now analyze the rules in Figure 6.4. The empty run-time context is an instance of every schema. Moreover if Γ' and $\mathcal{D}'$ are completed blocks which occur in $\mathcal{S}$, as denoted by the block notation, then $(\Gamma, \lceil\Gamma'\rceil); (\mathcal{D} \wedge \lceil\mathcal{D}'\rceil)$ is an instance of $\mathcal{S}$, provided that $\mathcal{D}'$ is a valid clause. Thus we need to define the instance relation simultaneously to clause (and in turn to goal) satisfaction. The judgment $\models_{\mathcal{S}} D$ is merely auxiliary to the $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$ one: here we mimic the construction on the run-time schema until, in the base case, we check whether the resulting context is an instance of the given schema.

The following Lemma ensures that when an assumption D is pulled from a legal run-time context, i.e. which satisfies a given schema, so does D.

Lemma 6.7 *If* $\Gamma; \mathcal{D} < \mathcal{S}$ *and* $D \sqsubseteq \mathcal{D}$, *then* $\models_{\mathcal{S}} D$.

Proof: By induction on the structure of the derivation of $\pi :: \Gamma; \mathcal{D} < \mathcal{S}$.

$$\frac{X \in \{Q, \top, \bot\} \text{ or (dis)eq} \qquad \Gamma; \mathcal{D} < \mathcal{S}}{\Gamma; \mathcal{D} \setminus X < \mathcal{S}} \setminus X$$

$$\frac{\Gamma; \mathcal{D} \setminus G_1 < \mathcal{S} \qquad \Gamma; \mathcal{D} \setminus G_2 < \mathcal{S}}{\Gamma; \mathcal{D} \setminus G_1 \wedge G_2 < \mathcal{S}} \setminus \wedge$$

$$\frac{\Gamma; \mathcal{D} \setminus G_1 < \mathcal{S} \qquad \Gamma; \mathcal{D} \setminus G_2 < \mathcal{S}}{\Gamma; \mathcal{D} \setminus G_1 \vee G_2 < \mathcal{S}} \setminus \vee$$

$$\frac{(\Gamma, y{:}A); \mathcal{D} \setminus [y/x]G < \mathcal{S}}{\Gamma; \mathcal{D} \setminus \forall x{:}A.\, G < \mathcal{S}} \setminus \forall^y$$

$$\frac{\models_{\mathcal{S}} D \qquad \Gamma; (\mathcal{D} \wedge D) \setminus G < \mathcal{S}}{\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}} \setminus \to$$

$$\frac{X \in \{Q, \top \bot\}}{\models_{\mathcal{S}} X} \models_{\mathcal{S}} X$$

$$\frac{\models_{\mathcal{S}} D_1 \qquad \models_{\mathcal{S}} D_2}{\models_{\mathcal{S}} D_1 \wedge D_2} \models_{\mathcal{S}} \wedge$$

$$\frac{\models_{\mathcal{S}} D_1 \qquad \models_{\mathcal{S}} D_2}{\models_{\mathcal{S}} D_1 \vee D_2} \models_{\mathcal{S}} \vee$$

$$\frac{\models_{\mathcal{S}} [u/x]D}{\models_{\mathcal{S}} \forall x : A.\, D} \models_{\mathcal{S}} \forall^u$$

$$\frac{\models_{\mathcal{S}} D \qquad \cdot; \top \setminus G < \mathcal{S}}{\models_{\mathcal{S}} G \to D} \models_{\mathcal{S}} \to$$

$$\frac{}{\cdot; \top < \mathcal{S}} <_1 \qquad \frac{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \quad \models_{\mathcal{S}} \mathcal{D}' \quad (\Gamma; \mathcal{D}) < \mathcal{S}}{(\Gamma, \lceil \Gamma' \rceil); (\mathcal{D} \wedge \lceil \mathcal{D}' \rceil) < \mathcal{S}} <_2$$

Figure 6.4: Judgments $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$, $\models_{\mathcal{S}} D$ and $\Gamma; \mathcal{D} < \mathcal{S}$

Case:

$$\pi = \frac{}{\cdot; \top < \mathcal{S}} <_1$$

Then $D = \top$ and immediately

$$\pi' = \frac{}{\cdot; \top \models_{\mathcal{S}} \top} \models_{\mathcal{S}} \top$$

Case:

$$\pi = \frac{\Gamma'' \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \quad \models_{\mathcal{S}} \mathcal{D}' \quad (\Gamma''; \mathcal{D}'') < \mathcal{S}}{(\Gamma'', \lceil \Gamma' \rceil); (\mathcal{D}'' \wedge \lceil \mathcal{D}' \rceil) < \mathcal{S}} <_2$$

Subcase: $D \equiv \mathcal{D}'$

$\models_{\mathcal{S}} \mathcal{D}'$ By hypothesis

Subcase: $D \sqsubseteq \mathcal{D}''$

$\Gamma''; \mathcal{D}'' < \mathcal{S}$ By sub-derivation

$\models_{\mathcal{S}} D$ By IH

□

We can also show that schema satisfaction is closed under substitution:

Lemma 6.8 *Let* $\Gamma; \mathcal{D} < \mathcal{S}$ *and* $\Gamma \vdash \theta : \Phi$.

1. *If* $\models_{\mathcal{S}} D$*, then* $\models_{\mathcal{S}} [\theta]D$.
2. *If* $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$*, then* $\Gamma; [\theta]\mathcal{D} \setminus [\theta]G < \mathcal{S}$.
3. *If* $\Gamma; \mathcal{D} < \mathcal{S}$ *then* $\Gamma; [\theta]\mathcal{D} < \mathcal{S}$.

Proof: By a straightforward mutual induction on the structure of $\pi :: \models_{\mathcal{S}} D$, $\gamma :: \Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\sigma :: \Gamma; \mathcal{D} < \mathcal{S}$. □

6.2.1 Schema Extraction

Roughly, we extract a context schema by collecting all negative occurrences in a goal. In fact, those will be dynamically added to the static program under evaluation. This is achieved by the judgment $\Gamma; \mathcal{D} \vdash_{\Phi} G \stackrel{G}{\Longrightarrow} \mathcal{S}$. The latter is mutually recursive to the judgment $\vdash_{\Phi} D \stackrel{D}{\Longrightarrow} \mathcal{S}$ which collects schemata for each clause in the given program, as depicted in Figure 6.5. When an atomic goal is reached, the current list of parameters and assumptions is returned, with the correct existential binding inferred from the context of global variables. With the notation $\Phi_{\mathcal{D}}$ we mean the restriction of context Φ to the free variables of $\mathcal{D}$, i.e. $\{u{:}A \mid u \in \mathrm{dom}(\Phi), u \in FV(\mathcal{D})\}$. We make the convention to absorb schema alternatives which are α-variants. Moreover the empty context behaves as a left and right zero element for $\|$, i.e. :

$$(\Gamma; \mathcal{D} \| \cdot; \top) = (\cdot; \top \| \Gamma; \mathcal{D}) = \Gamma; \mathcal{D}$$

Finally $\|$ is commutative.

Example 6.9 *If* $\mathcal{P}$ *is the program in Example 5.2,* $\mathcal{P} \stackrel{D}{\Longrightarrow} x{:}exp; linear(x) \| x{:}exp; \top_{linx}$.

We aim to show that if a schema context is extracted by a program, the latter satisfies the former. First, we need to establish two weakening lemmata w.r.t. schema alternatives:

Lemma 6.10 *If* $\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S}$ *then* $\Gamma \vdash \Gamma'; \mathcal{D}' \in S' \| S$.

Proof: By a straightforward induction on the structure of $\pi :: \Gamma; \mathcal{D} \in \mathcal{S}$. □

$$\frac{X \in \{Q, \top, \bot\} \text{ or (dis)eq}}{\Gamma; \mathcal{D} \vdash_\Phi X \overset{G}{\Longrightarrow} \Gamma; \text{SOME } \Phi_{\mathcal{D}}. \mathcal{D}} \overset{G}{\Longrightarrow} X$$

$$\frac{\Gamma; (\mathcal{D} \wedge D) \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\Gamma; \mathcal{D} \vdash_\Phi D \to G \overset{G}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{G}{\Longrightarrow} \to$$

$$\frac{(\Gamma, y{:}A); \mathcal{D} \vdash_\Phi [y/x]G \overset{G}{\Longrightarrow} \mathcal{S}}{\Gamma; \mathcal{D} \vdash_\Phi \forall x : A.\, G \overset{G}{\Longrightarrow} \mathcal{S}} \overset{G}{\Longrightarrow} \forall^y$$

$$\frac{\Gamma; \mathcal{D} \vdash_\Phi G_1 \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \Gamma; \mathcal{D} \vdash_\Phi G_2 \overset{G}{\Longrightarrow} \mathcal{S}_2}{\Gamma; \mathcal{D} \vdash_\Phi G_1 \wedge G_2 \overset{G}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{G}{\Longrightarrow} \wedge$$

$$\frac{\Gamma; \mathcal{D} \vdash_\Phi G_1 \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \Gamma; \mathcal{D} \vdash_\Phi G_2 \overset{G}{\Longrightarrow} \mathcal{S}_2}{\Gamma; \mathcal{D} \vdash_\Phi G_1 \vee G_2 \overset{G}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{G}{\Longrightarrow} \vee$$

$$\frac{X \in \{Q, \top, \bot\}}{\vdash_\Phi X \overset{D}{\Longrightarrow} \cdot; \top} \overset{D}{\Longrightarrow} X$$

$$\frac{\cdot; \top \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\vdash_\Phi (G \to D) \overset{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{D}{\Longrightarrow} \to$$

$$\frac{\vdash_{\Phi, u:A} [u/x]D \overset{D}{\Longrightarrow} \mathcal{S}}{\vdash_\Phi \forall x : A.\, D \overset{D}{\Longrightarrow} \mathcal{S}} \overset{D}{\Longrightarrow} \forall^u$$

$$\frac{\vdash_\Phi D_1 \overset{D}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D_2 \overset{D}{\Longrightarrow} \mathcal{S}_2}{\vdash_\Phi D_1 \wedge D_2 \overset{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{D}{\Longrightarrow} \wedge$$

$$\frac{\vdash_\Phi D_1 \overset{D}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D_2 \overset{D}{\Longrightarrow} \mathcal{S}_2}{\vdash_\Phi D_1 \vee D_2 \overset{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{D}{\Longrightarrow} \vee$$

Figure 6.5: Extracting contexts schemata

Note also that $\Gamma \vdash \Gamma'; \mathcal{D}' \in S' \| S$ iff $\Gamma \vdash \Gamma'; \mathcal{D}' \in S \| S'$.

Lemma 6.11 (Schema Weakening) *Let $\Gamma; \mathcal{D} < \mathcal{S}$. Then*

1. *If $\models_{\mathcal{S}} D$, then $\models_{\mathcal{S}\|\mathcal{S}'} D$.*
2. *If $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$, then $\Gamma; \mathcal{D} \setminus G < \mathcal{S}\|\mathcal{S}'$.*
3. *If $\Gamma; \mathcal{D} < \mathcal{S}$ then $\Gamma; \mathcal{D} < \mathcal{S}\|\mathcal{S}'$.*

Proof: By mutual induction on the structure of $\pi :: \models_{\mathcal{S}} D$, $\gamma :: \Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\sigma :: \Gamma; \mathcal{D} < \mathcal{S}$. We show the crucial cases:

Case:

$$\pi = \frac{X \in \{Q, \top, \bot\}}{\models_{\mathcal{S}} X} \models_{\mathcal{S}} X$$

Trivially

$$\pi' = \frac{X \in \ \{Q, \top, \bot\}}{\models_{\mathcal{S}\|\mathcal{S}'} Q} \models_{\mathcal{S}} X$$

Case:

$$\pi = \frac{\models_{\mathcal{S}} D \qquad \cdot; \top \setminus G < \mathcal{S}}{\models_{\mathcal{S}} G \to D} \models_{\mathcal{S}}\!\to$$

$\models_{\mathcal{S}} D$	By sub-derivation
$\models_{\mathcal{S}\|\mathcal{S}'} D$	By IH 1
$\cdot; \top \setminus G < \mathcal{S}$	By sub-derivation
$\cdot; \top \setminus G < \mathcal{S}\|\mathcal{S}'$	By IH 2
$\models_{\mathcal{S}\|\mathcal{S}'} G \to D$	By rule $\models_{\mathcal{S}}\!\to$

Case:

$$\gamma = \frac{X \in \{Q, \top, \bot\} \text{ or (dis)eq} \qquad \Gamma; \mathcal{D} < \mathcal{S}}{\Gamma; \mathcal{D} \setminus X < \mathcal{S}} \setminus X$$

$\Gamma; \mathcal{D} < \mathcal{S}$	By sub-derivation
$\Gamma; \mathcal{D} < \mathcal{S}\|\mathcal{S}'$	By IH 3
$\Gamma; \mathcal{D} \setminus X < \mathcal{S}\|\mathcal{S}'$	By rule $\setminus X$

Case:

$$\gamma = \frac{\models_{\mathcal{S}} D \qquad \Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}}{\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}} \setminus \to$$

$\models_{\mathcal{S}} D$	By sub-derivation
$\models_{\mathcal{S}\|\mathcal{S}'} D$	By IH 1
$\Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}$	By sub-derivation
$\Gamma; (\mathcal{D} \wedge D) \setminus G < \mathcal{S}\|\mathcal{S}'$	By IH 2
$\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}\|\mathcal{S}'$	By rule $\setminus \to$

Case:

$$\sigma = \frac{}{\cdot; \top < \mathcal{S}} <_1$$

Then immediately:

$$\sigma' = \frac{}{\cdot; \top < \mathcal{S}\|\mathcal{S}'} <_1$$

Case:

$$\sigma = \frac{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \quad \models_{\mathcal{S}} \mathcal{D}' \quad (\Gamma; \mathcal{D}) < \mathcal{S}}{(\Gamma, \lceil \Gamma' \rceil); (\mathcal{D} \wedge \lceil \mathcal{D}' \rceil) < \mathcal{S}} <_2$$

$\Gamma; \mathcal{D} < \mathcal{S}$	By sub-derivation
$\Gamma; \mathcal{D} < \mathcal{S} \| \mathcal{S}'$	By IH 3
$\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S}$	By sub-derivation
$\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S} \| \mathcal{S}'$	By Lemma 6.10
$\models_{\mathcal{S}} \mathcal{D}'$	By sub-derivation
$\models_{\mathcal{S} \| \mathcal{S}'} D$	By IH 1
$(\Gamma, \lceil \Gamma' \rceil); (\mathcal{D} \wedge \lceil \mathcal{D}' \rceil) < \mathcal{S} \| \mathcal{S}'$	By rule $<_2$

□

We are now ready to show that if a program yields a schema, then every instance of a clause used at run-rime will comply with the schema. We need to generalize this statement to take into account instances of goals as well; in the latter case, we need to synchronize the schema accumulator in $\Gamma'; \mathcal{D}' \vdash G \stackrel{G}{\Longrightarrow} \mathcal{S}$ with the possibly incomplete run-time context $\Gamma; \mathcal{D}$. A fortiori the result holds for compile-time contexts as well, by taking $\Gamma; \mathcal{D}$ as $\cdot; \top$.

Theorem 6.12 (Extracted Schema Satisfaction) *Let* $\Gamma \vdash \theta : \Phi$.

1. *If* $\vdash_\Phi D \stackrel{D}{\Longrightarrow} \mathcal{S}$, *then* $\models_{\mathcal{S}} [\theta]D$.
2. *If* $\Gamma'; \mathcal{D}' \vdash_\Phi G \stackrel{G}{\Longrightarrow} \mathcal{S}$, $\models_{\mathcal{S}} \mathcal{D}'$ *and* $\Gamma; \mathcal{D} < \mathcal{S}$, *then* $(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G < \mathcal{S}$.

Proof: By mutual induction on the structure of $\pi :: D \stackrel{D}{\Longrightarrow} \mathcal{S}$ and $\gamma :: \Gamma'; \mathcal{D}' \vdash_\Phi G \stackrel{G}{\Longrightarrow} \mathcal{S}$.

Case:

$$\pi = \frac{X \in \{Q, \top, \bot\}}{\vdash_\Phi X \stackrel{D}{\Longrightarrow} \cdot; \top} \stackrel{D}{\Longrightarrow} X$$

$\models_{\cdot;\top} X$	By rule $\models_{\mathcal{S}} X$

Case:

$$\pi = \frac{\vdash_\Phi D_1 \stackrel{D}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D_2 \stackrel{D}{\Longrightarrow} \mathcal{S}_2}{\vdash_\Phi D_1 \wedge D_2 \stackrel{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \stackrel{D}{\Longrightarrow} \wedge$$

$\vdash_\Phi D_1 \stackrel{D}{\Longrightarrow} \mathcal{S}_1$	By sub-derivation
$\vdash_\Phi D_2 \stackrel{D}{\Longrightarrow} \mathcal{S}_2$	By sub-derivation
$\models_{\mathcal{S}_1} [\theta]D_1$	By IH 1
$\models_{\mathcal{S}_2} [\theta]D_2$	By IH 1
$\models_{\mathcal{S}_1 \| \mathcal{S}_2} [\theta]D_1$	By Lemma 6.11
$\models_{\mathcal{S}_1 \| \mathcal{S}_2} [\theta]D_2$	By Lemma 6.11
$\models_{\mathcal{S}_1 \| \mathcal{S}_2} [\theta]D_1 \wedge [\theta]D_2$	By rule $\models_{\mathcal{S}} \wedge$
$\models_{\mathcal{S}_1 \| \mathcal{S}_2} [\theta](D_1 \wedge D_2)$	By subst.

Case: π ends in $\stackrel{D}{\Longrightarrow} \vee$: analogously to the above.

Case: π ends in $\stackrel{D}{\Longrightarrow} \forall^y$: by an immediate appeal to IH 1.

Case:

$$\pi = \cfrac{\cdot;\top \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\vdash_\Phi (G \to D) \overset{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{D}{\Longrightarrow}\to$$

$\vdash_\Phi D \overset{D}{\Longrightarrow} \mathcal{S}_2$	By sub-derivation
$\models_{\mathcal{S}_2} [\theta]D$	By IH 1
$\models_{\mathcal{S}_1\|\mathcal{S}_2} [\theta]D$	By Lemma 6.11
$\cdot;\top \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1$	By sub-derivation
$\cdot;\top < \mathcal{S}_1$	By rule $<_1$
$\cdot;\top \setminus [\theta]G < \mathcal{S}_1$	By IH 2
$\cdot;\top \setminus [\theta]G < \mathcal{S}_1\|\mathcal{S}_2$	By Lemma 6.11
$\models_{\mathcal{S}_1\|\mathcal{S}_2} [\theta]G \to [\theta]D$	By rule $\models_{\mathcal{S}}\to$

Case:

$$\gamma = \cfrac{X \in \{Q, \top, \bot\} \text{ or (dis)eq}}{\Gamma';\mathcal{D}' \vdash_\Phi X \overset{G}{\Longrightarrow} \Gamma';\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}'} \overset{G}{\Longrightarrow} X$$

$\Gamma';[\theta]\mathcal{D}' \equiv_\alpha \Gamma';[\theta]\mathcal{D}'$	By hypothesis
$\Gamma \vdash \Gamma';[\theta]\mathcal{D}' \in \Gamma';\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}'$	By rule $\in_1$
$\Gamma;\mathcal{D} < \Gamma';\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}'$	By hypothesis
$\models_{\Gamma';(\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}')} \mathcal{D}'$	By hypothesis
$\models_{\Gamma';(\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}')} [\theta]\mathcal{D}'$	By Lemma 6.8
$(\Gamma, \lceil\Gamma'\rceil);(\mathcal{D} \wedge \lceil[\theta]\mathcal{D}'\rceil) < \Gamma';\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}'$	By rule $<_2$
$(\Gamma, \Gamma');(\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]X < \Gamma';\text{SOME } \Phi_{\mathcal{D}'}.\,\mathcal{D}'$	By rule $\setminus X$

Case:

$$\gamma = \cfrac{(\Gamma', y{:}A);\mathcal{D}' \vdash_\Phi [y/x]G \overset{G}{\Longrightarrow} \mathcal{S}}{\Gamma';\mathcal{D}' \vdash_\Phi \forall x{:}A.\,G \overset{G}{\Longrightarrow} \mathcal{S}} \overset{G}{\Longrightarrow} \forall^y$$

$(\Gamma', y{:}A);\mathcal{D}' \vdash_\Phi [y/x]G \overset{G}{\Longrightarrow} \mathcal{S}$	By sub-derivation
$(\Gamma, \Gamma', y{:}A);(\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]\cdot[y/x]G < \mathcal{S}$	By IH 2
$(\Gamma, \Gamma');(\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]\forall x{:}A.\,G < \mathcal{S}$	By rule $\setminus\forall^y$

Case:

$$\gamma = \cfrac{\Gamma';(\mathcal{D}' \wedge D) \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \vdash_\Phi D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\Gamma';\mathcal{D}' \vdash_\Phi D \to G \overset{G}{\Longrightarrow} \mathcal{S}_1\|\mathcal{S}_2} \overset{G}{\Longrightarrow}\to$$

$\vdash_\Phi D \overset{D}{\Longrightarrow} \mathcal{S}_2$	By sub-derivation
$\models_{\mathcal{S}_2} [\theta]D$	By IH 1
$\models_{\mathcal{S}_1\|\mathcal{S}_2} [\theta]D$	By Lemma 6.11
$\Gamma;\mathcal{D} < \mathcal{S}_1$	By hypothesis
$\Gamma';(\mathcal{D}' \wedge D) \vdash_\Phi G \overset{G}{\Longrightarrow} \mathcal{S}_1$	By sub-derivation
$(\Gamma, \Gamma');(\mathcal{D} \wedge [\theta]\mathcal{D}' \wedge [\theta]D) \setminus [\theta]G < \mathcal{S}_1$	By IH 2
$(\Gamma, \Gamma');(\mathcal{D} \wedge [\theta]\mathcal{D}' \wedge [\theta]D) \setminus [\theta]G < \mathcal{S}_1\|\mathcal{S}_2$	By Lemma 6.11
$(\Gamma, \Gamma');(\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]D \to [\theta]G < \mathcal{S}_1\|\mathcal{S}_2$	By rule $\setminus\to$

Case:

$$\gamma = \cfrac{\Gamma';\mathcal{D}' \vdash_\Phi G_1 \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \Gamma';\mathcal{D}' \vdash_\Phi G_2 \overset{G}{\Longrightarrow} \mathcal{S}_2}{\Gamma';\mathcal{D}' \vdash_\Phi G_1 \wedge G_2 \overset{G}{\Longrightarrow} \mathcal{S}_1\|\mathcal{S}_2} \overset{G}{\Longrightarrow}\wedge$$

$\Gamma'; \mathcal{D}' \vdash_\Phi G_1 \overset{G}{\Longrightarrow} \mathcal{S}_1$ By sub-derivation
$\Gamma; \mathcal{D} < \mathcal{S}_1$ By hypothesis
$(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G_1 < \mathcal{S}_1$ By IH 2
$(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G_1 < \mathcal{S}_1 \| \mathcal{S}_2$ By Lemma 6.11
$\Gamma'; \mathcal{D}' \vdash_\Phi G_2 \overset{G}{\Longrightarrow} \mathcal{S}_2$ By sub-derivation
$\Gamma; \mathcal{D} < \mathcal{S}_2$ By hypothesis
$(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G_2 < \mathcal{S}_2$ By IH 2
$(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G_2 < \mathcal{S}_1 \| \mathcal{S}_2$ By Lemma 6.11
$(\Gamma, \Gamma'); (\mathcal{D} \wedge [\theta]\mathcal{D}') \setminus [\theta]G_1 \wedge [\theta]G_2 < \mathcal{S}_1 \| \mathcal{S}_2$ By rule $\setminus\wedge$

Case: γ ends in $\overset{G}{\Longrightarrow} \vee$: similarly to the above. to IH 2.

□

6.2.2 Context Preservation

We aim to prove that execution preserves contexts, provided that the program itself and the input goal satisfy a schema; that is, that every subgoal which arises in any given successful or failed (immediate and non-immediate sub-derivation) satisfies the same context schema.

Note that we have already established schema extraction (Theorem 6.12), that is we assume that $\mathcal{P} \overset{D}{\Longrightarrow} \mathcal{S}$. We write $\pi' < \pi$ to say that π' is a strict subproof of π, i.e. $\pi' \leq \pi$ if $\pi' < \pi$ or $\pi' = \pi$.

Theorem 6.13 (Context Preservation w.r.t. Provability) *Let $\models_\mathcal{S} \mathcal{P}$ and $D \sqsubseteq (\mathcal{P} \wedge \mathcal{D})$. For any $\Gamma; \mathcal{D} < \mathcal{S}$, if ($\pi :: \Gamma; \mathcal{D} \vdash_\mathcal{P} G$ and $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$) or ($\iota :: \Gamma; \mathcal{D} \vdash_\mathcal{P} D \gg Q$ and $\models_\mathcal{S} D$), then:*

1. *for every subproof $\pi' :: \Gamma'; \mathcal{D}' \vdash_\mathcal{P} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$.*
2. *for every subproof $\iota' :: \Gamma'; \mathcal{D}' \vdash_\mathcal{P} D' \gg Q$ and $\models_\mathcal{S} D'$.*

Proof: By mutual induction on the structure of $\pi :: \Gamma; \mathcal{D} \vdash_\mathcal{P} G$ and $\iota :: \Gamma; \mathcal{D} \vdash_\mathcal{P} D \gg Q$.

Case: $\pi = \Gamma; \mathcal{D} \vdash_\mathcal{P} \top$: trivial.

Case:

$$\pi = \cfrac{\begin{matrix}\pi_1 \\ \Gamma; \mathcal{D} \wedge D \vdash_\mathcal{P} G\end{matrix}}{\Gamma; \mathcal{D} \vdash_\mathcal{P} D \to G} \vdash\to$$

Subcase: $\pi' = \pi$: trivial.

Subcase: $\pi' < \pi$: then $\pi' \leq \pi_1$.

$\pi_1 :: \Gamma; \mathcal{D} \wedge D \vdash_\mathcal{P} G$ By sub-derivation
$\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}$ By hypothesis
$\Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}$ By inversion
$\iota' :: \Gamma'; \mathcal{D}' \vdash_\mathcal{P} D' \gg Q$ and $\models_\mathcal{S} D'$ and
$\pi'_1 :: \Gamma'; (\mathcal{D}' \wedge D') \vdash_\mathcal{P} G'$ and $\Gamma'; (\mathcal{D}' \wedge D') \setminus G' < \mathcal{S}$ By IH
$\pi' :: \Gamma'; \mathcal{D}' \vdash_\mathcal{P} D' \to G'$ and $\Gamma'; \mathcal{D}' \setminus D' \to G' < \mathcal{S}$ By rule

Case:

$$\pi \cfrac{\begin{matrix}\pi_1 \\ (\Gamma, y{:}A); \mathcal{D} \vdash_\mathcal{P} [y/x]G\end{matrix}}{\Gamma; \mathcal{D} \vdash_\mathcal{P} \forall x{:}A.\, G} \vdash \forall^y$$

Subcase: $\pi' = \pi$: trivial.

Subcase: $\pi' < \pi$: then $\pi' \leq \pi_1$.

$\pi_1 :: (\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G$ — By sub-derivation
$\Gamma; \mathcal{D} \setminus \forall x{:}A.\, G < \mathcal{S}$ — By hypothesis
$(\Gamma, y{:}A); \mathcal{D} \setminus [y/x]G < \mathcal{S}$ — By inversion
$\iota' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$ and
$\pi_1' :: (\Gamma', y{:}A); \mathcal{D}' \vdash_{\mathcal{P}} [y/x]G'$ and $(\Gamma', y{:}A); \mathcal{D}' \setminus [y/x]G' < \mathcal{S}$ — By IH
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} \forall x{:}A.\, G'$ and $\Gamma'; \mathcal{D}' \setminus \forall x{:}A.\, G' < \mathcal{S}$ — By rule

Case: π ends in $\vdash \wedge$, $\vdash \vee_1$, $\vdash \vee_2$: similarly.

Case:

$$\pi = \cfrac{\overset{\iota}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q}}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} Q} \vdash \mathrm{At}$$

Subcase: $\pi' = \pi$: trivial.

Subcase: $\pi' < \pi$: then $\pi' \leq \iota$.

Subcase: $\iota_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$, for $D \sqsubseteq \mathcal{P}$ — By sub-derivation
$\models_{\mathcal{S}} D$ — By hypothesis
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and
$\iota_1' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$ — By IH
$\Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} Q'$ and $\Gamma'; \mathcal{D}' \setminus Q' < \mathcal{S}$ — By rule

Subcase: $\iota_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$, for $D \sqsubseteq \mathcal{D}$ — By sub-derivation
$\models_{\mathcal{S}} D$ — By Lemma 6.7
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and
$\iota_1' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$ — By IH
$\Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} Q'$ and $\Gamma'; \mathcal{D}' \setminus Q' < \mathcal{S}$ — By rule

Case: $\iota :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bot \gg Q$: trivial.

Case:

$$\iota = \cfrac{\overset{\iota_1}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q} \gg\wedge_1$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$: then $\iota' \leq \iota_1$:

$\models_{\mathcal{S}} D_1 \wedge D_2$ — By hypothesis
$\models_{\mathcal{S}} D_1$ and $\models_{\mathcal{S}} D_2$ — By inversion
$\iota_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — By sub-derivation
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and
$\iota_1' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D_1' \gg Q$ and $\models_{\mathcal{S}} D_1'$ — By IH
$\iota' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D_1' \wedge D_2' \gg Q$ and $\models_{\mathcal{S}} D_1' \wedge D_2'$ — By rule

Case: ι ends in $\gg\wedge_2$. Symmetrical.

Case:

$$\iota = \cfrac{\overset{\iota_1}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D \gg Q}}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall x{:}A.\, D \gg Q} \gg\forall$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$: then $\iota' \leq \iota_1$.

$\iota_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D \gg Q$	By sub-derivation
$\models_{\mathcal{S}} \forall x{:}A.\, D$	By hypothesis
$\models_{\mathcal{S}} [u/x]D$	By inversion
$\models_{\mathcal{S}} [t/u, u/x]D$	By substitution
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and	
$\iota_1' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} [t/x]D' \gg Q$ and $\models_{\mathcal{S}} [t/x]D'$	By IH
$\iota' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} \forall x{:}A.\, D' \gg Q$ and $\models_{\mathcal{S}} \forall x{:}A.\, D'$	By rule

Case:

$$\iota = \dfrac{\overset{\iota_1}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q} \qquad \overset{\pi_1}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G}}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G \to D \gg Q} \gg\to$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$:

Subcase: $\iota' \leq \iota_1$

$\iota_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\pi_1 :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} G$	By sub-derivation
$\models_{\mathcal{S}} G \to D$	By hypothesis
$\Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\models_{\mathcal{S}} D$	By inversion
$\pi' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and	
$\iota_1' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$	By IH
$\iota' :: \Gamma'; \mathcal{D}' \vdash_{\mathcal{P}} G' \to D' \gg Q$ and $\models_{\mathcal{S}} G' \to D'$	By rule

Subcase: $\iota' \leq \pi_1$: analogously.

Case: ι ends in $\gg\vee, \gg\forall$: similarly.

□

We establish an analogous result w.r.t. denial.

Theorem 6.14 (Context Preservation w.r.t. Denial) *Let $\models_{\mathcal{S}} \mathcal{P}$ and $D \sqsubseteq (\mathcal{P} \wedge \mathcal{D})$: for any $\Gamma; \mathcal{D} < \mathcal{S}$, if ($\pi :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G$ and $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$) or ($\iota :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$ and $\models_{\mathcal{S}} D$), then:*

1. *for every subproof $\pi' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$.*
2. *for every subproof $\iota' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$.*

Proof: By mutual induction on the structure of $\pi :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G$ and $\iota :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$, similarly to the case of provability. We show only some cases:

Case:

$$\pi = \dfrac{\overset{\pi_1}{\Gamma; (\mathcal{D} \wedge D) \not\vdash_{\mathcal{P}} G}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \to G} \not\vdash\to$$

Subcase: $\pi' = \pi$: trivial.

Subcase: $\pi' < \pi$: then $\pi' \leq \pi_1$:

$\Gamma; (\mathcal{D} \wedge D) \not\vdash_{\mathcal{P}} G$	By sub-derivation
$\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}$	By hypothesis
$\Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}$	By inversion
$\iota' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$ and	
$\pi' :: \Gamma; (\mathcal{D}' \wedge D') \not\vdash_{\mathcal{P}} G'$ and $\Gamma; (\mathcal{D}' \wedge D') \setminus G' < \mathcal{S}$	By IH
$\pi'' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} (D' \to G')$ and $\Gamma'; \mathcal{D}' \setminus (D' \to G') < \mathcal{S}$	By rule

Case:

$$\iota = \cfrac{\overset{\iota_1}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \leftarrow G \gg Q} \not\gg\to_1$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$: then $\iota' \leq \iota_1$:

$\iota_1 :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G$	By sub-derivation
$\models_{\mathcal{S}} G \to D$	By hypothesis
$\Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\models_{\mathcal{S}} D$	By inversion
$\pi' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and	
$\iota' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$	By IH
$\iota'' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} (D' \leftarrow G') \gg Q$ and $\models_{\mathcal{S}} (D' \leftarrow G')$	By rule

Case:

$$\iota = \cfrac{\overset{\iota_1}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \leftarrow G \gg Q} \not\gg\to_2$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$: then $\iota' \leq \iota_1$:

$\iota_1 :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$	By sub-derivation
$\models_{\mathcal{S}} G \to D$	By hypothesis
$\Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\models_{\mathcal{S}} D$	By inversion
$\pi' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and	
$\iota' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} D' \gg Q$ and $\models_{\mathcal{S}} D'$	By IH
$\iota'' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} (D' \leftarrow G') \gg Q$ and $\models_{\mathcal{S}} (D' \leftarrow G')$	By rule

Case:

$$\iota = \cfrac{\overset{\iota_1}{\text{for all } n \ \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [n/x]D \gg Q}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall x{:}A.\, D \gg Q} \gg\forall$$

Subcase: $\iota' = \iota$: trivial.

Subcase: $\iota' < \iota$: then $\iota' \leq \iota_1$.

$\models_{\mathcal{S}} \forall x{:}A.\, D$	By hypothesis
$\models_{\mathcal{S}} [u/x]D$	By inversion
For all $\iota_1 :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [n/x]D \gg Q$	By sub-derivation
For all $\models_{\mathcal{S}} [n/u, u/x]D$	By substitution
$\pi' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} G'$ and $\Gamma'; \mathcal{D}' \setminus G' < \mathcal{S}$ and	
For all $\iota_1' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} [n/x]D' \gg Q$ and $\models_{\mathcal{S}} [n/x]D'$	By IH
$\iota' :: \Gamma'; \mathcal{D}' \not\vdash_{\mathcal{P}} \forall x{:}A.\, D' \gg Q$ and $\models_{\mathcal{S}} \forall x{:}A.\, D'$	By rule

□

6.3 Terminating Programs

We now introduce terminating programs. The notion of termination that we adopt is the very strong *universal* one, as it is known in logic programming [SD94]. This is abstractly achieved by means of a relation between goals or between goals and clause heads, namely '$\prec$', with the intended meaning of "G_1 $[G]$ can arise as a subgoal of G_2 $[D]$". Intuitively, a program terminates if it yields an ordering relation which admits no infinite descending chains. This is defined in Figure 6.6. We do not commit here to actual ways to verify the latter property. This issue is explored, in the higher-order setting, for example in [RP96]. More formally:

$$\mathit{Ordering} \quad R \quad ::= \quad \cdot \mid R, G \prec D \mid R, G_1 \prec G_2$$

Let $\mathcal{P} \overset{D}{\Longrightarrow} R$ and take the stable (that is, closed under substitutions) and transitive closure of R, call it $[R]$. We say that $\mathcal{P}$ is *terminating*, denoted $\mathcal{P}\downarrow$, whenever $[R]$ is well-founded.

Example 6.15 *If* `even` *defines the even number,* `even` $\overset{D}{\Longrightarrow}$ $even(X) \prec even(s(s(X)))$*. Furthermore the closure of this relation is well-founded as it can be proven say by induction on* X*. Consider instead the following:*

$$\begin{array}{l} \forall E\!:\!exp.\forall M\!:\!nat. \\ p\ (lam\ E)\ M \\ \quad \leftarrow (\forall x\!:\!exp.\,(\forall N\!:\!nat.\,p\ x\ (s\ N) \leftarrow p\ x\ (s\ N)) \rightarrow p\ (E\ x)\ M). \end{array}$$

The clause is not terminating as the subgoal relation contains the pair $[y/x](p\ x\ s(N) \prec p\ x\ s(N))$*, which yields an infinite descending chain* $(p\ y\ (s\ 0)) \prec (p\ y\ (s\ 0)) \prec \ldots$*.*

Finally , we define when a *schema* is terminating:

$$\frac{}{\circ\downarrow}\circ\downarrow \qquad \frac{\Gamma \vdash \mathcal{D}\downarrow \qquad \mathcal{S}\downarrow}{\mathcal{S}\|\Gamma;\mathrm{SOME}\ \Phi.\,\mathcal{D}\downarrow}\|\downarrow$$

Since Φ ranges over $FV(\mathcal{D}) \setminus \Gamma$, we verify the termination of $\mathcal{D}$ in the Γ context.

Lemma 6.16

1. *If* $\Gamma \vdash \mathcal{D} \overset{D}{\Longrightarrow} R$*,* $D \sqsubseteq \mathcal{D}$ *and* $\Gamma \vdash D \overset{D}{\Longrightarrow} R'$*, then* $R' \subseteq R$*.*
2. *If* $\Gamma \vdash G \overset{G}{\Longrightarrow} R$*,* $G' \prec G$ *and* $\Gamma \vdash G' \overset{G}{\Longrightarrow} R'$*, then* $R' \subseteq R$*.*

Proof: By a straightforward mutual induction on the structure of $\Gamma \vdash D \overset{D}{\Longrightarrow} R'$ and $\Gamma \vdash G' \overset{G}{\Longrightarrow} R'$. □

Corollary 6.17 *If* $\Gamma \vdash \mathcal{D}\downarrow$ *and* $D \sqsubseteq \mathcal{D}$*, then* $\Gamma \vdash D\downarrow$*.*

Proof: By definition, $\Gamma \vdash \mathcal{D}\downarrow$ if $\Gamma \vdash \mathcal{D} \overset{D}{\Longrightarrow} R$ and $[R]$ is well-founded; by Lemma 6.16 $\Gamma \vdash D \overset{D}{\Longrightarrow} R'$, and $R' \subseteq R$. Thus $[R']$ is well-founded, and $D\downarrow$. □

Lemma 6.18 *Let* $\mathcal{S}\downarrow$*. If* $\Gamma;\mathcal{D} < \mathcal{S}$ *and* $D \sqsubseteq \mathcal{D}$*, then* $D\downarrow$*.*

Proof: By induction on the structure of $\mathcal{S}\downarrow$, using Corollary 6.17. □

We can now prove that if a program is terminating, a non-proof of any ground G is a *denial* of G; we are going to reason classically that either there is a proof of G, or there is not such a proof. We recall that $\not\vdash_{\mathcal{P}}$ is the denial relation introduced in Figure 6.1 and 6.2.

Theorem 6.19 (Termination) *Let* $\mathcal{P}\downarrow$ *and* $\mathcal{S}\downarrow$ *be a schema such that* $\models_{\mathcal{S}} \mathcal{P}$ *and* $\Gamma;\mathcal{D} < \mathcal{S}$*: for any ground* G*, if not* $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} G$*, then* $\Gamma;\mathcal{D} \not\vdash_{\mathcal{P}} G$*.*

Proof: We generalize this to:

1. If not $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} G$, then $\Gamma;\mathcal{D} \not\vdash_{\mathcal{P}} G$.
2. For every $D \sqsubseteq \mathcal{P} \wedge \mathcal{D}$, if not $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D \gg Q$, then $\Gamma;\mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$.

We proceed by mutual induction on the goal ordering induced by $\mathcal{P}$ terminating and on the structure of D: we start with 2. Since D is an instance of a terminating clause either from the program or the run-time context, by stability and Corollary 6.18, it is terminating as well.

$$\frac{X \in \{Q, \top, \bot\} \text{ or (dis)eq}}{\Gamma \vdash X \overset{G}{\Longrightarrow} \cdot} \overset{G}{\Longrightarrow} X$$

$$\frac{\Gamma \vdash G \overset{G}{\Longrightarrow} R_1 \qquad \Gamma \vdash D \overset{D}{\Longrightarrow} R_2}{\Gamma \vdash D \to G \overset{G}{\Longrightarrow} R_1, R_2, G \prec D \to G} \overset{G}{\Longrightarrow}\to$$

$$\frac{\Gamma, y{:}A \vdash G \overset{G}{\Longrightarrow} R}{\Gamma \vdash \forall x{:}A.\, G \overset{G}{\Longrightarrow} R, [y/x]G \prec \forall x{:}A.\, G} \overset{G}{\Longrightarrow} \forall^y$$

$$\frac{\Gamma \vdash G_1 \overset{G}{\Longrightarrow} R_1 \qquad \Gamma \vdash G_2 \overset{G}{\Longrightarrow} R_2}{\Gamma \vdash G_1 \wedge G_2 \overset{G}{\Longrightarrow} R_1, R_2, G_1 \prec G_1 \wedge G_2, G_2 \prec G_1 \wedge G_2} \overset{G}{\Longrightarrow} \wedge$$

$$\frac{\Gamma \vdash G_1 \overset{G}{\Longrightarrow} R_1 \qquad \Gamma \vdash G_2 \overset{G}{\Longrightarrow} R_2}{\Gamma \vdash G_1 \vee G_2 \overset{G}{\Longrightarrow} R_1, R_2, G_1 \prec G_1 \vee G_2, G_2 \prec G_1 \vee G_2,} \overset{G}{\Longrightarrow} \vee$$

$$\frac{X \in \{Q, \top, \bot\}}{\Gamma \vdash X \overset{D}{\Longrightarrow} \cdot} \overset{D}{\Longrightarrow} X$$

$$\frac{\Gamma \vdash D \overset{D}{\Longrightarrow} R_1 \qquad \Gamma \vdash G \overset{G}{\Longrightarrow} R_2}{\Gamma \vdash (G \to D) \overset{D}{\Longrightarrow} R_1, R_2, G \prec D} \overset{D}{\Longrightarrow}\to$$

$$\frac{\Gamma \vdash [u/x]D \overset{D}{\Longrightarrow} R}{\Gamma \vdash \forall x{:}A.\, D \overset{D}{\Longrightarrow} R} \overset{D}{\Longrightarrow} \forall^u$$

$$\frac{\Gamma \vdash D_1 \overset{D}{\Longrightarrow} R_1 \qquad \Gamma \vdash D_2 \overset{D}{\Longrightarrow} R_2}{\Gamma \vdash D_1 \wedge D_2 \overset{D}{\Longrightarrow} R_1, R_2} \overset{D}{\Longrightarrow} \wedge$$

$$\frac{\Gamma \vdash D_1 \overset{D}{\Longrightarrow} R_1 \qquad \Gamma \vdash D_2 \overset{D}{\Longrightarrow} R_2}{\Gamma \vdash D_1 \vee D_2 \overset{D}{\Longrightarrow} R_1, R_2} \overset{D}{\Longrightarrow} \vee$$

Figure 6.6: Generation of the subgoal relation

Case: $D = \bot$: trivial.

Case: $D = \top$.

$$\frac{}{\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} \top \gg Q} \gg\top$$

Case: $D = D_1 \leftarrow G_1$. Either $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ or not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ or not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$. By termination of $D_1 \leftarrow G_1$, $G_1 \prec D_1$, hence we can apply the IH to G_1: There are four sub-cases:

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \leftarrow G_1 \gg Q$ — By rule $\gg \rightarrow$
2. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} G_1$ — By IH 1
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \leftarrow G_1 \gg Q$ — By rule $\gg \rightarrow_2$
3. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \leftarrow G_1 \gg Q$ — By rule $\gg \rightarrow_1$
4. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \leftarrow G_1 \gg Q$By rule — By rule $\gg \rightarrow_1$

Case: $D = D_1 \vee D_2$.

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \vee D_2 \gg Q$ — By rule $\gg\vee$
2. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_2 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \vee D_2 \gg Q$ — By rule
3. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   Not $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \vee D_2 \gg Q$ — By rule
4. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_2 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_1 \vee D_2 \gg Q$ — By rule

Case: $D = D_1 \wedge D_2$.

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q$ — By rule $\gg\wedge$
2. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D_2 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q$ — By rule

3. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q$ — By rule
4. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q$ — By IH 2
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_2 \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \wedge D_2 \gg Q$ — By rule

Case: $D = \forall x : A. D'$. For some ground $\Gamma \vdash t : A$, $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D' \gg Q$ or for all ground t, not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D' \gg Q$:

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D' \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall x{:}A. D' \gg Q$ — By rule $\gg\forall$
2. For all t ground not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [t/x]D' \gg Q$ — Subcase
   For all t ground $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [t/x]D' \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall x{:}A. D' \gg Q$ — By rule

Case: $G = \top$: trivially true.

Case: $G = \bot$:

$$\frac{}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \bot} \not\vdash \bot$$

Case: $G = Q$: either $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ or for every D not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$:

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} Q$ — By rule
2. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ for every D — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$ — By IH 2
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} Q$ — By rule

Case: $G = G_1 \wedge G_2$:

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$ — Subcase
   $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \wedge G_2$ — By rule
2. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_2$ — By IH 1
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1 \wedge G_2$ — By rule
3. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$ — Subcase
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1$ — By IH 1
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1 \wedge G_2$ — By rule
4. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1$ — By IH 1
   Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$ — Subcase
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_2$ — By IH 1
   $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1 \wedge G_2$ — By rule

Case: $G = G_1 \vee G_2$:

1. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$	Subcase
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$	Subcase
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \vee G_2$	By rule
2. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$	Subcase
Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$	Subcase
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_2$	By IH 1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2 \vee G_1$	By rule
3. $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$	Subcase
Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$	Subcase
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1$	By IH 1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2 \vee G_1$	By rule
4. Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$	Subcase
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1$	By IH 1
Not $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$	Subcase
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_2$	By IH 1
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G_1 \vee G_2$	By rule

Case: $G = \forall x{:}A.\, G'$: for a new parameter y, $(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$ or not $(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$:

1. $(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$	Subcase
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall x{:}A.\, G'$	By rule
2. Not $(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$	Subcase
$(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$	By IH 1
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall x{:}A.\, G'$	By rule

Case: $G = D' \rightarrow G'$:

1. $\Gamma; (\mathcal{D} \wedge D') \vdash_{\mathcal{P}} G'$	Subcase
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D' \rightarrow G'$	By rule
2. Not $\Gamma; (\mathcal{D} \wedge D') \vdash_{\mathcal{P}} G'$	Subcase
$\Gamma; (\mathcal{D} \wedge D') \not\vdash_{\mathcal{P}} G'$	By IH 1
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D' \rightarrow G'$	By rule

Case: The (dis)equality case follows immediately from the decidability of the $\doteq, \not\doteq$ rules.

□

6.4 Complementable Clauses

We restrict ourselves to programs with:

- *Complementable* clauses as defined in Figure 6.7.
- Rules of the form $\forall(Q \leftarrow G)$, such that every (input) term in Q is rigid.
- Parameters of base type and occurring only in *head* position, called *Shallow Parameter Expressions*:

$$SPE \quad e_x \quad ::= \quad x : a \mid \lambda x \,.\, e_x$$

We do not formalize here the former assumptions on terms; note that the rigidity restriction applies only to predicates mutually recursive to non-Horn ones – see the comment at the end of the proof of Theorem 6.32. We do give a definition of complementable clause in Figure 6.7.

$$\frac{X \in \{\top, \bot\} \text{ or (dis)eq}}{\Gamma; \mathcal{D} \vdash X \text{ compl}}\, cg\ X$$

$$\frac{}{\Gamma; \top \vdash Q \text{ compl}}\, cg\text{At}$$

$$\frac{\text{for all } D \sqsubseteq \mathcal{D} : \text{dom}(\Gamma) \cap par(D) \neq \emptyset}{\Gamma; \mathcal{D} \vdash Q \text{ compl}}\, cg\text{At}\cap$$

$$\frac{\Gamma; (\mathcal{D} \wedge D) \vdash G \text{ compl} \qquad \Gamma; \mathcal{D} \vdash D \text{ compl}}{\Gamma; \mathcal{D} \vdash D \to G \text{ compl}}\, cg \to$$

$$\frac{(\Gamma, y{:}a); \mathcal{D} \vdash [y/x]G \text{ compl}}{\Gamma; \mathcal{D} \vdash \forall x{:}a.\, G \text{ compl}}\, cg\forall^y$$

$$\frac{\Gamma; \mathcal{D} \vdash G_1 \text{ compl} \qquad \Gamma; \mathcal{D} \vdash G_2 \text{ compl}}{\Gamma; \mathcal{D} \vdash G_1 \wedge G_2 \text{ compl}}\, cg\wedge$$

$$\frac{\Gamma; \mathcal{D} \vdash G_1 \text{ compl} \qquad \Gamma; \mathcal{D} \vdash G_2 \text{ compl}}{\Gamma; \mathcal{D} \vdash G_1 \vee G_2 \text{ compl}}\, cg\vee$$

$$\frac{X \in \{\top, \bot\}}{\Gamma; \mathcal{D} \vdash X \text{ compl}}\, cd\ X$$

$$\frac{\Gamma; \mathcal{D} \vdash D \text{ compl} \qquad \Gamma; \mathcal{D} \vdash G \text{ compl}}{\Gamma; \mathcal{D} \vdash D \leftarrow G \text{ compl}}\, cd \leftarrow$$

$$\frac{\Gamma; \mathcal{D} \vdash [u/x]D \text{ compl}}{\Gamma; \mathcal{D} \vdash \forall x{:}A.\, D \text{ compl}}\, cd\forall^u$$

$$\frac{\Gamma; \mathcal{D} \vdash D_1 \text{ compl} \qquad \Gamma; \mathcal{D} \vdash D_2 \text{ compl}}{\Gamma; \mathcal{D} \vdash D_1 \wedge D_2 \text{ compl}}\, cd\wedge$$

$$\frac{\Gamma; \mathcal{D} \vdash D_1 \text{ compl} \qquad \Gamma; \mathcal{D} \vdash D_2 \text{ compl}}{\Gamma; \mathcal{D} \vdash D_1 \vee D_2 \text{ compl}}\, cd\vee$$

Figure 6.7: Complementable clause and goal: $\Gamma; \mathcal{D} \vdash D$ compl and $\Gamma; \mathcal{D} \vdash G$ compl

Example 6.20 *The clause encoding the introduction rule for implication in natural deduction from Example 5.1 is* not *complementable:*

$$impi \quad : \quad nd(A\ imp\ B) \leftarrow (nd(A) \rightarrow nd(B)).$$

On the other hand, the following is allowed by rule cgAt:

$$\begin{array}{lll} oplam^{\top} & : & \forall E{:}exp \rightarrow exp.\, open\ (lam\ E) \\ & & \quad \leftarrow (\forall x{:}exp.\, \top_{open} \rightarrow open\ (E\ x)). \end{array}$$

The restriction on goals and clauses yields this revised grammar:

$$\begin{array}{llll} \textit{Clauses} & D & ::= & \top \mid \bot \mid \forall(Q \leftarrow G) \mid D_1 \wedge D_2 \mid D_1 \vee D_2 \\ \textit{Goals} & G & ::= & Q \mid \top \mid \bot \mid \vec{M} \doteq \vec{N} \mid \vec{M} \not\doteq \vec{N} \mid \\ & & & G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \rightarrow G \mid \forall x{:}a.\, G \end{array}$$

We use $\forall(Q \leftarrow G)$ as a normal form for 'rules', where the quantifier bounds every free variable in $Q \leftarrow G$. This has the technical advantage to provide a handle for both the head of a clause and the set of all its free variables at the same time, which will be crucial to describe the complement algorithm. In particular, 'facts' are represented by $\forall(Q) \leftarrow \top$, although in examples we will omit to mention the body. We accordingly specialize the immediate implication and denial rules as follows:

$$\frac{\Gamma \vdash \sigma : FV(\vec{N}) \setminus \mathrm{dom}(\Gamma) \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\vec{N} \doteq \vec{M} \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \vec{N} \leftarrow G) \gg q\ \vec{M}} \gg\rightarrow$$

$$\frac{\text{for all } \theta\ \Gamma \vdash \theta : FV(\vec{N}) \setminus \mathrm{dom}(\Gamma) \qquad \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\theta]\vec{N} \doteq \vec{M}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall(q\ \vec{N} \leftarrow G) \gg q\ \vec{M}} \not\gg\rightarrow_1$$

$$\frac{\Gamma \vdash \sigma : FV(\vec{N}) \setminus \mathrm{dom}(\Gamma) \qquad \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]\vec{N} \not\doteq \vec{M} \qquad \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall(q\ \vec{N} \leftarrow G) \gg q\ \vec{M}} \not\gg\rightarrow_2$$

We remark that the 'guards' in the immediate denial rules are constructed so as to make the positive and negative judgment mutually independent. Thus $\not\gg\rightarrow_1$ says that a clause $\forall(q\ \vec{N} \leftarrow G)$ denies a goal $q\ \vec{M}$ if for all well-typed substitutions θ $[\theta]\vec{N}$ and $\vec{M}$ clashes. Rule $\not\gg\rightarrow_2$ instead attributes denial with the same conclusion due to 'failure' in the body.

Clause complementation is taken *definition-wise*. We cluster a program in a conjunction of possibly mutually recursive predicate definitions. We accomplish this with the following:

Definition 6.21 ($\Gamma \vdash \mathbf{def}(q, D)$)

$$\begin{array}{rcll} \Gamma \vdash def(q, \top) & = & \top_q & \\ \Gamma \vdash def(q, \bot) & = & \bot_q & \\ \Gamma \vdash def(q, \forall(Q \leftarrow G)) & = & \forall(Q \leftarrow G) & \textit{if } Q \equiv q\ \vec{M} \\ \Gamma \vdash def(q, \forall(Q \leftarrow G)) & = & \top_q & \textit{otherwise} \\ \Gamma \vdash def(q, D_1 \wedge D_2) & = & \Gamma \vdash def(q, D_1) \wedge \Gamma \vdash def(q, D_2) & \\ \Gamma \vdash def(q, D_1 \vee D_2) & = & \Gamma \vdash def(q, D_1) \vee \Gamma \vdash def(q, D_2) & \end{array}$$

If $D = \mathcal{P}$, i.e. the underlying program (seen as a conjunction of clauses), we call $\cdot \vdash \mathrm{def}(q, \mathcal{P})$ the *static* definition of q and if $\Gamma; \mathcal{D}$ is a conjunction of a run-time context, we call the latter the *dynamic* definition of q. We avoid mentioning the parameter context when it can be inferred from the context. As a special case, an undefined predicate q, i.e. a predicate which occurs in a body of a clause but not as a head is represented by the empty conjunction $\top_q$. Conversely $\bot_q$ denotes the universal definition for q. We agree to consider $\top_q$ ($\bot_q$ resp.) as the zero (one) element for conjunction and disjunction and implicitly apply the appropriate absorption operations; for example:

$$\top_q \wedge \forall(q\ \vec{M} \leftarrow G) = \forall(q\ \vec{M} \leftarrow G)$$

We overload the notation $D \sqsubseteq \mathcal{D}$ in the case $D \sqsubseteq \Gamma \vdash \mathrm{def}(q, \mathcal{P} \wedge \mathcal{D})$ to indicate the relation that satisfies the following rules:

$$\frac{}{D \sqsubseteq D} \qquad \frac{D_1 \sqsubseteq \mathcal{D} \qquad D_2 \sqsubseteq \mathcal{D}}{D_1 \wedge D_2 \sqsubseteq \mathcal{D}} \qquad \frac{D_1 \sqsubseteq \mathcal{D} \qquad D_2 \sqsubseteq \mathcal{D}}{D_1 \vee D_2 \sqsubseteq \mathcal{D}}$$

The immediate implication and denial judgments is specialized as obvious:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$	Clause D from $\Gamma \vdash \mathrm{def}(q, \mathcal{P} \wedge \mathcal{D})$ immediately entails atom Q.
$\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} D \gg Q$	Clause D from $\Gamma \vdash \mathrm{def}(q, \mathcal{P} \wedge \mathcal{D})$ immediately denies atom Q.

In particular, the 'At' rules are:

$$\frac{D \sqsubseteq \mathrm{def}(Q, \mathcal{P} \wedge \mathcal{D}) \qquad \lceil\Gamma\rceil; \lceil\mathcal{D}\rceil \vdash_{\mathcal{P}} D \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} Q} \vdash \mathrm{At}$$

$$\frac{D \sqsubseteq \mathrm{def}(Q, \mathcal{P} \wedge \mathcal{D}) \qquad \lceil\Gamma\rceil; \lceil\mathcal{D}\rceil \nvdash_{\mathcal{P}} D \gg Q}{\Gamma; \mathcal{D} \nvdash_{\mathcal{P}} Q} \nvdash \mathrm{At}$$

Accordingly, we also specialized the rule for schema satisfaction: we use ρ to denote a *global* substitution, e.g. such that $\mathrm{dom}(\rho)$ is a set of new global parameters.

$$\frac{\cdot; \top \setminus [\rho]G < \mathcal{S}}{\models_{\mathcal{S}} \forall(Q \leftarrow G) \gg q\ \vec{M}} \models_{\mathcal{S}} \to^{\rho}$$

It is clear that all the above rules are derived rules of inference, as can be proven by a straightforward induction on the number of free variables in $\vec{N}$.

6.4.1 Normalization of Input Variables

We present in Figure 6.8 the rules for normalization of input variables. Since we are currently working with the restriction to ground goals, those are all the variables which are universally quantified in the head of a clause. When they occur positively in dynamic assumptions, they will carry some instantiation. It helps to simplify the presentation of the clause complementation algorithm if we forbid this kind of occurrence. The idea it to replace every positive occurrence of an input variable in an assumption with a new (local) universal variable, which is then constrained to be equal to the input ones.

Example 6.22 *Consider the typing clause for lambda terms:*

$$\begin{array}{lll} \mathit{oflam} & : & \forall E\!:\!\mathit{exp} \to \mathit{exp}. \forall T_1, T_2\!:\!\mathit{tp}. \\ & & \quad \mathit{of}\ (\mathit{lam}\ E)\ (\mathit{arrow}\ T_1\ T_2) \\ & & \qquad \leftarrow (\forall x\!:\!\mathit{exp}. \mathit{of}\ x\ T_1 \to \mathit{of}\ (E\ x)\ T_2). \end{array}$$

As T_1 is an input term occurring positively in the assumption 'of x T_1', normalization of input variables inserts a new variable T and constrains it to be equal to T_1:

$$\begin{array}{lll} \mathit{oflam}' & : & \forall E\!:\!\mathit{exp} \to \mathit{exp}. \forall T_1, T_2\!:\!\mathit{tp}. \\ & & \quad \mathit{of}\ (\mathit{lam}\ E)\ (\mathit{arrow}\ T_1\ T_2) \\ & & \qquad\qquad (\forall T\!:\!\mathit{tp}. \mathit{of}\ x\ T \leftarrow T \doteq T_1) \to \\ & & \qquad\qquad\qquad \mathit{of}\ (E\ x)\ T_2). \end{array}$$

This procedure is realized by the judgments:

- $D \overset{D}{\mapsto} D'$: clause normalization.

- $\Gamma \vdash_\Phi G \overset{G}{\mapsto} G'$: goal normalization.
- $\Phi; \Gamma \vdash D \mapsto D'$: assumption normalization w.r.t. global parameters in Φ.
- $\Lambda; \Phi; \Gamma \vdash M \mapsto M', E$: term M normalizes to term M' returning a conjunction of equations E w.r.t. bound (Γ), global (Φ) and existential (Λ) variables.

The first three judgments simply recur on the structure of clauses, goal and assumptions, keeping track of bound, global and existential variables in assumptions, until we normalize an assumption $\forall(Q \leftarrow G)$: here we descend in every term by introducing new local existential variable and binding them to the global one via an equation E. For the sake of conciseness, we do this for the simply-typed fragment; the generalization to the strict one is immediate. In rule $\overset{D}{\mapsto}\leftarrow^\rho$ we introduce in the global context all the global parameters in the domain of ρ with the appropriate typing.

It is clear, although very tedious, to verify that this transformation preserves provability and denial:

Lemma 6.23 *Assume $\Lambda; \Phi; \Gamma \vdash M \mapsto M', E$; for every ground N, every $\Gamma; \mathcal{D}$, $\Gamma; \mathcal{D} \vdash M \doteq N$ iff $\Gamma; \mathcal{D} \vdash (M' \doteq N) \wedge E$.*

Proof: By a straightforward induction on the structure of $\Lambda; \Phi; \Gamma \vdash M \mapsto M', E$. □

Theorem 6.24 *Let $\Gamma \vdash \theta : \Phi$, $\Phi; \Gamma \vdash \mathcal{D} \mapsto \mathcal{D}'$, $D \overset{D}{\mapsto} D'$ and $\Gamma \vdash_\Phi G \overset{G}{\mapsto} G'$:*

1. *$\Gamma; [\theta]\mathcal{D} \vdash [\theta]D \gg Q$ iff $\Gamma; [\theta]\mathcal{D}' \vdash [\theta]D' \gg Q$.*
2. *$\Gamma; [\theta]\mathcal{D} \vdash [\theta]G$ iff $\Gamma; [\theta]\mathcal{D}' \vdash [\theta]G'$.*

Proof: By a straightforward mutual induction on the structure of the given derivations, using Lemma 6.23. □

6.5 The Clause Complement Algorithm

We now introduce in Figure 6.9 and Figure 6.10 the rules for *static* and *dynamic clause complementation*. Consider a rule $\forall(q\ \vec{M} \leftarrow G)$; its complement must contain a 'factual' part motivating failure due to clash with the head; the remainder $\mathrm{Not_G}(G)$ expresses failure in the body, if any. Clause complementation must discriminate whether this rule belongs to the static or dynamic definition of a predicate. In the first case all the relevant information is already present in the head of the clause and we can, without further ado, use the term complementation algorithm described in Chapter 4. This is accomplished by the rule $\mathrm{Not_D} \rightarrow$, where a set of negative facts is built via term complementation $\mathrm{Not}(\vec{M})$ applied in the empty context to the (vector of) terms in the clause head; namely $\bigwedge_{\vec{N} \in \cdot \vdash \mathrm{Not}(\vec{M})} \forall(\neg(q\ \vec{N}) \leftarrow \top)$; moreover the negative counterpart of the source clause is obtained via complementation of the body. The Partition Lemma (Corollary 4.21) guarantees the soundness and completeness of this case.

Assumption complementation is realized by the judgment $\Gamma \vdash \mathrm{Not}_\alpha(D)$, which can be seen as a type directed *parameter-conscious* version of clause complementation. In a first approximation, we can think of complementation of assumptions, which are by definition parametric in some x, as static clause complementation w.r.t. x. Informally, for an atomic assumption, say $q\ M_1 \ldots M_{i-1}\ e_x\ M_{i+1} \ldots M_n$, its complement can be taken as $\mathrm{Not_D}(q_{e_x}\ M_1 \ldots M_{i-1}\ M_{i+1} \ldots M_n)$, for a shallow parameter expression e_x. This is accomplished in two main phases: first, e_x is propagated in every position $1 \leq j \leq n$ holding a rigid term of compatible type, but different from e_x itself. This alone builds an element in the complement set. Secondly, the idea is to can apply term complementation 'around' e_x. In particular, if M_j is a variable, by normalization of input variables, it must be a local one and term complement does not contribute anything as expected. If M_j is a parameter expression or a compound term, we simply take the term complement of the former term, with the notable difference of passing the current parameter context to term complement.

$$\frac{D_1 \stackrel{D}{\mapsto} D_1' \qquad D_2 \stackrel{D}{\mapsto} D_2'}{D_1 \wedge D_2 \stackrel{D}{\mapsto} D_1' \wedge D_2'} \stackrel{D}{\mapsto} \wedge \qquad \frac{D_1 \stackrel{D}{\mapsto} D_1' \qquad D_2 \stackrel{D}{\mapsto} D_2'}{D_1 \vee D_2 \stackrel{D}{\mapsto} D_1' \vee D_2'} \stackrel{D}{\mapsto} \vee$$

$$\frac{X \in \{\top, \bot\}}{X \stackrel{D}{\mapsto} X} \stackrel{D}{\mapsto} X \qquad \frac{\cdot \vdash_{\mathrm{dom}(\rho)} [\rho]G \stackrel{G}{\mapsto} [\rho]G'}{\forall(Q \leftarrow G) \stackrel{D}{\mapsto} \forall(Q \leftarrow G')} \stackrel{D}{\mapsto}\leftarrow^{\rho}$$

$$\frac{X \in \{\top, \bot, Q\} \text{ or (dis)eq}}{\Gamma \vdash_\Phi X \stackrel{G}{\mapsto} X} \stackrel{G}{\mapsto} X \qquad \frac{(\Gamma, y{:}A); \mathcal{D} \vdash_\Phi [y/x]G \stackrel{G}{\mapsto} [y/x]G'}{\Gamma \vdash_\Phi \forall x : a.\, G \stackrel{G}{\mapsto} \forall x : a.\, G'} \stackrel{G}{\mapsto} \forall^y$$

$$\frac{\Phi; \Gamma \vdash D \mapsto D' \qquad \Gamma; \mathcal{D} \wedge D \vdash_\Phi G \stackrel{G}{\mapsto} G'}{\Gamma \vdash_\Phi (D \to G) \stackrel{G}{\mapsto} (D' \to G')} \stackrel{G}{\mapsto}\to$$

$$\frac{\Gamma \vdash_\Phi G_1 \stackrel{G}{\mapsto} G_1' \qquad \Gamma \vdash_\Phi G_2 \stackrel{G}{\mapsto} G_2'}{\Gamma \vdash_\Phi G_1 \wedge G_2 \stackrel{G}{\mapsto} G_1' \wedge G_2'} \stackrel{G}{\mapsto} \wedge \qquad \frac{\Gamma \vdash_\Phi G_1 \stackrel{G}{\mapsto} G_1' \qquad \Gamma \vdash_\Phi G_2 \stackrel{G}{\mapsto} G_2'}{\Gamma \vdash_\Phi G_1 \vee G_2 \stackrel{G}{\mapsto} G_1' \vee G_2'} \stackrel{G}{\mapsto} \vee$$

$$\frac{\Lambda = FV(\overline{M_n}) \setminus \mathrm{dom}(\Gamma \cup \Phi) \qquad \Lambda; \Phi; \Gamma \vdash M_1 \mapsto M_1', E_1 \cdots \Lambda; \Phi; \Gamma \vdash M_n \mapsto M_n', E_n}{\Phi; \Gamma \vdash \forall(q\ \overline{M_n} \leftarrow G) \mapsto \forall(q\ \overline{M_n'} \leftarrow E_1 \wedge \ldots \wedge E_n \wedge G)} \mapsto\to$$

$$\frac{X \in \{\top, \bot\}}{\Phi; \Gamma \vdash X \mapsto X} \mapsto X$$

$$\frac{\Phi; \Gamma \vdash D_1 \mapsto D_1' \qquad \Phi; \Gamma \vdash D_2 \mapsto D_2'}{\Phi; \Gamma \vdash D_1 \wedge D_2 \mapsto D_1' \wedge D_2'} \mapsto \wedge \qquad \frac{\Phi; \Gamma \vdash D_1 \mapsto D_1' \qquad \Phi; \Gamma \vdash D_2 \mapsto D_2'}{\Phi; \Gamma \vdash D_1 \vee D_2 \mapsto D_1' \vee D_2'} \mapsto \vee$$

$$\frac{}{\Lambda; \Phi; (\Gamma, x{:}A) \vdash x \mapsto x, \top}\, bv \qquad \frac{Z \text{ new}}{\Lambda; (\Phi, u{:}A); \Gamma \vdash u \mapsto Z, Z \doteq u}\, iv \qquad \frac{}{(\Lambda, x{:}A); \Phi; \Gamma \vdash x \mapsto x, \top}\, lv$$

$$\frac{\Lambda; \Phi; \Gamma \vdash M_1 \mapsto M_1', E_1 \qquad \Lambda; \Phi; \Gamma \vdash M_2 \mapsto M_2', E_2}{\Lambda; \Phi; \Gamma \vdash M_1\ M_2 \mapsto M_1'\ M_2', E_1 \wedge E_2}\, App \qquad \frac{\Lambda; \Phi; (\Gamma, x{:}A) \vdash M \mapsto M', E}{\Lambda; \Phi; \Gamma \vdash \lambda x : A.\, M \mapsto \lambda x : A.\, M', E}\, \lambda$$

Figure 6.8: Clause, goal, assumption and term normalization

$$\frac{}{\mathrm{Not_D}(\top) = \bot}\ \mathrm{Not_D}\top \qquad \frac{}{\mathrm{Not_D}(\bot) = \top}\ \mathrm{Not_D}\bot$$

$$\frac{\cdot \vdash \mathrm{Not_G}(G) = G'}{\mathrm{Not_D}(\forall(q\ \vec{M} \leftarrow G)) = \bigwedge_{\vec{N} \in \cdot \vdash \mathrm{Not}(\vec{M})} \forall(\neg(q\ \vec{N}) \leftarrow \top) \wedge \forall(\neg q\ \vec{M} \leftarrow G')}\ \mathrm{Not_D} \leftarrow$$

$$\frac{\mathrm{Not_D}(D_1) = D_1' \qquad \mathrm{Not_D}(D_2) = D_2'}{\mathrm{Not_D}(D_1 \wedge D_2) = D_1' \vee D_2'}\ \mathrm{Not}\wedge$$

$$\frac{\mathrm{Not_D}(D_1) = D_1' \qquad \mathrm{Not_D}(D_2) = D_2'}{\mathrm{Not_D}(D_1 \vee D_2) = D_1' \wedge D_2'}\ \mathrm{Not}\vee$$

Figure 6.9: Clause complementation: $\mathrm{Not_D}(D) = D'$

However, not all parameters are born alike; in many situations, a parametric judgment is used simply to descend into a scoping construct, while the parameter itself does not play a role w.r.t. provability and denial. Consider, for example, the following program that checks whether a universal formula is a D-clause:

$$\begin{array}{rcl}
form, term & : & type. \\
all & : & (term \to form) \to form). \\
isd & : & form \to \mathbf{o} \\
isdall & : & \forall D{:}term \to form.\, isd\ (all\ D) \\
& & \qquad \leftarrow \forall x{:}term.\, isd\ (D\ x).
\end{array}$$

This phenomenon is known in the literature under the name of *subordination* and has been extensively studies in the dependent typed context [Vir99]. In the simply typed setting, this relation collapses to merely checking whether the type of the parameters is equal to the target type of some argument of the predicate. More formally, we say that $x{:}A$ is *relevant to* Q if $head(Q) \equiv q$, $\Sigma(q) = A_1, \ldots, A_n \to \mathbf{o}$ and for some $1 \leq i \leq n$ it holds that $\mathrm{target}(A_i) \equiv A$; we denote this with xR^iq. In the above example, as $term \not\equiv form$, then $x : term$ is *not* relevant to $isd : form \to \mathbf{o}$. Wrt. clause complementation, if the parameter x is *not* relevant to q, we do not need to build complementary facts out of it, although they would not impact soundness, i.e. Exclusivity (Theorem 6.32).

We concentrate on rules $\mathrm{Not}_\alpha\top$ and $\mathrm{Not}_\alpha \to$. The notation $[e_x/Z_i]\overline{Z_n}$ is an abbreviation for $Z_1 \ldots Z_{i-1}\ e_x\ Z_{i+1} \ldots Z_n$, where the Z's are fresh logic variables; similarly for $[e_x/Z_i, N/Z_j]\overline{Z_n}$. The main loop goes as follows:

- Choose a parameter $x{:}a \in \Gamma$.
- Propagate e_x in every 'odd' position.
- Locate a type A_i such that x is relevant to Q at i:
  - Complement D w.r.t. x and i.
  - Repeat for every relevant position i.
- Repeat for every x.

Rule $\mathrm{Not}_\alpha \to$ is the most complicated one: fixed a parameter x, there are two ways in which an atomic assumption $q\ M_1 \ldots M_{i-1}\ e_x\ M_{i+1} \ldots M_n$, needs to be complemented. First, any atom with same head with a term $N_j \equiv e_x$ for $i \neq j$ is in the complement of the former. Moreover, since they do differ in one coordinate,

$$\frac{}{\Gamma \vdash \mathrm{Not}_{\alpha}(\bot) = \top}\ \mathrm{Not}_{\alpha}\bot$$

$$\frac{\Sigma(q) = A_1 \to \cdots \to A_n \to \mathbf{o}}{\Gamma \vdash \mathrm{Not}_{\alpha}(\top_q) = \bigwedge_{x \in dom(\Gamma)} (\bigwedge_{1 \leq i \leq n, xR^i q} \Gamma \vdash \mathrm{Not}^C_x(\top_q))}\ \mathrm{Not}_{\alpha}\top$$

$$\frac{\Gamma \vdash \mathrm{Not}_{\mathrm{G}}(G) = G'}{\Gamma \vdash \mathrm{Not}_{\alpha}(\forall(q\ \overline{M_n} \leftarrow G)) = (\bigwedge_{x \in dom(\Gamma)} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{M_n}) \wedge (\bigwedge_{1 \leq i \leq n, xR^i q} \Gamma \vdash \mathrm{Not}^i_x(q\ \overline{M_n}))) \wedge \forall(\neg q\ \overline{M_n}) \leftarrow G')}\ \mathrm{Not}_{\alpha} \leftarrow$$

$$\frac{\Gamma \vdash \mathrm{Not}_{\alpha}(D_1) = D_1' \qquad \Gamma \vdash \mathrm{Not}_{\alpha}(D_2) = D_2'}{\Gamma \vdash \mathrm{Not}_{\alpha}(D_1 \wedge D_2) = D_1' \vee D_2'}\ \mathrm{Not}\wedge$$

$$\frac{\Gamma \vdash \mathrm{Not}_{\alpha}(D_1) = D_1' \qquad \Gamma \vdash \mathrm{Not}_{\alpha}(D_2) = D_2'}{\Gamma \vdash \mathrm{Not}_{\alpha}(D_1 \vee D_2) = D_1' \wedge D_2'}\ \mathrm{Not}\vee$$

$$\frac{\Gamma \vdash M_i : A_i \text{ rigid}, \cdot \vdash sh(x, A_i) = e_x, M_i \not\equiv e_x}{\Gamma \vdash \mathrm{Not}^C_x(q\ \overline{M_n}) = \bigwedge_{1 \leq i \leq n} \forall Z_1 \!:\! A_1. \ldots \forall Z_n \!:\! A_n. \neg q\ [e_x/Z_i]\overline{Z_n}}\ \Gamma \vdash \mathrm{Not}^C_x$$

$$\frac{M_i \equiv e_x}{\Gamma \vdash \mathrm{Not}^i_x(q\ \overline{M_n}) = \bigwedge_{1 \leq j \leq n}^{j \neq i} (\bigwedge_{N \in (\Gamma \vdash \mathrm{Not}(M_j))} (\forall Z_1 \!:\! A_1. \ldots \forall Z_n \!:\! A_n. \neg q\ [e_x/Z_i, N/Z_j]\overline{Z_n}))}\ \mathrm{Not}^i_x$$

$$\frac{}{\Gamma \vdash sh(x, a) = x}\ e_x\mathrm{At} \qquad \frac{\Gamma, y \!:\! A \vdash sh(x, B) = e_x}{\Gamma \vdash sh(x, A \to B) = \lambda y \!:\! A.\, e_x}\ e_x \to^y$$

Figure 6.10: Assumption complementation: $\Gamma \vdash \mathrm{Not}_{\alpha}(D) = D'$

we may as well leave open every other term. This is encoded in rule $\Gamma \vdash \mathrm{Not}_x^C$. Of course, we have to make sure that N_j is not flex and that the type is appropriate. Since x is passed as a parameter, the $\Gamma \vdash sh(x, A)$ judgment builds a shallow parameter expression as required by the type of the position in q where it ought to occur.

Secondly, we have to take into account the case $M_i \equiv e_x$: here we can build a set of complementary facts by pivoting on e_x and making another position different. Luckily, we can achieve this via a call to term complementation and build a set of complementary facts similarly to clause complementation. The difference is that we pass to term complementation, as a context, the set of parametric bound variables. Finally, both processes are repeated for every M_i.

Notice the different treatment of the trivial clause $\top$ by rules $\mathrm{Not_D}\top$ and $\mathrm{Not}_\alpha\top$: if no parameter has been assumed, then $\top$ truly stands for the empty predicate definition and its complement is the universal definition $\perp$. If, on the other side Γ is *not* empty, it means that $\top_q$ has been introduced during the $\top$-normalization preprocessing phase and has been localized to the predicate q. Here we need to construct a new negative assumption w.r.t. q, x, i in case $\top_q$ is the only dynamic definition of q. As $\top_q$ carries no information at all concerning q, the most general negative assumption is added. This is accomplished again by rule $\Gamma \vdash \mathrm{Not}_x^C$, where we make the convention to view $\top_q$ as a degenerate case of $q\ \overline{M_n}$ where the sequence $\overline{M_n}$ is empty (and thus the condition trivially satisfied).

The remaining (common) rules for static and dynamic clause complementation simply recur on the program respecting the duality of conjunction and disjunction w.r.t. negation. This a somewhat delicate point and therefore we discuss it in some details. Intuitively, negative clauses stemming from the complement of the definition of a predicate need to be considered simultaneously. In fact, if an goal is unprovable from its definition, then its negation must be provable from the complement of each clause of its definition; symmetrically if it is provable, then its negation must be unprovable from the complement of at least one. What we have described coincides the operational semantics of an operator, which works exactly as disjunction. This is arguably at odd with the commonly held goal-oriented interpretation of the sequent calculus as uniform proofs, since case analysis makes the latter incomplete w.r.t. minimal logic (but see [NL95] for ways to incorporate the former in the framework of uniform proofs). In Section 6.9.2 we will show how to 'compile away' all occurrences of $\vee$ in clauses. This is an higher-order equivalent of the intersection operator '@' described in [BMPT90] and will restore completeness of uniform proofs

Goal complementation, that is the judgment $\mathrm{Not_G}(G)$ depicted in Figure 6.11 is straightforward, since it only brings the body into a normalized format; namely, in what we may call *parametric negation normal form*, to stress the distinction from classical negation normal form or even from negation normal form in constructive logics as extension of intuitionism with strong negation [Nel49]. This re-iterates the problem with strong negation we have hinted in Section 5.3; negation is pushed inward but jumps over parametric-hypothetical judgments to respect the operational interpretation of unprovability.

As a final remark, we note again that we must take the complementation of a program, seen as a conjunction, *predicate definition-wise* rather that *clause-wise.* In fact, it would be incorrect to simply negate a program as it would introduce disjunctions rather than conjunction among predicate definitions. The same remark applies to the 'dynamic' program $\mathcal{D}$. Formally, given a fixed signature $\Sigma_\mathcal{P}$ and a program $\mathcal{P}$:

$$\mathrm{Not_D}(\mathcal{P}) \quad = \quad \bigwedge_{q \in \Sigma_\mathcal{P}} \mathrm{Not_D}(\mathrm{def}(q, \mathcal{P}))$$

Similarly for a (possibly run-time) context $\Gamma; \mathcal{D}$:

$$\Gamma \vdash \mathrm{Not}_\alpha(\mathcal{D}) \quad = \quad \bigwedge_{q \in \Sigma_\mathcal{P}} \Gamma \vdash \mathrm{Not}_\alpha(\mathrm{def}(q, \mathcal{D}))$$

Note that if $\Gamma; \mathcal{D}$ is a run-time context, it consists of a conjunction of blocks, i.e. it has the form $\Gamma'', \lceil\Gamma'\rceil; \mathcal{D}'' \wedge \lceil\mathcal{D}'\rceil$; in this case the definition of q consists of the conjunction of the definition for *every* block, namely: $\Gamma' \vdash \mathrm{def}(q, \mathcal{D}') \wedge \Gamma'' \vdash \mathrm{def}(q, \mathcal{D}'')$. We abbreviate $\mathrm{Not_D}(\mathrm{def}(q))$ in $\mathrm{def}(\neg q)$. If $\mathcal{P}^+$ is the source program, we use $\mathcal{P}^-$ for $\mathrm{Not_D}(\mathcal{P}^+)$.

Finally, we provide an example, which, even though is somewhat a special case of the general procedure, assumption complementation being trivial, it helps to clarify the rules for clause and goal complementation

$$\frac{}{\Gamma \vdash \mathrm{Not_G}(\top) = \bot}\,\mathrm{Not_G}\top \qquad \frac{}{\Gamma \vdash \mathrm{Not_G}(\bot) = \top}\,\mathrm{Not_G}\bot$$

$$\frac{}{\Gamma \vdash \mathrm{Not_G}(\vec{M} \doteq \vec{N}) = (\vec{M} \not\doteq \vec{N})}\,\mathrm{Not} \doteq \qquad \frac{}{\Gamma \vdash \mathrm{Not_G}(\vec{M} \not\doteq \vec{N}) = (\vec{M} \doteq \vec{N})}\,\mathrm{Not} \not\doteq$$

$$\frac{}{\Gamma \vdash \mathrm{Not_G}(Q) = \neg Q}\,\mathrm{Not_G}At$$

$$\frac{\Gamma, y{:}a \vdash \mathrm{Not_G}([y/x]G) = [y/x]G'}{\Gamma \vdash \mathrm{Not_G}(\forall x{:}a.\, G) = \forall x{:}a.\, G'}\,\mathrm{Not_G}\forall^y$$

$$\frac{\Gamma \vdash \mathrm{Not_G}(G_1) = G_1' \qquad \Gamma \vdash \mathrm{Not_G}(G_2) = G_2'}{\Gamma \vdash \mathrm{Not_G}(G_1 \wedge G_2) = G_1' \vee G_2'}\,\mathrm{Not}\wedge$$

$$\frac{\Gamma \vdash \mathrm{Not_G}(G_1) = G_1' \qquad \Gamma \vdash \mathrm{Not_G}(G_2) = G_2'}{\Gamma \vdash \mathrm{Not_G}(G_1 \vee G_2) = G_1' \wedge G_2'}\,\mathrm{Not}\vee$$

$$\frac{\Gamma \vdash \mathrm{Not_G}(G) = G'}{\Gamma \vdash \mathrm{Not_G}(D \to G) = D \to G'}\,\mathrm{Not_G} \to$$

Figure 6.11: Goal complementation: $\Gamma \vdash \mathrm{Not_G}(G) = G'$

in isolation. We refer to the next Section (6.6) for more complex examples. We use $\forall F_1, F_2 : A.\, X$ as an abbreviation of $\forall F_1{:}A.\, \forall F_2{:}A.\, X$.

Example 6.25 *A λ-expression is* closed *if it has no occurrence of free variables; let $z : exp$ be a constant:*

$$\begin{array}{rcl} cloz & : & closed\ z. \\ clolam & : & \forall E{:}exp.\, closed\ (lam\ E) \leftarrow (\forall x{:}exp.\ (closed\ x \to closed\ (E\ x).) \\ cloapp & : & \forall E_1{:}exp.\, \forall E_2{:}exp.\, closed\ (app\ E_1\ E_2) \leftarrow closed\ E_1 \wedge closed\ E_2. \end{array}$$

Now, def(closed) = $cloz \wedge clolam \wedge cloapp$. Note that $x{:}exp \vdash \mathrm{Not}_\alpha(closed\ x) = \top$. Therefore:

$$def(\neg closed) \quad = \quad \mathrm{Not_D}(cloz) \vee \mathrm{Not_D}(clolam) \vee \mathrm{Not_D}(cloapp)$$

where:

$$\begin{array}{rcl} \mathrm{Not_D}(cloz) & = & \displaystyle\bigwedge_{N \in \cdot \vdash \mathrm{Not}(z)} \neg\forall(closed\ N) \\ & = & (\forall F{:}exp.\, \neg closed\ (lam\ F)) \wedge \forall F_1, F_2{:}exp.\, \neg closed\ (app\ F_1\ F_2). \\ \mathrm{Not_D}(clolam) & = & \displaystyle\bigwedge_{N \in \cdot \vdash \mathrm{Not}(lam\ E)} \forall(\neg closed\ N) \wedge \\ & & \forall E{:}exp.\, \neg closed\ (lam\ E) \leftarrow \mathrm{Not_G}(\forall x{:}exp.\ (closed\ x \to closed\ (E\ x)) \\ & = & \neg closed\ z \wedge (\forall F_1, F_2{:}exp.\, \neg closed\ (app\ F_1\ F_2)) \wedge \\ & & \forall E{:}exp.\, \neg closed\ (lam\ E) \leftarrow (\forall x{:}exp.\ (closed\ x \to \neg closed\ (E\ x)). \\ \mathrm{Not_D}(cloapp) & = & \displaystyle\bigwedge_{N \in \cdot \vdash \mathrm{Not}(app\ E_1\ E_2))} \neg\forall(closed\ N) \wedge \end{array}$$

$$\frac{X \in \{\top, \bot\}}{aug_{\mathrm{D}}(X) = X}\ aug_{\mathrm{D}}X$$

$$\frac{aug_{\mathrm{D}}(D_1) = D_1^a \qquad aug_{\mathrm{D}}(D_2) = D_2^a}{aug_{\mathrm{D}}(D_1 \wedge D_2) = D_1^a \wedge D_2^a}\ aug_{\mathrm{D}}\wedge$$

$$\frac{aug_{\mathrm{D}}(D_1) = D_1^a \qquad aug_{\mathrm{D}}(D_2) = D_2^a}{aug_{\mathrm{D}}(D_1 \vee D_2) = D_1^a \vee D_2^a}\ aug_{\mathrm{D}}\vee$$

$$\frac{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}([\rho]G) = [\rho]G^a}{aug_{\mathrm{D}}(\forall(Q \leftarrow G)) = \forall(Q \leftarrow G^a)}\ aug_{\mathrm{D}} \leftarrow^{\rho}$$

Figure 6.12: Clause augmentation: $aug_{\mathrm{D}}(D) = D^a$

$$\begin{array}{rl} & \forall E_1, E_2 : exp.\, \neg closed\ (app\ E_1\ E_2) \leftarrow \mathrm{Not}_{\mathrm{G}}(closed\ E_1 \wedge closed\ E_2) \\ = & \neg closed\ z \wedge (\forall F : exp.\, \neg closed(lam\ F)) \wedge \\ & \forall E_1, E_2 : exp.\, \neg closed\ (app\ E_1\ E_2) \leftarrow \neg closed\ E_1 \vee \neg closed\ E_2. \end{array}$$

If you want to see the definition simplified, please skip to Example 6.39 and the final result in Example 6.43.

6.6 Augmentation

Now that we have discussed how to perform clause, assumption and goal complementation, we synchronize it together in a phase we call *augmentation*, which simply inserts, at compile-time, the correct assumption complementation in a goal and in turn in a clause. We give one judgment to augment a program, $aug_{\mathrm{D}}(D)$, depicted in 6.12. which merely recurs on the structure of clauses until it calls goal augmentation, $\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G)$ (Figure 6.13). The latter traverses a goal collecting parameters in Γ and assumptions in $\mathcal{D}$. When it reaches an atom, either it stops, as no parameter has been introduced, or it passes $\Gamma, \mathcal{D}$ to assumption complementation.

Some examples will make the whole process clear; consider the `copy` program on λ-terms:

$$\begin{array}{rcl} cpapp & : & \forall E_1, E_2, F_1, F_2 : exp. \\ & & copy\ (app\ E_1\ E_2)\ (app\ F_1\ F_2) \\ & & \quad \leftarrow copy\ E_1\ F_1 \\ & & \quad \leftarrow copy\ E_2\ F_2. \\ cplam & : & \forall E : exp \to exp. \forall F : exp \to exp. \\ & & copy\ (lam\ E)\ (lam\ F) \\ & & \quad \leftarrow (\forall x : exp.\, copy\ x\ x \\ & & \qquad \to copy\ (E\ x)\ (F\ x)). \end{array}$$

The augmentation judgment $aug_{\mathrm{D}}(cplam)$ calls $\cdot \vdash aug_{\mathrm{G}}(\forall x : exp.\, copy\ x\ x \to copy\ (E\ x)\ (F\ x))$, which collects the context $x{:}exp; copy\ x\ x$ and calls $x{:}exp \vdash \mathrm{Not}_{\alpha}(copy\ x\ x)$. We start by observing that $\Gamma \vdash \mathrm{Not}_x^C(copy\ x\ x) = \top$, since the conditions on rule $\Gamma \vdash \mathrm{Not}_x^C$ are not satisfied. Then, note that $x{:}exp \vdash \mathrm{Not}(x) = \{lam\ F', app\ F_1\ F_2\}$. This yields:

$$x{:}exp \vdash \mathrm{Not}_1^x(copy\ x\ x) \quad = \quad (\forall F', F_1, F_2 : exp.\, \neg copy\ x\ (lam\ F') \wedge \neg copy\ x\ (app\ F_1\ F_2))$$

Symmetrically:

$$x{:}exp \vdash \mathrm{Not}_2^x(copy\ x\ x) \quad = \quad (\forall F'', F_3, F_4 : exp.\, \neg copy\ (lam\ F'')\ x \wedge \neg copy\ (app\ F_3\ F_4)\ x)$$

$$\frac{X \in \{\top, \bot\} \text{ or (dis)eq}}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(X) = X}\, aug_{\mathrm{G}}X$$

$$\frac{}{\cdot; \top \vdash aug_{\mathrm{G}}(Q) = Q}\, aug_{\mathrm{G}}\mathrm{At}^{\top}$$

$$\frac{\Gamma \vdash \mathrm{Not}_{\alpha}(\mathcal{D}) = \mathcal{D}_{\neg}}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(Q) = \mathcal{D}_{\neg} \to Q}\, aug_{\mathrm{G}}At$$

$$\frac{\Gamma; (\mathcal{D} \wedge D) \vdash aug_{\mathrm{G}}(G) = G^a}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(D \to G) = D \to G^a}\, aug_{\mathrm{G}} \to$$

$$\frac{(\Gamma, y{:}a); \mathcal{D} \vdash aug_{\mathrm{G}}([y/x]G) = [y/x]G^a}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(\forall x{:}a.\, G) = \forall x{:}a.\, G^a}\, aug_{\mathrm{G}}\forall^y$$

$$\frac{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_1) = G_1^a \qquad \Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_2) = G_2^a}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_1 \wedge G_2) = G_1^a \wedge G_2^a}\, aug_{\mathrm{G}}\wedge$$

$$\frac{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_1) = G_1^a \qquad \Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_2) = G_2^a}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G_1 \vee G_2) = G_1^a \vee G_2^a}\, aug_{\mathrm{G}}\vee$$

Figure 6.13: Goal augmentation: $\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(G) = G^a$

This will yield the augmented clause:

$$\begin{array}{lll} aug_{\mathrm{D}}(cplam) & : & \forall E{:}exp \to exp.\, \forall F{:}exp \to exp. \\ & & copy\ (lam\ E)\ (lam\ F) \\ & & \quad \leftarrow (\forall x{:}exp. \\ & & \quad (\forall F', F'', F_1, F_2, F_3, F_4{:}exp. \\ & & \qquad \neg copy\ x\ (lam\ F') \wedge \neg copy\ x\ (app\ F_1\ F_2) \wedge \\ & & \qquad \neg copy\ (lam\ F'')\ x \wedge \neg copy\ (app\ F_3\ F_4)\ x) \\ & & \qquad\quad \to copy\ x\ x \to copy\ (E\ x)\ (F\ x)). \end{array}$$

while of course, $aug_{\mathrm{D}}(cpapp) = cpapp$, as enforced by $aug_{\mathrm{G}}\mathrm{At}\top$. If we had a two-parameter version of *copy*:

$$\begin{array}{lll} cplam' & : & \forall E{:}exp \to exp.\, \forall F{:}exp \to exp. \\ & & copy'\ (lam\ E)\ (lam\ F) \\ & & \quad \leftarrow (\forall x{:}exp.\, \forall y{:}exp.\, (copy'\ x\ y) \\ & & \qquad \to copy'\ (E\ x)\ (F\ y)). \end{array}$$

Then we would get first:

$$\begin{array}{rcl} \Gamma \vdash \mathrm{Not}^C_x(copy'\ x\ y) & = & (\forall E''{:}exp.\, \neg copy'\ E''\ x) \\ \mathrm{Not}^C_y(copy'\ x\ y) & = & (\forall F''{:}exp.\, \neg copy'\ y\ F''). \end{array}$$

Secondly, by computing, respectively, for $\Gamma = x{:}exp, y{:}exp$, $\Gamma \vdash \mathrm{Not}(y)$ and $\Gamma \vdash \mathrm{Not}(x)$:

$$\begin{array}{rcl} \Gamma \vdash \mathrm{Not}^x_1(copy'\ x\ y) & = & (\forall F_0, F_1, F_2{:}exp.\, \neg copy'\ x\ (lam\ F_0) \wedge \neg copy'\ x\ (app\ F_1\ F_2) \wedge \neg copy' x\ x) \\ \Gamma \vdash \mathrm{Not}^y_2(copy'\ x\ y) & = & (\forall F_0', F_1', F_2'{:}exp.\, \neg copy'\ (lam\ F_0')\ y \wedge \neg copy'\ (app\ F_1'\ F_2')\ y \wedge \neg copy'\ y\ y) \end{array}$$

yielding the augmented clause:

$$
\begin{aligned}
aug_{\mathrm{D}}(cplam') \quad : \quad & \forall E\!:\!exp \to exp.\, \forall F\!:\!exp \to exp. \\
& copy'\ (lam\ E)\ (lam\ F) \\
& \quad \leftarrow (\forall x\!:\!exp.\, \forall y\!:\!exp. \\
& \quad\quad (\forall E'\!:\!exp.\, \neg copy'\ E'\ x) \wedge \\
& \quad\quad (\forall F'\!:\!exp.\, \neg copy'\ y\ F') \wedge \\
& \quad\quad \forall(\neg copy'\ x\ (lam\ F_0) \wedge \neg copy'\ x\ (app\ F_1\ F_2) \wedge \neg copy' x\ x \wedge \\
& \quad\quad \neg copy'\ (lam\ F_0')\ y \wedge \neg copy'\ (app\ F_1'\ F_2')\ y \wedge \neg copy'\ y\ y) \\
& \quad\quad\quad \to copy'\ x\ y \to copy'\ (E\ x)\ (F\ y)).
\end{aligned}
$$

Note that by static analysis of `copy'`, we know that x will never end up in the second argument of `copy'` and symmetrically this applies to y, too. Thus the call to $\Gamma \vdash \mathrm{Not}_x^C, \mathrm{Not}_y^C$ are, in this case, useless; however since this kind of data flow analysis is in general undecidable, the augmentation procedure inserts the negation of a clause for every 'odd' position relevant to the pivot parameter. Now, consider the typing clause for lambda terms:

$$
\begin{aligned}
oflam \quad : \quad & \forall E\!:\!exp \to exp.\, \forall T_1, T_2\!:\!tp. \\
& of\ (lam\ E)\ (arrow\ T_1\ T_2) \\
& \quad \leftarrow (\forall x\!:\!exp.\, of\ x\ T_1 \to of\ (E\ x)\ T_2).
\end{aligned}
$$

As T_1 is an input term, normalization of input variables inserts the appropriate equation, where `oflam` $\overset{D}{\mapsto}$ `oflam'`:

$$
\begin{aligned}
oflam' \quad : \quad & \forall E\!:\!exp \to exp.\, \forall T_1, T_2\!:\!tp. \\
& of\ (lam\ E)\ (arrow\ T_1\ T_2) \\
& \quad\quad (\forall T\!:\!tp.\, of\ x\ T \leftarrow T \doteq T_1) \to \\
& \quad\quad\quad of\ (E\ x)\ T_2).
\end{aligned}
$$

Both $\Gamma \vdash \mathrm{Not}_x^C$ and Not_x^1 generate no contribution; in particular $x{:}exp \vdash \mathrm{Not}_x^1(\forall T\!:\!tp.\, of\ x\ T)$ calls the Not_x^i rule, but term complementation (applied to the existential variable T) yields the trivial clause. Therefore, by $\mathrm{Not}_{\mathrm{G}}(T \doteq T_1)$, augmentation will result into:

$$
\begin{aligned}
aug_{\mathrm{D}}(oflam) \quad : \quad & \forall E\!:\!exp \to exp.\, \forall T_1, T_2\!:\!tp. \\
& of\ (lam\ E)\ (arrow\ T_1\ T_2) \\
& \quad \leftarrow (\forall x\!:\!exp.\, of\ x\ T_1 \to \\
& \quad\quad (\forall T\!:\!tp.\, \neg of\ x\ T \leftarrow T \not\doteq T_1) \to \\
& \quad\quad\quad of\ (E\ x)\ T_2).
\end{aligned}
$$

Consider now a predicate which counts the number of bound variables in a lambda-term:

$$
\begin{aligned}
cntlam \quad : \quad & \forall E\!:\!exp \to exp.\, \forall N\!:\!nat. \\
& cnt\ (lam\ E)\ N \\
& \quad \leftarrow (\forall x\!:\!exp.\, cnt\ x\ s(0) \to \\
& \quad\quad cnt\ (E\ x)\ N).
\end{aligned}
$$

The call to $x{:}exp \vdash \mathrm{Not}_\alpha(cnt\ x\ s(0))$ leads to $\mathrm{Not}_x^1(cnt\ x\ s(0))$, since, again, $\Gamma \vdash \mathrm{Not}_x^C$ does not contribute: Not_x^1 collects the term complement of $s(0)$ in the context, $x : exp$, but the typing discipline constrains the result to be $\{0, s(s(M))\}$, as expected:

$$
\begin{aligned}
aug_{\mathrm{D}}(cntlam) \quad : \quad & \forall E\!:\!exp \to exp.\, \forall N\!:\!nat. \\
& cnt\ (lam\ E)\ N \\
& \quad \leftarrow (\forall x\!:\!exp.\, (\neg cnt\ x\ 0 \wedge \forall M\!:\!nat.\, \neg cnt\ x\ s(s(M))) \\
& \quad\quad \to cnt\ (E\ x)\ N).
\end{aligned}
$$

Let us see how rule $\text{Not}_\alpha\top$ enters the picture: recall the $\top$-normalized `linx` lambda clause:

$$\begin{array}{rcl} linxlam & : & \forall E\!:\!exp \to exp \to exp. \\ & & linx\ (\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ & & \quad \leftarrow (\forall z\!:\!exp.\top_{linx} \to linx\ (\lambda x\,.\,E\ x\ z)). \end{array}$$

The judgment $z{:}exp;\top_{linx} \vdash aug_{\text{G}}(linx\ E\ x\ z)$ triggers the rule $\text{Not}_\alpha\top$; in turn $\cdot \vdash sh(z, exp \to exp) = \lambda y\,.\,z$ and thus $z{:}exp \vdash \text{Not}_1^z(\top_{linx}) = \neg linx\ (\lambda x\,.\,z)$:

$$\begin{array}{rcl} aug_{\text{D}}(linxlam) & : & \forall E\!:\!exp \to exp \to exp. \\ & & linx\ (\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \\ & & \quad \leftarrow (\forall z\!:\!exp.\neg linx\ (\lambda x\,.\,z) \to linx\ (\lambda x\,.\,E\ x\ z)). \end{array}$$

Example 6.26 *Let us apply the complement algorithm to the* `linx` *predicate definition; note the vacuous application* $E_2\ x^0$*:*

$$\begin{array}{rcl} linxx & : & linx(\lambda x\,.\,x). \\ linxap1 & : & linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^0)) \leftarrow linx(\lambda x\,.\,E_1\ x). \\ linxap2 & : & linx(\lambda x\,.\,app\ (E_1\ x^0)\ (E_2\ x)) \leftarrow linx(\lambda x\,.\,E_2\ x). \\ aug_{\text{D}}(linxlm) & : & linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \leftarrow (\forall y\!:\!exp.\top_{linx} \to linx(\lambda x\,.\,E\ x\ y)). \end{array}$$

$$\begin{array}{l} \text{Not}_{\text{D}}(def(linx)) = \\ \text{Not}_{\text{D}}(linxx) \vee \text{Not}_{\text{D}}(linxap1) \vee \text{Not}_{\text{D}}(linxap2) \vee \text{Not}_{\text{D}}(linxlm) = \\ \quad (\neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y)))) \vee \\ \quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^1)) \\ \quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^0)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x)) \vee \\ \quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x^1)\ (E_2\ x)) \\ \quad \leftarrow \neg linx(\lambda x\,.\,app\ (E_1\ x^0)\ (E_2\ x)) \leftarrow \neg linx(\lambda x\,.\,E_2\ x)) \vee \\ \quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \\ \quad \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \leftarrow (\forall y\!:\!exp.\neg linx(\lambda x\,.\,y) \to \neg linx(\lambda x\,.\,E\ x\ y))). \end{array}$$

We prove that augmented clauses and goals satisfy the augmented schema, where we define $\Gamma; aug_{\text{D}}(\mathcal{D})$ as $\Gamma; \mathcal{D} \wedge \neg\mathcal{D}$, for $\Gamma \vdash \text{Not}_\alpha(\mathcal{D}) = \neg\mathcal{D}$ and

$$\frac{}{aug(\circ) = \circ}\ aug\circ$$

$$\frac{aug(\mathcal{S}) = \mathcal{S}^a \qquad aug_{\text{D}}(\mathcal{D}) = \mathcal{D}^a}{aug(\mathcal{S}\|(\Gamma;\text{SOME }\Phi.\,\mathcal{D})) = \mathcal{S}^a\|(\Gamma;\text{SOME }\Phi.\,\mathcal{D}^a)}\ aug\|$$

Now, the usual lemma on schema membership:

Lemma 6.27 *If* $\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S}$ *then* $\Gamma \vdash \Gamma'; aug_{\text{D}}(\mathcal{D}') \in aug(\mathcal{S})$*.*

Proof: By induction on the structure of $\pi :: \Gamma; \mathcal{D} \in \mathcal{S}$.

Case:

$$\frac{\Gamma \vdash \theta : \Phi \qquad (\Gamma'; \mathcal{D}') \equiv_\alpha (\Gamma''; \theta\mathcal{D})}{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S}\|(\Gamma''; \text{SOME }\Phi.\,\mathcal{D})}\ \in_1$$

$\Gamma'; \mathcal{D}' \equiv_\alpha \Gamma''; [\theta]\mathcal{D}$ By sub-derivation
$\Gamma'; aug_{\mathrm{D}}(\mathcal{D}') \equiv_\alpha \Gamma''; [\theta]aug_{\mathrm{D}}(\mathcal{D})$ By definition
$\Gamma'; aug_{\mathrm{D}}(\mathcal{D}') \in aug(\mathcal{S}) \| \Gamma''; \mathrm{SOME}\ \Phi.\, aug_{\mathrm{D}}(\mathcal{D})$ By rule $\in_1$
$\Gamma'; aug_{\mathrm{D}}(\mathcal{D}') \in aug(\mathcal{S} \| \Gamma''; \mathrm{SOME}\ \Phi.\, \mathcal{D})$ By rule $aug\|$

Case: π ends in $\in_2$: by IH.

□

We use $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D)$ for $\vdash aug_{\mathrm{D}}(D) = D^a$ and $\models_{\mathcal{S}^a} D^a$; analogously for $aug_{\mathrm{G}}(G)$.

Lemma 6.28 (Schema Augmentation) *Let $\Gamma; \mathcal{D} < \mathcal{S}$ and $aug(S) = \mathcal{S}^a$. Then:*

1. *If $\models_{\mathcal{S}} D$, then $\models_{\mathcal{S}^a} aug_{D}(D)$.*
2. *If $\Gamma; \mathcal{D} \setminus G < \mathcal{S}$, then $\Gamma; \mathcal{D} \setminus aug_{G}(G) < \mathcal{S}^a$.*
3. *If $\Gamma; \mathcal{D} < \mathcal{S}$ then $\Gamma; aug_{D}(\mathcal{D}) < \mathcal{S}^a$.*

Proof: By mutual induction on the structure of $\pi :: \models_{\mathcal{S}} D$, $\gamma :: \Gamma; \mathcal{D} \setminus G < \mathcal{S}$ and $\sigma :: \Gamma; \mathcal{D} < \mathcal{S}$. We show the crucial cases:

Case:

$$\pi = \frac{X \in \{Q, \top, \bot\}}{\models_{\mathcal{S}} X} \models_{\mathcal{S}} X$$

Immediately $\Gamma; \mathcal{D} \vdash aug_{\mathrm{D}}(X) = X$ and

$$\pi' = \frac{X \in \ \{Q, \top, \bot\}}{\models_{\mathcal{S}^a} X} \models_{\mathcal{S}} X$$

Case:

$$\pi = \frac{\cdot; \top \setminus [\rho]G < \mathcal{S}}{\models_{\mathcal{S}} \forall(G \to Q)} \models_{\mathcal{S}} \to^{\sigma}$$

$\cdot; \top \setminus [\rho]G < \mathcal{S}$ By sub-derivation
$\cdot; \top \setminus aug_{\mathrm{G}}([\rho]G) < \mathcal{S}^a$ By IH 2
$\models_{\mathcal{S}^a} aug_{\mathrm{G}}([\rho]G) \to [\rho]Q$ By rule $\models_{\mathcal{S}}\to$
$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(\forall(G \to Q))$ By rule $aug_{\mathrm{D}} \to$

Case:

$$\gamma = \frac{X \in \{\top, \bot\} \text{ or (dis)eq} \qquad \Gamma; \mathcal{D} < \mathcal{S}}{\Gamma; \mathcal{D} \setminus X < \mathcal{S}} \setminus X$$

$\Gamma; \mathcal{D} < \mathcal{S}$ By sub-derivation
$\Gamma; aug_{\mathrm{D}}(\mathcal{D}) < \mathcal{S}^a$ By IH 3
$\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(X) = X$ By rule $aug_{\mathrm{G}}X$
$\Gamma; aug_{\mathrm{D}}(\mathcal{D}) \setminus X < \mathcal{S}^a$ By rule $\setminus X$

Case:

$$\gamma = \frac{\Gamma; \mathcal{D} < \mathcal{S}}{\Gamma; \mathcal{D} \setminus Q < \mathcal{S}} \setminus \mathrm{At}$$

By sub-derivation $\Gamma; \mathcal{D} < \mathcal{S}$; there are two ways to augment an atom:

Subcase: $\cdot; \top \vdash aug_{\mathrm{G}}(Q) = Q$:

$\cdot; \top < \mathcal{S}^a$ By rule $<_1$

$\cdot; \top \setminus Q < \mathcal{S}^a$ By rule $aug_{\mathrm{G}}\mathrm{At}\top$

$\cdot; \top \setminus aug_{\mathrm{G}}(Q) < \mathcal{S}^a$ By rule $\setminus\mathrm{At}$

Subcase: $\Gamma; \mathcal{D} \vdash aug(Q) = \mathrm{Not}_\alpha(\mathcal{D}) \to Q$:

$\Gamma; aug_{\mathrm{D}}(\mathcal{D}) < \mathcal{S}^a$ By IH 3

$\Gamma; aug_{\mathrm{D}}(\mathcal{D}) \setminus Q < \mathcal{S}^a$ By rule $\setminus\mathrm{At}$

$\Gamma; \mathcal{D} \wedge \mathrm{Not}_\alpha(\mathcal{D}) \setminus Q < \mathcal{S}^a$ By def. of $aug_{\mathrm{D}}(\mathcal{D})$

$\Gamma; \mathcal{D} \setminus \ \mathrm{Not}_\alpha(\mathcal{D}) \to Q < \mathcal{S}^a$ By rule $aug_{\mathrm{G}}\mathrm{At}$

$\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(Q) < \mathcal{S}^a$ By rule $\setminus \to$

Case:

$$\gamma = \frac{\models_{\mathcal{S}} D \qquad \Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}}{\Gamma; \mathcal{D} \setminus D \to G < \mathcal{S}} \setminus \to$$

$\models_{\mathcal{S}} D$ By sub-derivation

$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D)$ By IH 1

$\Gamma; \mathcal{D} \wedge D \setminus G < \mathcal{S}$ By sub-derivation

$\Gamma; \mathcal{D} \wedge D \setminus aug_{\mathrm{G}}(G) < \mathcal{S}^a$ By IH 2

$\Gamma; \mathcal{D} \setminus D \to aug_{\mathrm{G}}(G) < \mathcal{S}^a$ By rule $\setminus \to$

$\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(D \to G) < \mathcal{S}^a$ By rule $aug_{\mathrm{G}} \to$

Case:

$$\sigma = \frac{}{\cdot; \top < \mathcal{S}} <_1$$

Then immediately:

$$\sigma' = \frac{}{\cdot; \top < \mathcal{S}^a} <_1$$

Case:

$$\sigma = \frac{\Gamma \vdash (\Gamma'; \mathcal{D}') \in \mathcal{S} \quad \models_{\mathcal{S}} \mathcal{D}' \quad (\Gamma; \mathcal{D}) < \mathcal{S}}{(\Gamma, \lceil\Gamma'\rceil); (\mathcal{D} \wedge \lceil\mathcal{D}'\rceil) < \mathcal{S}} <_2$$

$\Gamma; \mathcal{D} < \mathcal{S}$ By sub-derivation

$\Gamma; aug_{\mathrm{D}}(\mathcal{D}) < \mathcal{S}^a$ By IH 3

$\Gamma \vdash \Gamma'; \mathcal{D}' \in \mathcal{S}$ By sub-derivation

$\Gamma'; aug_{\mathrm{D}}(\mathcal{D}') \in \mathcal{S}^a$ By Lemma 6.27

$\models_{\mathcal{S}} \mathcal{D}'$ By sub-derivation

$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D')$ By IH 1

$(\Gamma, \lceil\Gamma'\rceil); aug_{\mathrm{D}}(\mathcal{D}) \wedge aug_{\mathrm{D}}(\lceil\mathcal{D}')\rceil) < \mathcal{S}^a$ By rule $<_2$

$(\Gamma, \lceil\Gamma'\rceil); aug_{\mathrm{D}}(\mathcal{D} \wedge \lceil\mathcal{D}'\rceil) < \mathcal{S}^a$ By rule $aug_{\mathrm{D}}\wedge$

□

The following Lemma ensures that augmented clauses are closed under negation. In particular, for $\Gamma; \mathcal{D} < \mathcal{S}^a$, if $D \in \mathcal{D}$, so is $\Gamma \vdash \mathrm{Not}_\alpha(D)$. We use $\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(G) < \mathcal{S}^a$ for $\mathrm{Not}_{\mathrm{G}}(G) = G_\neg$ and $\Gamma; \mathcal{D} \setminus G_\neg < \mathcal{S}^a$. Similarly for $\mathrm{Not}_{\mathrm{D}}(D)$. We will need the following technical remark:

Remark 6.29 *If $\vec{M}$ is a simple term, then* $\models_{\mathcal{S}^a} \bigwedge_{\vec{N} \in \mathrm{Not}(\vec{M})} \forall(\neg(q\ \vec{N}) \leftarrow \top)$.

Proof: By rules $\models_{\mathcal{S}^a} \wedge, \models_{\mathcal{S}} \forall, \models_{\mathcal{S}^a} \to, \setminus\top, \models_{\mathcal{S}} \mathrm{At}, \setminus\mathrm{At}$. □

Lemma 6.30 (Negative Schema Augmentation) *Let $\Gamma; \mathcal{D} < \mathcal{S}^a$. Then:*

1. *If $\models_{\mathcal{S}^a} aug_D(D)$, then $\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_D(D))$.*

2. *If* $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(G) < \mathcal{S}^a$*, then* $\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(G)) < \mathcal{S}^a$.

Proof: By mutual induction on the structure of $\pi :: \models_{\mathcal{S}^a} aug_{\mathrm{D}}(D)$ and $\gamma :: \Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(G) < \mathcal{S}^a$.

Case:

$$\pi = \frac{}{\models_{\mathcal{S}^a} aug_{\mathrm{D}}(\bot)} \models_{\mathcal{S}^a} \bot$$

By rule $aug_{\mathrm{D}}(\bot) = \bot$ and $\vdash \mathrm{Not}_{\mathrm{D}}(\bot) = \top$: thus

$$\pi' = \frac{}{\models_{\mathcal{S}^a} \top} \models_{\mathcal{S}^a} \top$$

Case:

$$\pi = \frac{}{\models_{\mathcal{S}^a} aug_{\mathrm{D}}(\top)} \models_{\mathcal{S}^a} \top$$

By rule $aug_{\mathrm{D}}(\top) = \top$ and $\vdash \mathrm{Not}_{\mathrm{D}}(\top) = \bot$: thus

$$\pi' = \frac{}{\models_{\mathcal{S}^a} \bot} \models_{\mathcal{S}^a} \bot$$

Case: π ends in $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(\forall(q\ \vec{M} \leftarrow G))$:

$\models_{\mathcal{S}^a} (\forall(q\ \vec{M}) \leftarrow aug_{\mathrm{G}}(G))$	By rule $aug_{\mathrm{D}} \rightarrow$
$\cdot; \top \setminus aug_{\mathrm{G}}([\rho]G) < \mathcal{S}^a$	By inversion on rule $\models_{\mathcal{S}}\rightarrow$
$\cdot; \top \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}([\rho]G)) < \mathcal{S}^a$	By IH 2
$\models_{\mathcal{S}^a} [\rho](\neg q\ \vec{M})$	By rule $\models_{\mathcal{S}^a}$ At
$\models_{\mathcal{S}^a} \forall(\neg q\ \vec{M} \leftarrow \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(G)))$	By rule $\models_{\mathcal{S}^a}\rightarrow$
$\models_{\mathcal{S}^a} \bigwedge_{\vec{N} \in \mathrm{Not}(\vec{M})} \forall(\neg(q\ \vec{N}))$	By Remark 6.29
$\models_{\mathcal{S}^a} \bigwedge_{\vec{N} \in \mathrm{Not}(\vec{M})} \forall(\neg(q\ \vec{N}) \leftarrow \top) \wedge \forall(\neg q\ \vec{M} \leftarrow \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(G)))$	By rule $\models_{\mathcal{S}^a} \wedge$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(\forall(q\ \vec{M}) \leftarrow aug_{\mathrm{G}}(G))$	By rule $\mathrm{Not}_{\mathrm{D}} \rightarrow$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(\forall(q\ \vec{M} \leftarrow G)))$	By rule $aug_{\mathrm{D}} \rightarrow$

Case: π ends in $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1 \wedge D_2)$:

$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1) \wedge aug_{\mathrm{D}}(D_2)$	By rule $aug_{\mathrm{D}}\wedge$
$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1)$ and $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_2)$	By rule $\models_{\mathcal{S}} \wedge$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1))$ and $\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_2))$	By IH 1
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1)) \vee \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_2))$	By rule $\models_{\mathcal{S}} \vee$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1) \wedge aug_{\mathrm{D}}(D_2))$	By rule $\mathrm{Not}_{\mathrm{D}}\wedge$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1 \wedge D_2))$	By rule $aug_{\mathrm{D}}\wedge$

Case: π ends in $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1 \vee D_2)$:

$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1) \vee aug_{\mathrm{D}}(D_2)$	By rule $aug_{\mathrm{D}}\vee$
$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_1)$ and $\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D_2)$	By rule $\models_{\mathcal{S}} \vee$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1))$ and $\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_2))$	By IH 1
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1)) \wedge \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_2))$	By rule $\models_{\mathcal{S}} \wedge$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1) \vee aug_{\mathrm{D}}(D_2))$	By rule $\mathrm{Not}_{\mathrm{D}}\vee$
$\models_{\mathcal{S}^a} \mathrm{Not}_{\mathrm{D}}(aug_{\mathrm{D}}(D_1 \vee D_2))$	By rule $aug_{\mathrm{D}}\vee$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(\top) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus \top < \mathcal{S}^a$	By rule $aug_{\mathrm{G}}\top$
$\Gamma; \mathcal{D} < \mathcal{S}^a$	By sub-derivation
$\Gamma; \mathcal{D} \vdash \mathrm{Not}_{\mathrm{G}}(\top) = \bot$	By rule $\mathrm{Not}_{\mathrm{G}}\top$
$\Gamma; \mathcal{D} \setminus \bot < \mathcal{S}^a$	By rule $\setminus\bot$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(\bot) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus \bot < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}}\bot$
$\Gamma; \mathcal{D} < \mathcal{S}^a$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash \mathrm{Not}_{\mathrm{G}}(\bot) = \top$ — By rule $\mathrm{Not}_{\mathrm{G}}\bot$
$\Gamma; \mathcal{D} \setminus \top < \mathcal{S}^a$ — By rule $\setminus\bot$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(\vec{M} \doteq \vec{N}) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus \vec{M} \doteq \vec{N} < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}} \doteq$
$\Gamma; \mathcal{D} < \mathcal{S}^a$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash \mathrm{Not}_{\mathrm{G}}(\vec{M} \doteq \vec{N}) = (\vec{M} \dot{\neq} \vec{N})$ — By rule $\mathrm{Not}_{\mathrm{G}} \doteq$
$\Gamma; \mathcal{D} \setminus \vec{M} \dot{\neq} \vec{N} < \mathcal{S}^a$ — By rule $\setminus \dot{\neq}$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(\vec{M} \dot{\neq} \vec{N}) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus \vec{M} \dot{\neq} \vec{N} < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}} \dot{\neq}$
$\Gamma; \mathcal{D} < \mathcal{S}^a$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash \mathrm{Not}_{\mathrm{G}}(\vec{M} \dot{\neq} \vec{N}) = (\vec{M} \doteq \vec{N})$ — By rule $\mathrm{Not}_{\mathrm{G}} \dot{\neq}$
$\Gamma; \mathcal{D} \setminus \vec{M} \doteq \vec{N} < \mathcal{S}^a$ — By rule $\setminus \dot{\neq}$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(Q) < \mathcal{S}^a$:
By sub-derivation $\Gamma; \mathcal{D} < \mathcal{S}^a$; there are two ways to augment an atom:

Subcase: $\cdot; \top \vdash aug_{\mathrm{G}}(Q) = Q$:

$\cdot; \top \setminus \ Q < \mathcal{S}^a$ — By rule $\setminus$At
$\cdot; \top \setminus \neg Q < \mathcal{S}^a$ — By rule $\mathrm{Not}_{\mathrm{G}}$At
$\cdot; \top \setminus \mathrm{Not}_{\mathrm{G}}(Q) < \mathcal{S}^a$ — By rule $\mathrm{Not}_{\mathrm{G}} \to$
$\cdot; \top \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(Q)) < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}}\mathrm{At}\top$

Subcase: $\Gamma; \mathcal{D} \vdash alg(Q) = \mathrm{Not}_{\alpha}(\mathcal{D}) \to G$:

$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\alpha}(\mathcal{D}) \to Q < \mathcal{S}^a$ — By rule aug_{G}At
$\Gamma; (\mathcal{D} \wedge \mathrm{Not}_{\alpha}(\mathcal{D})) \setminus Q < \mathcal{S}^a$ — By rule $\setminus \to$
$\Gamma; (\mathcal{D} \wedge \mathrm{Not}_{\alpha}(\mathcal{D})) < \mathcal{S}^a$ — By sub-derivation
$\Gamma; (\mathcal{D} \wedge \mathrm{Not}_{\alpha}(\mathcal{D})) \setminus \neg Q < \mathcal{S}^a$ — By rule $\setminus$At
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\alpha}(\mathcal{D}) \to \neg Q < \mathcal{S}^a$ — By rule $\setminus \to$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\alpha}(\mathcal{D}) \to \mathrm{Not}_{\mathrm{G}}(Q) < \mathcal{S}^a$ — By rule $\mathrm{Not}_{\mathrm{G}}$At
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(\mathrm{Not}_{\alpha}(\mathcal{D}) \to Q) < \mathcal{S}^a$ — By rule $\mathrm{Not}_{\mathrm{G}} \to$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(Q)) < \mathcal{S}^a$ — By rule aug_{G}At

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(D \to G) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus aug_{\mathrm{D}}(D) \to aug_{\mathrm{G}}(G) < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}} \to$
$\models_{\mathcal{S}^a} aug_{\mathrm{D}}(D)$ and $\Gamma; (\mathcal{D} \wedge D) \setminus aug_{\mathrm{G}}(G) < \mathcal{S}^a$ — By rule $\setminus \to$
$\Gamma; (\mathcal{D} \wedge aug_{\mathrm{D}}(D)) \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(G)) < \mathcal{S}^a$ — By IH 2
$\Gamma; \mathcal{D} \setminus aug_{\mathrm{D}}(D) \to \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(G)) < \mathcal{S}^a$ — By rule $\setminus \to$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{D}}(D) \to aug_{\mathrm{G}}(G)) < \mathcal{S}^a$ — By rule $\mathrm{Not}_{\mathrm{G}} \to$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\mathrm{G}}(aug_{\mathrm{G}}(D \to G)) < \mathcal{S}^a$ — By rule $aug_{\mathrm{G}} \to$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\mathrm{G}}(\forall x{:}a.\, G) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus \forall x{:}a.\, aug_{\rm G}(G) < \mathcal{S}^a$ By rule $aug_{\rm G}\forall$
$(\Gamma, y{:}A); \mathcal{D} \setminus [y/x]aug_{\rm G}(G) < \mathcal{S}^a$ By rule $\setminus\forall$
$(\Gamma, y{:}A); \mathcal{D} \setminus [y/x]\mathrm{Not}_{\rm G}(aug_{\rm G}(G)) < \mathcal{S}^a$ By IH 2
$\Gamma; \mathcal{D} \setminus \forall x{:}a.\, \mathrm{Not}_{\rm G}(aug_{\rm G}(G)) < \mathcal{S}^a$ By rule $\setminus\forall$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(\forall x{:}a.\, aug_{\rm G}(G)) < \mathcal{S}^a$ By rule $\mathrm{Not}_{\rm G}\forall$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(\forall x{:}a.\, G)) < \mathcal{S}^a$ By rule $aug_{\rm G}\forall$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1 \vee G_2) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1) \vee aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $aug_{\rm G}\vee$
$\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1) < \mathcal{S}^a$ and $\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $\setminus\vee$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1)) < \mathcal{S}^a$ and $\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_2)) < \mathcal{S}^a$ By IH 2
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1)) \wedge \mathrm{Not}_{\rm G}(aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $\setminus\wedge$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1) \vee aug_{\rm G}(G_2)) < \mathcal{S}^a$ By rule $\mathrm{Not}_{\rm G}\vee$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1 \vee G_2)) < \mathcal{S}^a$ By rule $aug_{\rm G}\vee$

Case: γ ends in $\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1 \wedge G_2) < \mathcal{S}^a$:

$\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1) \wedge aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $aug_{\rm G}\wedge$
$\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_1) < \mathcal{S}^a$ and $\Gamma; \mathcal{D} \setminus aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $\setminus\wedge$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1)) < \mathcal{S}^a$ and $\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_2)) < \mathcal{S}^a$ By IH 2
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1)) \vee \mathrm{Not}_{\rm G}(aug_{\rm G}(G_2) < \mathcal{S}^a$ By rule $\setminus\vee$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1) \wedge aug_{\rm G}(G_2)) < \mathcal{S}^a$ By rule $\mathrm{Not}_{\rm G}\wedge$
$\Gamma; \mathcal{D} \setminus \mathrm{Not}_{\rm G}(aug_{\rm G}(G_1 \wedge G_2)) < \mathcal{S}^a$ By rule $aug_{\rm G}\wedge$

□

We can combine the two latter lemmata 6.28 and 6.30 to prove that augmentation guarantees that static and dynamic clauses are closed under complementation.

Corollary 6.31 (Closure under Complementation) *If* $\models_{\mathcal{S}} D$*, then* $\models_{\mathcal{S}^a} \mathrm{Not}_{\rm D}(aug_{\rm D}(D))$.

6.7 Exclusivity

We are now in the position to prove the main result of this Chapter, namely that clause complementation satisfies the boolean rules of negation, in the form of exclusivity and exhaustivity. We remark that this holds due to the fact that context schemata allow to pose only 'well-behaved' goals. For example, consider the query $G \equiv \neg even(0) \rightarrow even(s(s(0)))$, which is such that both $\cdot; \top \vdash_{even} G$ and $\cdot; \top \vdash_{\neg even} \mathrm{Not}_{\rm G}(G)$ are provable. By definition of schema satisfaction, this is not a legal query. Similarly, the following counter-example to exhaustivity $\cdot; \top \vdash_{even} \forall x : nat.\, even(x)$ is not allowed. Moreover, the Context Preservation Theorems (Theorem 6.13 and 6.14) guarantees that, from allowed run-time contexts and goals, only allowed subgoals are generated.

We use $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\rm G}(G)$ for $\Gamma \vdash \mathrm{Not}_{\rm G}(G) = G'$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G'$; also, $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\rm D}(D) \gg \neg Q$ for $\mathrm{Not}_{\rm D}(D) = D'$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D' \gg \neg Q$; similarly for Not_{α}. We use $\mathcal{P}$ to denote the conjunction of a positive program $\mathcal{P}^+$ and its negation $\mathcal{P}^- = \mathrm{Not}_{\rm D}(\mathcal{P}^+)$.

Theorem 6.32 (Exclusivity) *Let* $\models_{\mathcal{S}^a} aug_{\rm D}(\mathcal{P})$ *and* $\Gamma; \mathcal{D} < \mathcal{S}^a$*. For every goals such that* $\Gamma; \mathcal{D} \setminus G < \mathcal{S}^a$*, It is not the case that there is* $\mathcal{S}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} G$ *and there is* $\mathcal{S}^- :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\rm G}(G)$.

Proof: We generalize this to:

$\boxed{2.1}$ Let $D \sqsubseteq \mathrm{def}(q, \mathcal{P})$: It is not the case that $\mathcal{I}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\mathcal{I}^- :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\rm D}(D) \gg \mathrm{Not}_{\rm G}(Q)$. By mutual induction on the structure of $\mathcal{I}^+$ and $\mathcal{S}^+$, by assuming their existence:

Case:

$$\mathcal{I}^+ = \frac{}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bot_q \gg Q} \gg\bot$$

But there can be no proof of $\neg Q$ from $\mathrm{Not_D}(\bot_q) = \top_q$.

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n} \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M_n}} \gg\to$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(\forall(q\ \overline{N_n} \leftarrow G)) \gg \neg q\ \overline{M_n}$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n}$	By sub-derivation
$\cdot \vdash \overline{M_n} \in \lVert\overline{N_n}\rVert$ $(*)$	By definition of of $\lVert\_\rVert$
$\mathcal{S}_1^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})} \forall(\neg q\ \overline{S_n} \leftarrow \top) \wedge \forall(\neg q\ \overline{N_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$	By rule $\mathrm{Not_D} \to$
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})} \forall(\neg q\ \overline{S_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$	By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{S_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$ for some $\overline{S_n}$	By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta](\neg q\ \overline{S_n}) \leftarrow \top \gg \neg q\ \overline{M_n}$ for some θ	By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]\overline{S_n} \doteq \overline{M_n}$	By inversion
$\cdot \vdash \overline{M_n} \in \lVert\mathrm{Not}(\overline{N_n})\rVert$	By definition of $\lVert\_\rVert$
$\bot$	By term exclusivity with $(*)$
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{N_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$	By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}([\sigma]G)$	By inversion
$\bot$	By IH 1 w.r.t. $\mathcal{S}_1^+$

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (D_1 \vee D_2) \gg Q} \gg\vee$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1 \vee D_2) \gg \neg Q$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \wedge \mathrm{Not_D}(D_2) \gg \neg Q$	By rule $\mathrm{Not_D}\vee$
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \gg \neg Q$	By inversion
$\bot$	By IH 2.1
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_2) \gg \neg Q$	By inversion
$\bot$	By IH 2.1

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1 \wedge D_2) \gg \neg Q$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \vee \mathrm{Not_D}(D_2) \gg \neg Q$	By rule $\mathrm{Not_D}\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1)$	By inversion
$\bot$	By IH 2.1

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1 \wedge D_2) \gg \neg Q$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \vee \mathrm{Not_D}(D_2) \gg \neg Q$ By rule $\mathrm{Not_D}\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_2) \gg \neg Q$ By inversion
$\perp$ By IH 2.1

$\boxed{2.2}$ Let $D \sqsubseteq \mathrm{def}(q, \mathcal{P})$ and $D' \sqsubseteq \mathrm{def}(q, \mathcal{D})$. It is not the case that $\mathcal{I}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\mathcal{I}^- :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D') \gg \neg Q$. By simultaneous induction on the structure of $\mathcal{I}^+$ and $\mathcal{I}^-$:

No case for $\mathcal{I}^+$ ends in $\gg\top$

Case:

$$\mathcal{I}^+ = \frac{}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \perp_q \gg Q} \gg\mathrm{At}$$

By schema satisfaction there can be no formula in the dynamic definition of q.

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{S_n} \doteq \overline{M_n} \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{S_n} \leftarrow G) \gg q\ \overline{M_n}} \gg \to$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{T_n} \leftarrow H)) \gg \neg q\ \overline{M_n}$.

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{S_n} \doteq \overline{M_n}$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]S_i \doteq M_i$ for every $1 \leq i \leq n$ $(*)$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{x \in \mathrm{dom}(\Gamma)} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{T_n}) \wedge (\bigwedge_{1 \leq i \leq n, xR^iq} \mathrm{Not}^i_x(q\ \overline{T_n}))) \wedge \forall(\neg q\ \overline{T_n}) \leftarrow \mathrm{Not_G}(G) \gg \neg q\ \overline{M_n}$
By rule $\mathrm{Not}_\alpha \to$

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{T_n}) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \leq j \leq n} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_j]\overline{Z_n} \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ By inversion
$\perp$ From line $(*)$ and S_j rigid

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \leq i \leq n, xR^iq} \mathrm{Not}^i_x(q\ \overline{T_n}) \gg \neg q\ \overline{M_n}$ for a fixed x By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}^i_x(q\ \overline{T_n}) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \leq j \leq n} \bigwedge_{N \in \Gamma \vdash \mathrm{Not}(T_j)} (\forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_i, N/Z_j]\overline{Z_n}) \gg \neg q\ \overline{M_n}$
By rule $\mathrm{Not}^i_x \to$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_i$ By inversion
$\perp$ From line $(*)$ and S_i rigid

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{T_n} \leftarrow \mathrm{Not_G}(H)) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]\overline{T_n} \doteq \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ for some $1 \leq j \leq n$ By complementable clause
$\perp$ From line $(*)$ and S_j rigid

Case: $\mathcal{I}^+$ ends in $\gg \to$, $\mathcal{I}^-$ ends in $\gg\perp$: there can be no proof of $\neg Q$ from $\Gamma \vdash \mathrm{Not}_\alpha(\perp_q) = \top_q$.

Case: $\mathcal{I}^+$ ends in $\gg \to$, $\mathcal{I}^-$ ends in $\gg\top$:

Subcase: $\Gamma = \cdot$ or for all $x \in \mathrm{dom}(\Gamma)$, for all i, $1 \leq i \leq n$ it is not the case that xR^iq: then $\Gamma \vdash \mathrm{Not}_\alpha(\top_q) = \top_q$ and an be no proof of $\neg Q$ from $\top_q$.

Subcase: else $\Gamma \vdash \mathrm{Not}_\alpha(\top_q) = \bigwedge_{x \in dom(\Gamma)}(\bigwedge_{1 \leq i \leq n, xR^iq} \Gamma \vdash \mathrm{Not}^C_x(\top_q))$; fix x, i:
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(\top_q) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \leq i \leq n} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_j]\overline{Z_n} \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ By inversion
$\perp$ Above line and rigidity restriction on S_j $(*)$.

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (D_1 \vee D_2) \gg Q} \gg\vee$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash \mathrm{Not}_\alpha(D') \gg \neg Q$:

$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ By sub-derivation
$\bot$ By IH 2.2

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge_1$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash \mathrm{Not}_{\alpha}(D') \gg \neg Q$:

$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ By sub-derivation
$\bot$ By IH 2.2

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge_2$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash \mathrm{Not}_{\alpha}(D') \gg \neg Q$: Symmetric to the above.

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_1') \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_1' \vee D_2') \gg \neg Q} \gg\wedge_1$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.2

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_2') \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_1' \vee D_2') \gg \neg Q} \gg\wedge_2$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.2.

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_1') \gg \neg Q \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_2') \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D_1' \wedge D_2') \gg \neg Q} \gg\vee$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.2.

$\boxed{2.3}$ Let $D \sqsubseteq \mathrm{def}(q,\mathcal{D})$ and $D' \sqsubseteq \mathrm{def}(q,\mathcal{P})$. It is not the case that $\mathcal{I}^+ :: \Gamma;\mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\mathcal{I}^- :: \Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{D}}(D') \gg \neg Q$. By simultaneous induction on the structure of $\mathcal{I}^+$ and $\mathcal{I}^-$. Note that by the restriction to complementable programs, some SPE e_x occurs in D; we thus use the notation $\overline{N^i_{e_x}}$.

No case for for $\mathcal{I}^+$ ends in $\gg\top$.

No case for for $\mathcal{I}^+$ ends in $\gg\bot$, since by the restriction to complementable clauses, $\bot$ cannot occur as an assumption.

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N^i_{e_x}} \doteq \overline{M_n} \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]H}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{N^i_{e_x}} \leftarrow H) \gg q\ \overline{M_n}} \gg\to$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{D}}(\forall(q\ \overline{S_n} \leftarrow G)) \gg \neg q\ \overline{M_n}$.

$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N^i_{e_x}} \doteq \overline{M_n}$ By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_i$ $(*)$ for some $1 \leq i \leq n$ By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{T_n} \in \cdot\vdash\mathrm{Not}(\overline{S_n})} \forall(\neg q\ \overline{T_n} \leftarrow \top) \wedge \forall(\neg q\ \overline{S_n} \leftarrow \mathrm{Not}_{\mathrm{G}}(G)) \gg \neg q\ \overline{M_n}$ By rule $\mathrm{Not}_{\mathrm{D}} \to$

Subcase: $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{T_n} \in \cdot\vdash\mathrm{Not}(\overline{S_n})} \forall(\neg q\ \overline{T_n} \leftarrow \top) \gg q\ \overline{M_n}$ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{T_n} \leftarrow \top) \gg q\ \overline{M_n}$ for some $\overline{T_n} \in \cdot \vdash \mathrm{Not}(\overline{S_n})$ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\theta](\neg q\ \overline{T_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$ for some θ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\theta]\overline{T_n} \doteq \overline{M_n}$ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\theta]T_i \doteq M_i$ By inversion
$\bot$ $(*)$

Subcase: $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{S_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\theta']\overline{S_n} \doteq \overline{M_n}$ By inversion
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\theta']S_i \doteq M_i$ By inversion
$\perp$ From line $(*)$ and S_i rigid.

No case for $\mathcal{I}^+$ ends in $\gg \rightarrow$, $\mathcal{I}^-$ ends in $\gg\perp$: if $\top_q$ is the only definition for q, then by schema satisfaction $\forall(q\ \overline{N^i_{e_x}} \leftarrow H)$ could not be an assumption.

No case for $\mathcal{I}^+$ ends in $\gg \rightarrow$, $\mathcal{I}^-$ ends in $\gg\perp$: ditto.

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge_1$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash \mathrm{Not_D}(D') \gg \neg Q$:

$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ By sub-derivation
$\perp$ By IH 2.2

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge_2$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash \mathrm{Not_D}(D') \gg \neg Q$: Symmetric to the above.

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_1) \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_1 \vee D'_2) \gg \neg Q} \gg\wedge_1$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.3

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_2) \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_1 \vee D'_2) \gg \neg Q} \gg\wedge_2$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.3.

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_1) \gg \neg Q \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_2) \gg \neg Q}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D'_1 \wedge D'_2) \gg \neg Q} \gg\vee$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash D \gg Q$. By IH 2.3.

$\boxed{2.4}$ Let $D \sqsubseteq \mathrm{def}(q,\mathcal{D})$ and $D' \sqsubseteq \mathrm{def}(q,\mathcal{D})$. There are two main cases, according to whether D and D' belong to the same block (and in this case it suffices to consider $D \equiv D'$) or not:

1. It is not the case that $\mathcal{I}^+ :: \Gamma;\mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\mathcal{I}^- :: \Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D) \gg \neg Q$. By mutual induction on the structure of $\mathcal{I}^+$ and $\mathcal{S}^+$.

Case:

$$\mathcal{I}^+ = \frac{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n} \qquad \Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M_n}} \gg \rightarrow$$

and $\mathcal{I}^-$ ends in $\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{N_n} \leftarrow G)) \gg \neg q\ \overline{M_n}$.
$\mathcal{S}^+_1 :: \Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]G$ By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n}$ By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{x \in \mathrm{dom}(\Gamma)} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{N_n}) \wedge (\bigwedge_{1 \le i \le n, xR^i q} \mathrm{Not}^i_x(q\ \overline{N_n}))) \wedge \forall(\neg q\ \overline{N_n}) \leftarrow \mathrm{Not_G}(G) \gg \neg q\ \overline{M_n}$
By rule $\mathrm{Not}_\alpha \rightarrow$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} [\sigma]N_i \doteq M_i$ $(+)$ for all $1 \le i \le n$ By inversion
$\Gamma \vdash M_i \in \|N_i\|$ $(*)$ for all $1 \le i \le n$ By inversion

Subcase: $[\sigma]N_i \not\equiv e_x$, $[\sigma]N_i$ rigid:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{N_n}) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \le i \le n} \forall Z_1 : A_1. \dots \forall Z_n : A_n. \neg q\ [e_x/Z_j]\overline{Z_n} \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ By inversion

$\perp$ Above line and (+).

Subcase: $[\sigma]N_i \equiv e_x$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{1 \le i \le n, xR^i q} \mathrm{Not}^i_x(q\ \overline{N_n}))) \gg [\sigma]\neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}^i_x(\overline{N_n}) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \le i \le n} \bigwedge_{N \in \Gamma \vdash \mathrm{Not}(M_i)} \forall(\neg q\ [e_x/Z_i, N/Z_j]\overline{Z_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{N \in \Gamma \vdash \mathrm{Not}(M_i)} \forall(\neg q\ [e_x/Z_i, N/Z_j]\overline{Z_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} M_i \doteq [\theta]N$ By inversion

$\Gamma \vdash M_i \in \|N\|$ for some $N \in \Gamma \vdash \mathrm{Not}(N_i)$ By definition of $\|\_\|$

$\perp$ By term exclusivity (4.21) with line $(*)$

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \neg\forall(q\ \overline{N_n} \leftarrow \mathrm{Not}_{\mathrm{G}}(G)) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}([\sigma]G)$ By inversion

$\perp$ By IH 1 on $\mathcal{S}_1^+$

Case: $\mathcal{I}^+$ ends in $\gg\vee, \gg\wedge, \gg\forall$: by IH 2.4.1 similarly to case 2.1 with Not_α in place of $\mathrm{Not}_{\mathrm{D}}$.

2. It is not the case that $\mathcal{I}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\mathcal{I}^- :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D') \gg \neg Q$. By simultaneous induction on the structure of $\mathcal{I}^+$ and $\mathcal{I}^-$:

No case for $\mathcal{I}^+$ ends in $\gg\top$.

No case for $\mathcal{I}^+$ ends in $\gg\perp$, by restriction to complementable clauses and schema satisfaction.

No case for $\mathcal{I}^+$ ends in $\gg\rightarrow$, $\mathcal{I}^-$ ends in $\gg\perp$, as above.

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N^y_i} \doteq \overline{M_n} \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{N^y_i} \leftarrow G) \gg q\ \overline{M_n}} \gg\rightarrow$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{S_n} \leftarrow G')) \gg \neg q\ \overline{M_n}$, from a different block.

$\mathcal{S}_1^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G$ By sub-derivation

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N^y_i} \doteq \overline{M_n}$ By sub-derivation

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{x \in \mathrm{dom}(\Gamma)} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{S_n}) \wedge (\bigwedge_{1 \le i \le n, xR^i q} \mathrm{Not}^i_x(q\ \overline{S_n}))) \wedge \forall(\neg q\ \overline{S_n}) \leftarrow \mathrm{Not}_{\mathrm{G}}(G') \gg \neg q\ \overline{M_n}$

By rule $\mathrm{Not}_\alpha \rightarrow$, $y \not\equiv x$

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]N_i \doteq M_i$ (+) for all $1 \le i \le n$ By inversion

Subcase: $[\sigma]N_i \not\equiv e_y$, $[\sigma]N_i$ rigid:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{S_n}) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \le i \le n} \forall Z_1 : A_1. \dots \forall Z_n : A_n. \neg q\ [e_x/Z_j]\overline{Z_n} \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ By inversion

$\perp$ Above line, N_j rigid and (+).

Subcase: $[\sigma]N_i \equiv e_y$:

$\perp$ Above line, eigenvariable condition and (+).

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \neg\forall(q\ \overline{S_n} \leftarrow \mathrm{Not}_{\mathrm{G}}(G')) \gg \neg q\ \overline{M_n}$ By inversion

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{S_n} \doteq \overline{M_n}$ By inversion

$\perp$ Above line, eigenvariable condition and (+).

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n} \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M_n}} \gg\rightarrow$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\top_q)) \gg \neg q\ \overline{M_n}$, from a different block.

$\mathcal{S}_1^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]G$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n}$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{x \in \mathrm{dom}(\Gamma)}(\bigwedge_{1 \le i \le n, xR^i q} \Gamma \vdash \mathrm{Not}_x^C(\top_q)) \gg \neg q\ \overline{M_n}$
By rule $\mathrm{Not}_\alpha\top$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]N_i \doteq M_i$ (+) for all $1 \le i \le n$ By inversion
$\Gamma \vdash M_i \in \|N_i\|$ (∗) for all $1 \le i \le n$ By inversion
$[\sigma]N_i \not\equiv e_x$, $[\sigma]N_i$ rigid
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}_x^C(q\ \overline{N_n}) \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{1 \le i \le n} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_j]\overline{Z_n} \gg \neg q\ \overline{M_n}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} e_x \doteq M_j$ By inversion
$\bot$ Above line and (+).

Case:

$$\mathcal{I}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \gg\wedge_1$$

and $\mathcal{I}^-$ ends in $\Gamma; \mathcal{D} \vdash \mathrm{Not}_\alpha(D') \gg \neg Q$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D_1 \gg Q$ By sub-derivation
$\bot$ By IH 2.4.2

Case: $\mathcal{I}^+$ ends in $\gg\wedge_1, \gg\vee$, $\mathcal{I}^-$ ends in $\gg\wedge_i, \gg\vee$: analogously by IH 2.4.2.

$\boxed{1}$ By induction on the structure of $\mathcal{S}^+$.

Case:

$$\mathcal{S}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(q, \mathcal{P} \wedge \mathcal{D}) \gg Q}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} Q} \vdash \mathrm{atm}$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(Q)$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(q, \mathcal{P} \wedge \mathcal{D}) \gg Q$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \neg Q$ By rule $\mathrm{Not}_{\mathrm{G}}\mathrm{At}$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(\neg q, \mathcal{P} \wedge \mathcal{D}) \gg \neg Q$ By inversion

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{D}}(\mathrm{def}(q, \mathcal{P}))$ By definition
$\bot$ By IH 2.i, $i \in \{1, 3\}$ according to D static or dynamic
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\mathrm{def}(q, \mathcal{D}))$ By definition
$\bot$ By IH 2.i, $i \in \{2, 4\}$ according to D static or dynamic

Case:

$$\mathcal{S}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \qquad \Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \wedge G_2} \vdash \wedge$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(G_1 \wedge G_2)$

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2$ By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(G_1) \vee \mathrm{Not}_{\mathrm{G}}(G_2)$ By rule $\mathrm{Not}_{\mathrm{G}}\wedge$

Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(G_1)$ By inversion
$\bot$ By IH 1
Subcase: $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(G_2)$ By inversion
$\bot$ By IH 1

Case:

$$\mathcal{S}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \vee G_2} \vdash \vee$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\mathrm{G}}(G_1 \vee G_2)$

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1) \wedge \mathrm{Not_G}(G_2)$ — By rule $\mathrm{Not_G}\vee$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1)$ — By inversion
$\perp$ — By IH 1

Case:

$$\mathcal{S}^+ = \frac{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_2}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} G_1 \vee G_2} \vdash \vee$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1 \vee G_2)$. Symmetrical.

Case:

$$\mathcal{S}^+ = \frac{(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall x{:}a.\, G'} \vdash\forall^*$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(\forall x{:}a.\, G')$:

$(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} [y/x]G'$ — By sub-derivation
$(\Gamma, y{:}A); \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}([y/x]G')$ — By rule $\mathrm{Not_G}\forall$
$\perp$ — By IH 1

Case:

$$\mathcal{S}^+ = \frac{\Gamma; (\mathcal{D} \wedge D') \vdash_{\mathcal{P}} G'}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D' \to G'} \vdash \to$$

and $\mathcal{S}^-$ ends in $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(D' \to G')$:

$\Gamma; (\mathcal{D} \wedge D') \vdash_{\mathcal{P}} G'$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D' \to \mathrm{Not_G}(G')$ — By rule $\mathrm{Not_G} \to$
$\Gamma; (\mathcal{D} \wedge D') \vdash_{\mathcal{P}} \mathrm{Not_G}(G')$ — By inversion
$\perp$ — By IH 1

Case: The (dis)equality case follows from the decidability of the (dis)equality judgment.

□

Note that the proof goes through as there is no 'bad' interaction between the static and dynamic definition of a predicate; namely in sub-case 2.2 there is no overlap between a clause from $\mathrm{def}(q, \mathcal{P})$ and $\mathrm{def}(\neg q, \mathcal{D})$ since in every atomic assumption there must be an occurrence of a new parameter *and* every term at the same position in a program clause head must be *rigid*. Symmetrically for 2.3. Sub-case 2.4 holds analogously to the first case 2.1, which is based on term exclusivity (Corollary 4.21). In the latter case, it suffices to consider, for $D \sqsubseteq \mathrm{def}(q, \mathcal{P})$, only $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} D \gg Q$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D) \gg \neg Q$, because the positive definition is a conjunction, while the negative one is a disjunction; 2.4 needs to discriminate whether the two dynamic assumptions belong to the same block: if so the same remark applies. If not, the (pivot) eigenvariable condition ensures non-overlapping.

The condition for programs to require clause heads with rigid terms may seem too restrictive; the natural way to relax it, requiring that only *one* of those terms which occur in the same position of a parameter in each assumption needs to be rigid, turns out to be inconsistent; in fact, the property needs to be preserved in the negative program as well. Consider the following counterexample:

$$\begin{array}{lcl} plam & : & p\ (lam\ E)\ (lam\ F) \leftarrow \forall x{:}exp.\, p\ (E\ x)\ (F\ x). \\ papp & : & p\ (app\ E_1\ E_2)\ F. \end{array}$$

Augmentation will insert the clause $\forall(\neg p\ x\ H \wedge \neg p\ H'\ x)$, which now overlaps with $\neg$`papp`. As a result both $p\ (lam(\lambda x{:}exp.\, app\ x\ x))\ (lam(\lambda y{:}exp.\, y))$ and its negation are provable.

As we remarked earlier, the rigidity condition applies only to predicate definitions which are mutually recursive to non-Horn ones. Thus, any such program which uses a 'catch-all' clause is those positions is forbidden; nevertheless it is easy to avoid catch-all clauses via explicit coercion or (sometimes) partial evaluation. Finally, we remark that any other (decidable) condition which avoids overlap between static and dynamic clauses will do; the one we have proposed is statically checkable, certainly not 'ad hoc' and has been dictated by the practice of logical frameworks.

6.8 Exhaustivity

Theorem 6.33 (Exhaustivity) *Let* $\models_{\mathcal{S}^a} aug_D(\mathcal{P})$ *and* $\Gamma; \mathcal{D} < \mathcal{S}^a$. *For every goal* G *such that* $\Gamma; \mathcal{D} \backslash G < \mathcal{S}^a$, *if* $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G$ *then* $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G)$.

Proof: Note that $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(\neg q, \mathcal{P} \wedge \mathcal{D}) \gg \neg Q$ iff so does every disjunct. We generalize this to:

1. If $\mathcal{S}^- :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} G$, then $\mathcal{S}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G)$.

2.1 If $D \sqsubseteq \mathrm{def}(Q, \mathcal{P})$ $\mathcal{I}^- :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$, then $\mathcal{I}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D) \gg \neg Q$.

2.2 If $D \sqsubseteq \mathrm{def}(Q, \mathcal{D})$ $\mathcal{I}^- :: \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D \gg Q$, then $\mathcal{I}^+ :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_\alpha}(D) \gg \neg Q$.

The proof is by mutual induction on the structure of $\mathcal{S}^-$ and $\mathcal{I}^-$. We start with part 2.1:

$\boxed{2.1}$

Case:

$$\mathcal{I}^- = \frac{}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \top \gg Q} \not\gg \top$$

Then $\mathrm{Not_D}(\top) = \bot$ and

$$\mathcal{I}^+ = \frac{}{\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bot \gg \neg Q} \gg \bot$$

Case:

$$\mathcal{I}^- = \frac{\text{for all } \theta \ \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\theta]\overline{N_n} \doteq \overline{M_n}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall (q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M_n}} \not\gg \rightarrow^1$$

for all θ $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\theta]\overline{N_n} \doteq \overline{M_n}$	By sub-derivation
for all θ $[\theta]\overline{N_n} \neq \overline{M_n}$	By rule $\not\vdash\doteq$
$\cdot \vdash \overline{M_n} \notin \lVert\overline{N_n}\rVert$	By definition of $\lVert\_\rVert$
$\cdot \vdash \overline{M_n} \in \lVert\mathrm{Not}(\overline{N_n})\rVert$	By term exhaustivity (4.21)
$\overline{M_n} = [\sigma]\overline{S_n}$ for some σ and $\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})$	By definition of $\lVert\_\rVert$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \overline{M_n} \doteq [\sigma]\overline{S_n}$	By rule $\vdash\doteq$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{S_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$	By rule $\gg\forall$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})} \forall(\neg q\ \overline{S_n} \leftarrow \top) \gg \neg q\ \overline{M_n}$	By appropriate applications of rule $\gg\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})} \forall(\neg q\ \overline{S_n} \leftarrow \top) \wedge \forall(\neg q\ \overline{M_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$	By rule $\gg\wedge_1$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(\forall(q\ \overline{N_n} \leftarrow G) \gg \neg q\ \overline{M_n})$	By rule $\mathrm{Not_D}\gg$

Case:

$$\mathcal{I}^- = \frac{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]\overline{N_n} \not\doteq \overline{M_n} \qquad \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]G}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall (q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M_n}} \not\gg \rightarrow_2$$

$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]\overline{N_n} \not\doteq \overline{M_n}$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\sigma]\overline{N_n} \doteq \overline{M_n}$	By rule $\vdash\not\doteq$
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\sigma]G$	By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}[\sigma](G)$	By IH 1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall(\neg q\ \overline{N_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$	By rule $\gg \rightarrow$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{\overline{S_n} \in \cdot \vdash \mathrm{Not}(\overline{N_n})} \forall(\neg q\ \overline{S_n} \leftarrow \top) \wedge \forall(\neg q\ \overline{N_n} \leftarrow \mathrm{Not_G}(G)) \gg \neg q\ \overline{M_n}$	By rule $\gg\wedge_2$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(\forall(q\ \overline{N_n} \leftarrow G)) \gg \neg q\ \overline{M_n}$	By rule $\mathrm{Not_D} \rightarrow$

Case:

$$\mathcal{I}^- = \frac{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} (D_1 \vee D_2) \gg Q} \not\gg \vee_1$$

$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \gg \neg Q$ — By IH 2.1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \wedge \mathrm{Not_D}(D_2) \gg \neg Q$ — By rule $\gg\wedge_1$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1 \vee D_2) \gg \neg Q$ — By rule $\mathrm{Not_D}\vee$

Case:

$$\mathcal{I}^- = \frac{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} (D_1 \vee D_2) \gg Q} \not\gg\vee_2$$

Symmetrical to the above.

Case:

$$\mathcal{I}^- = \frac{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q \qquad \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_2 \gg Q}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} (D_1 \wedge D_2) \gg Q} \not\gg\wedge$$

$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_1 \gg Q$ — By sub-derivation
$\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} D_2 \gg Q$ — By sub-derivation
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \gg \neg Q$ — By IH 2.1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_2) \gg \neg Q$ — By IH 2.1
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1) \vee \mathrm{Not_D}(D_2) \gg \neg Q$ — By rule $\gg\vee$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D_1 \wedge D_2) \gg \neg Q$ — By rule $\mathrm{Not_D}\wedge$

$\boxed{2.2}$ Recall that $q\ \overline{M^i_{e_x}}$ stands for $q\ M_1 \ldots M_{i-1}\ e_x\ M_{i+1} \ldots M_n$, for $x{:}A \in \Gamma$:

Case:

$$\mathcal{I}^- = \frac{}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \top_q \gg q\ \overline{M^i_{e_x}}} \not\gg\top$$

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\overline{M^i_{e_x}}/\overline{Z_n}]\overline{Z_n} \doteq \overline{M^i_{e_x}}$ — By rule $\vdash\doteq$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_i]\overline{Z_n Z_n} \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\gg\forall$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(\top_q) \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\Gamma \vdash \mathrm{Not}^C_x$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \bigwedge_{x \in dom(\Gamma)} \bigwedge_{1 \le i \le n, xR^i q} \mathrm{Not}^i_x(\top_q) \gg \neg q\ \overline{M^i_{e_x}}$ — By appropriate applications of rule $\gg\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\top_q) \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\mathrm{Not}_\alpha\top$

Case:

$$\mathcal{I}^- = \frac{x{:}a \in \Gamma, \text{ for all } \theta\ \Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \overline{N_n} \doteq \overline{M^i_{e_x}}}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall(q\ \overline{N_n} \leftarrow G) \gg q\ \overline{M^i_{e_x}}} \not\gg \to^1$$

for all θ $\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} [\theta]\overline{N_n} \doteq q\ \overline{M^i_{e_x}}$ and $x{:}a \in \Gamma$ — By sub-derivation

Subcase: $e_x \not\doteq [\theta]N_i$, N_i rigid by definition for some $1 \le i \le n$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\overline{M^i_{e_x}}/\overline{Z_n}]\overline{Z_n} \doteq \overline{M^i_{e_x}}$ — By rule $\vdash\doteq$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_i]\overline{Z_n} \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\gg\forall$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{1 \le i \le n} \forall Z_1 : A_1. \ldots \forall Z_n : A_n. \neg q\ [e_x/Z_i]\overline{Z_n}) \gg \neg q\ \overline{M^i_{e_x}}$ — By appropriate applications of rule $\gg\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{N_n}) \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\Gamma \vdash \mathrm{Not}^C_x$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} (\bigwedge_{x \in dom(\Gamma)} \Gamma \vdash \mathrm{Not}^C_x(q\ \overline{M_n}) \wedge (\bigwedge_{1 \le i \le n, xR^i q} \Gamma \vdash \mathrm{Not}^i_x(q\ \overline{M_n}))) \wedge \forall(\neg q\ \overline{M_n}) \leftarrow G' \gg \neg q\ \overline{M^i_{e_x}}$ — By appropriate applications of rule $\gg\wedge$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{N_n} \leftarrow G)) \gg \neg q\ \overline{M^i_{e_x}}$ — By rule $\mathrm{Not}_\alpha \to$

Subcase: $e_x \doteq [\theta]N_i, M_j \not\doteq [\theta]N_j$ for $1 \le i, j \le n, i \ne i$:

for all θ $[\theta]N_j \ne M_j$ — By rule $\not\vdash\doteq$
$\Gamma \vdash M_j \notin \|N_j\|$ — By definition of $\|\_\|$
$\Gamma \vdash M_j \in \|\mathrm{Not}(N_j)\|$ — By term exhaustivity (4.21)
$M_j = [\sigma]N$ for some σ and $N \in \Gamma \vdash \mathrm{Not}(N_j)$ — By definition of $\|\_\|$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} M_j \doteq [\sigma]N$ — By rule $\vdash\doteq$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \forall(\neg q\ [e_x/Z_i,N/Z_j]\overline{Z_n}\leftarrow\top)\gg\neg q\ \overline{M^i_{e_x}}$ By rule $\gg\rightarrow$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \bigwedge_{N\in\Gamma\vdash\mathrm{Not}(M_j)}\forall(\neg q\ [e_x/Z_i,N/Z_j]\overline{Z_n}\leftarrow\top)\gg\neg q\ \overline{M^i_{e_x}}$

By appropriate applications of rule $\gg\wedge$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \bigwedge_{1\leq j\leq n}\bigwedge_{N\in\Gamma\vdash\mathrm{Not}(M_j)}\forall(\neg q\ [e_x/Z_i,N/Z_j]\overline{Z_n}\leftarrow\top)\gg\neg q\ \overline{M^i_{e_x}}$

By appropriate applications of rule $\gg\wedge$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}^i_x(\overline{N_n})\gg\neg q\ \overline{M^i_{e_x}}$ By rule Not^i_x

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} (\bigwedge_{1\leq i\leq n,xR^iq}\mathrm{Not}^i_x(\overline{N_n})\gg\neg q\ \overline{M^i_{e_x}}$

By appropriate applications of rule $\gg\wedge$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} (\bigwedge_{x\in dom(\Gamma)}\Gamma\vdash\mathrm{Not}^C_x(q\ \overline{M_n})\wedge(\bigwedge_{1\leq i\leq n,xR^iq}\mathrm{Not}^i_x(q\ \overline{M_n})))\gg\neg q\ \overline{M^i_{e_x}}$

By appropriate applications of rule $\gg\wedge$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} (\bigwedge_{x\in dom(\Gamma)}\Gamma\vdash\mathrm{Not}^C_x(q\ \overline{M_n})\wedge(\bigwedge_{1\leq i\leq n,xR^iq}\mathrm{Not}^i_x(q\ \overline{M_n})))\wedge\forall(\neg q\ \overline{M_n}\leftarrow G')\gg\neg q\ \overline{M^i_{e_x}}$

By rule $\gg\wedge_2$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{N_n}\leftarrow G))\gg\neg q\ \overline{M^i_{e_x}}$ By rule $\mathrm{Not}_\alpha\rightarrow$

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}[\sigma]\overline{N_n}\not\doteq\overline{M^i_{e_x}}\qquad\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}[\sigma]G}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}\forall(q\ \overline{N_n}\leftarrow G)\gg q\ \overline{M^i_{e_x}}}\ \not\gg\rightarrow_2$$

$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} [\sigma]\overline{N_n}\not\doteq\overline{M^i_{e_x}}$ By sub-derivation

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} [\sigma]\overline{N_n}\doteq\overline{M^i_{e_x}}$ By rule $\vdash\not\doteq$

$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} [\sigma]G$ By sub-derivation

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not_G}[\sigma](G)$ By IH 1

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \forall(\neg q\ \overline{N_n}\leftarrow\mathrm{Not_G}(G))\gg\neg q\ \overline{M^i_{e_x}}$ By rule $\gg\rightarrow$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} (\bigwedge_{x\in dom(\Gamma)}\Gamma\vdash\mathrm{Not}^C_x(q\ \overline{M_n})\wedge(\bigwedge_{1\leq i\leq n,xR^iq}\mathrm{Not}^i_x(q\ \overline{M_n})))\wedge\forall(\neg q\ \overline{M_n})\leftarrow G'\gg\neg q\ \overline{M^i_{e_x}}$

By rule $\gg\wedge_2$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(\forall(q\ \overline{N_n}\leftarrow G))\gg\neg q\ \overline{M^i_{e_x}}$ By rule $\mathrm{Not}_\alpha\rightarrow$

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_1\gg Q}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}(D_1\vee D_2)\gg Q}\ \not\gg\vee_1$$

$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_1\gg Q$ By sub-derivation

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1)\gg\neg Q$ By IH 2.2

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1)\wedge\mathrm{Not}_\alpha(D_2)\gg\neg Q$ By rule $\gg\wedge_1$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1\vee D_2)\gg\neg Q$ By rule $\mathrm{Not}_\alpha\vee$

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_2\gg Q}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}(D_1\vee D_2)\gg Q}\ \not\gg\vee_2$$

Symmetrical to the above

Case:

$$\mathcal{I}^- = \frac{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_1\gg Q\qquad\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_2\gg Q}{\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}}(D_1\wedge D_2)\gg Q}\ \not\gg\wedge$$

$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_1\gg Q$ By sub-derivation

$\Gamma;\mathcal{D}\not\vdash_{\mathcal{P}} D_2\gg Q$ By sub-derivation

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1)\gg\neg Q$ By IH 2.2

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_2)\gg\neg Q$ By IH 2.2

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1)\vee\mathrm{Not}_\alpha(D_2)\gg\neg Q$ By rule $\gg\vee$

$\Gamma;\mathcal{D}\vdash_{\mathcal{P}} \mathrm{Not}_\alpha(D_1\wedge D_2)\gg\neg Q$ By rule $\mathrm{Not_D}\wedge$

$\boxed{1}$

Case:

$$\mathcal{S}^- = \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} \mathrm{def}(Q,\mathcal{P}\wedge\mathcal{D}) \gg Q}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} Q} \nvdash \mathrm{atm}$$

Subcase: $D \sqsubseteq \mathrm{def}(Q,\mathcal{P})$

$\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D \gg Q$	By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_D}(D) \gg \neg Q$	By IH 2.1
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(\neg Q,\mathcal{P}) \gg \neg Q$	By repeated applications of rule $\gg\vee$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(Q)$	By rule $\vdash$ At

Subcase: $D \sqsubseteq \mathrm{def}(Q,\mathcal{D})$

$\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D \gg Q$	By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not}_{\alpha}(D) \gg \neg Q$	By IH 2.2
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{def}(\neg Q,\mathcal{D}) \gg \neg Q$	By repeated applications of $\gg\vee$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(Q)$	By rule $\vdash$ At

Case:

$$\mathcal{S}^- = \frac{}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} \bot} \nvdash \bot$$

Then $\mathrm{Not_G}(\bot) = \top = \top$ and

$$\mathcal{S}^+ = \frac{}{\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(\top)} \vdash \top$$

Case:

$$\mathcal{S}^- = \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_1 \qquad \Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_2}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} (G_1 \vee G_2)} \nvdash \vee$$

$\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_1$	By sub-derivation
$\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_2$	By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1)$	By IH 1
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_2)$	By IH 1
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1) \wedge \mathrm{Not_G}(G_2)$	By rule $\vdash\wedge$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1 \vee G_2)$	By rule $\mathrm{Not_G}\wedge$

Case:

$$\mathcal{S}^- = \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_1}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} (G_1 \wedge G_2)} \nvdash \wedge_1$$

$\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_1$	By sub-derivation
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1)$	By IH 1
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1) \vee \mathrm{Not_G}(G_2)$	By rule $\vdash\vee$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(G_1 \wedge G_2)$	By rule $\mathrm{Not_G}\vee$

Case:

$$\mathcal{S}^- = \frac{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} G_2}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} (G_1 \wedge G_2)} \nvdash \wedge_2$$

Symmetrical to the above.

Case:

$$\mathcal{S}^- = \frac{\Gamma;\mathcal{D}\wedge D' \nvdash_{\mathcal{P}} G'}{\Gamma;\mathcal{D} \nvdash_{\mathcal{P}} D' \to G'} \nvdash \to$$

$\Gamma;\mathcal{D}\wedge D' \nvdash_{\mathcal{P}} G'$	By sub-derivation
$\Gamma;\mathcal{D}\wedge D' \vdash_{\mathcal{P}} \mathrm{Not_G}(G')$	By IH 1
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} D' \to \mathrm{Not_G}(G')$	By rule $\vdash\to$
$\Gamma;\mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(D' \to G')$	By rule $\mathrm{Not_G} \to$

Case:

$$\mathcal{S}^- = \frac{(\Gamma, y{:}a); \mathcal{D} \not\vdash_{\mathcal{P}} [y/x]G'}{\Gamma; \mathcal{D} \not\vdash_{\mathcal{P}} \forall x{:}a.\, G'} \not\vdash \forall$$

$(\Gamma, y{:}a); \mathcal{D} \not\vdash_{\mathcal{P}} [y/x]G'$	By sub-derivation
$(\Gamma, y{:}a); \mathcal{D} \vdash_{\mathcal{P}} [y/x]\mathrm{Not_G}(G')$	By IH 1
$(\Gamma, y{:}a); \mathcal{D} \vdash_{\mathcal{P}} \forall x{:}a.\, \mathrm{Not_G}(G')$	By rule $\vdash \forall$
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} \mathrm{Not_G}(\forall x{:}a.\, G')$	By rule $\mathrm{Not_G}\forall$

Case: (dis)equations: immediate.

$\square$

Corollary 6.34 *Clause complementation satisfies the boolean rules of negation.*

6.9 Refinements

In the following sections we will move towards an operational semantics for our language.

6.9.1 More on Termination

We prove that clause complementation preserve termination, so that a negative program can be used as a decision procedure if so is its positive counterpart. First some preliminaries: define $\mathrm{Not}(R)$ as follows:

$$\begin{array}{rcl} \mathrm{Not}(\cdot) & = & \cdot \\ \mathrm{Not}(R, G_1 \prec G_2) & = & \mathrm{Not}(R), \mathrm{Not_G}(G_1) \prec \mathrm{Not_G}(G_2) \end{array}$$

Observe that, by the restriction to clauses of the form $\forall(Q \leftarrow G)$, it is sufficient to consider only well-ordering on goals. It is clear that negation is preserved under relation union, that is:

$$\mathrm{Not}(R_1, R_2) \quad = \quad \mathrm{Not}(R_1), \mathrm{Not}(R_2)$$

We will also need the following technical Lemma, stating that goal complementation commutes with substitution:

Lemma 6.35 *Let* $\Gamma \vdash \theta : \Phi$. $\Gamma \vdash_{\Phi} \mathrm{Not_G}([\theta]G) = \Gamma \vdash_{\Phi} [\theta]\mathrm{Not_G}(G)$.

Proof: By a simple induction on the structure of the derivation of $\Gamma \vdash_{\Phi} \mathrm{Not_G}(G)$. $\square$

Lemma 6.36 *If* $[R]$ *is well-founded, so is* $[\mathrm{Not}(R)]$.

Proof: Suppose $[\mathrm{Not}(R)]$ is not well-founded. Then there is an infinite descending chain $\ldots \prec [\theta]G_i \prec \ldots \prec [\theta]G_2 \prec [\theta]G_1$. By definition of $\mathrm{Not}(R)$ there is G'_i such that $\mathrm{Not_G}(G'_i) = G_i$ and that $G'_i \prec G'_{i-1} \in R$ for some $i \in \omega$. Since by Lemma 6.35 $\mathrm{Not_G}([\theta]G_i) = [\theta]\mathrm{Not_G}(G_i)$, then $[R]$ allows an infinite descending chain $\ldots \prec [\theta]G'_i \prec \ldots \prec [\theta]G'_2 \prec [\theta]G'_1$, impossible. $\square$

The final piece is to show that the relation induced by the complement of a program is the complement of the relation induced by the program itself.

Lemma 6.37

1. *If* $\pi :: \Gamma \vdash D \stackrel{D}{\Longrightarrow} R$ *then* $\mathrm{Not_D}(D) \stackrel{D}{\Longrightarrow} \mathrm{Not}(R)$.
2. *If* $\alpha :: \Gamma \vdash D \stackrel{D}{\Longrightarrow} R$ *then* $\Gamma \vdash \mathrm{Not}_{\alpha}(D) \stackrel{D}{\Longrightarrow} \mathrm{Not}(R)$.
3. *If* $\gamma :: \Gamma \vdash G \stackrel{G}{\Longrightarrow} R$ *then* $\Gamma \vdash \mathrm{Not_G}(G) \stackrel{G}{\Longrightarrow} \mathrm{Not}(R)$.

Proof: By induction on the structure of the given derivations. We prove some selected cases:

Case: $\pi :: \top \overset{D}{\Longrightarrow} \cdot;\ \mathrm{Not_D}(\top) = \bot \overset{D}{\Longrightarrow} \cdot = \mathrm{Not}(\cdot)$.

Case: π ends in $\forall(q\ \overline{M_n} \leftarrow G) \overset{D}{\Longrightarrow} R, [\rho]G \prec [\rho]q\ \overline{M_n}$, for a global substitution ρ:

$\cdot \vdash [\rho]G \overset{G}{\Longrightarrow} R$	By sub-derivation
$\cdot \vdash \mathrm{Not_G}([\rho]G) \overset{G}{\Longrightarrow} \mathrm{Not}(R)$	By IH 2
$\forall(\neg q\ \overline{M_n} \leftarrow \mathrm{Not_G}(G)) \overset{D}{\Longrightarrow} \mathrm{Not}(R), \mathrm{Not}(G) \prec \neg q\ \overline{M_n}$	By rule $\overset{D}{\Longrightarrow}\rightarrow$
$\bigwedge_{\overline{N_n} \in \mathrm{Not}(\overline{M_n})} \forall(\neg q\ \overline{N_n}) \overset{D}{\Longrightarrow} \cdot$	By rule $\overset{D}{\Longrightarrow} \wedge$
$\bigwedge_{\overline{N_n} \in \mathrm{Not}(\overline{M_n})} \forall(\neg q\ \overline{N_n}) \wedge \forall(\neg q\ \overline{M_n} \leftarrow \mathrm{Not_G}(G)) \overset{D}{\Longrightarrow} \mathrm{Not}(R), \mathrm{Not}(G) \prec \neg q\ \overline{M_n}$	By rule $\overset{D}{\Longrightarrow} \wedge$
$\bigwedge_{\overline{N_n} \in \mathrm{Not}(\overline{M_n})} \forall(\neg(q\ \overline{N_n})) \wedge \forall(\neg q\ \overline{M_n} \leftarrow \mathrm{Not_G}(G)) \overset{D}{\Longrightarrow} \mathrm{Not}(R), \mathrm{Not}(G) \prec \mathrm{Not_G}(q\ \overline{M_n})$	By rule $\mathrm{Not_G}$At
$\mathrm{Not_D}(\forall(q\ \overline{M_n} \leftarrow G)) \overset{D}{\Longrightarrow} \mathrm{Not}(R, G \prec \neg q\ \overline{M_n})$	By def. of $\mathrm{Not}(R)$

Case: $\gamma :: Q \overset{G}{\Longrightarrow} \cdot;\ \mathrm{Not_D}(Q) = \neg Q \overset{D}{\Longrightarrow} \cdot = \mathrm{Not}(\cdot)$.

Case: γ ends in $\Gamma \vdash G_1 \wedge G_2 \overset{G}{\Longrightarrow} R_1, R_2, G_1 \prec G_1 \wedge G_2, G_2 \prec G_1 \wedge G_2$:

$\Gamma \vdash G_1 \overset{G}{\Longrightarrow} R_1$ and $\Gamma \vdash G_2 \overset{G}{\Longrightarrow} R_2$	By sub-derivation
$\Gamma \vdash \mathrm{Not_G}(G_1) \overset{G}{\Longrightarrow} \mathrm{Not}(R_1)$ and $\Gamma \vdash \mathrm{Not_G}(G_2) \overset{G}{\Longrightarrow} \mathrm{Not_G}(R_2)$	By IH 2
$\Gamma \vdash \mathrm{Not}(G_1) \vee \mathrm{Not_G}(G_2) \overset{G}{\Longrightarrow} \mathrm{Not}(R_1), \mathrm{Not}(R_2),$ $\mathrm{Not_G}(G_1) \prec \mathrm{Not_G}(G_1) \vee \mathrm{Not_G}(G_2), \mathrm{Not_G}(G_2) \prec \mathrm{Not_G}(G_1) \vee \mathrm{Not_G}(G_2)$	By rule
$\Gamma \vdash \mathrm{Not}(G_1 \wedge G_2) \overset{G}{\Longrightarrow} \mathrm{Not}(R_1), \mathrm{Not}(R_2),$ $\mathrm{Not_G}(G_1) \prec \mathrm{Not_G}(G_1 \wedge G_2), \mathrm{Not_G}(G_2) \prec \mathrm{Not}(G_1 \wedge G_2)$	By rule
$\Gamma \vdash \mathrm{Not}(G_1 \wedge G_2) \overset{G}{\Longrightarrow} \mathrm{Not}(R_1, R_2, G_1 \prec G_1 \wedge G_2, G_2 \prec G_1 \wedge G_2)$	By union and def. of $\mathrm{Not}(R)$

□

Corollary 6.38 *If $\mathcal{P}$ is terminating, so is* $\mathrm{Not_D}(\mathcal{P})$.

Proof: By definition, $\mathcal{P} \downarrow$ if $\mathcal{P} \overset{D}{\Longrightarrow} R$ and $[R]$ is well-founded: by Lemma 6.37 $\mathrm{Not_D}(\mathcal{P}) \overset{D}{\Longrightarrow} \mathrm{Not}(R)$ and by Lemma 6.36 $[\mathrm{Not}(R)]$ is well-founded. Thus $\mathrm{Not_D}(\mathcal{P}) \downarrow$. □

6.9.2 Elimination of $\vee$

We now show how to eliminate the $\vee$ operator preserving provability; this will recover uniformity in proof-search. We generalize the approach in [BMPT90], where the operation was defined as follows: for $m_1 : Q_1 \leftarrow G_1$ and $m_2 : Q_2 \leftarrow G_2$:

$$m_1 \vee m_2 \quad = \quad \theta(Q_1 \leftarrow G_1 \wedge G_2) \quad \text{where } \theta = mgu(Q_1, Q_2)$$

As we remarked earlier, the $\vee$ operator was introduced to preserve the duality under negation between conjunction and disjunction in clauses. Its use, as a clause constructor, is limited to clauses in the same predicate definition and therefore it can be eliminated simulating unification in the definition. Yet, the strict higher-order unification problem is quite complex and potentially even more so complicated by the mixed quantifier structure of HHF, though this is not the case for our language, which does not have existentials. Moreover, we have already in our language variable-variable (dis)equations stemming from left-linearization and normalization of input variables. Finally, we have defined unification of simple terms in Section 4.3 as an intersection operation. This has greatly simplified the presentation, but does not immediately give us a notion of most general unifiers seen as sets of substitutions. We thus choose to compile our source into an intermediate language which makes unification problems explicit as simple equational problems in the style of the unification logic introduced in [Pfe91a]. We can perform unification as constraint simplification as used in *Elf* [Pfe89] and *Twelf* [SP98] in a later stage that we do not describe here.

We adapt the *residuation* technique used in [Pfe92] to compile immediate implication into resolution. We define the judgment $D_1 \vee D_2 \setminus D$ in Figure 6.14 by simultaneous induction on D_1 and D_2, with the intended meaning of '$D_1 \vee D_2$ compiles to D'.

$$\frac{}{\top \vee D \setminus \top}\vee\top D \qquad \frac{}{D \vee \top \setminus \top}\vee D\top$$

$$\frac{}{\bot \vee D \setminus D}\vee\bot D \qquad \frac{}{D \vee \bot \setminus D}\vee D\bot$$

$$\frac{}{\forall(q\ \vec{N}_1 \leftarrow G_1) \vee \forall(q\ \vec{N}_2 \leftarrow G_2) \setminus \forall(q\ \vec{N}_1 \leftarrow (\vec{N}_1 \doteq \vec{N}_2) \wedge G_1 \wedge G_2)}\vee\mathrm{AtAt}$$

$$\frac{q \not\equiv q'}{\forall(q\ \vec{N}_1 \leftarrow G_1) \vee \forall(q'\ \vec{N}_2 \leftarrow G_2) \setminus \top}\vee\mathrm{AtAt}\not\equiv$$

$$\frac{D_1 \vee D_2 \setminus D \qquad D_1' \vee D_2 \setminus D'}{(D_1 \vee D_1') \vee D_2 \setminus D \vee D'}\vee\vee L \qquad \frac{D_1 \vee D_2 \setminus D \qquad D_1' \vee D_2 \setminus D'}{(D_1 \wedge D_1') \vee D_2 \setminus D \wedge D'}\vee\wedge L$$

$$\frac{D_1 \vee D_2 \setminus D \qquad D_1' \vee D_2 \setminus D'}{D_2 \vee (D_1 \vee D_1') \setminus D \vee D'}\vee\vee R \qquad \frac{D_1 \vee D_2 \setminus D \qquad D_1' \vee D_2 \setminus D'}{D_2 \vee (D_1 \wedge D_1') \setminus D \wedge D'}\vee\wedge R$$

Figure 6.14: $\vee$-Elimination: $D_1 \vee D_2 \setminus D$

Example 6.39 *Continuing Example 6.25, recall that:*

$$\begin{aligned}
&\mathrm{Not_D}(cloz) = \\
&\quad \forall F\!:\!exp. \neg closed\ (lam\ F) \wedge (\forall F_1, F_2\!:\!exp. \neg closed\ (app\ F_1\ F_2)). \\
&\mathrm{Not_D}(clolam) = \\
&\quad \neg closed\ z \wedge \forall F_1, F_2\!:\!exp. \neg closed\ (app\ F_1\ F_2) \wedge \\
&\quad \forall E\!:\!exp. \neg closed\ (lam\ E) \leftarrow \forall x\!:\!exp.\ (closed\ x \rightarrow \neg closed\ (E\ x)).
\end{aligned}$$

Removing trivially unsatisfiable clauses, the computation of $\mathrm{Not_D}(cloz) \vee \mathrm{Not_D}(clolam) \setminus D$ *results in:*

$$\begin{aligned}
&\forall E\!:\!exp. \neg closed\ (lam\ E) \leftarrow (\forall x\!:\!exp.\ closed\ x \rightarrow \neg closed\ (E\ x)) \wedge \\
&\forall F_1, F_2\!:\!exp. \neg closed\ (app\ F_1\ F_2).
\end{aligned}$$

Going back to the `linx` *example:*

$$\begin{aligned}
&\mathrm{Not_D}(linxx) \vee \mathrm{Not_D}(linxap1) = \\
&\quad (\neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x)) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y)))) \vee \\
&\quad (\neg linx(\lambda x\,.\,x) \wedge \neg linx(\lambda x\,.\,lam(\lambda y\,.\,(E\ x\ y))) \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^1)) \\
&\quad \wedge \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^0)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x))
\end{aligned}$$

$\mathrm{Not_D}(linxx) \vee \mathrm{Not_D}(linxap1) \setminus D$, *where*

$$\begin{aligned}
&D = \\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^1)) \wedge \\
&\quad \neg linx(\lambda x\,.\,app\ (E_1\ x)\ (E_2\ x^0)) \leftarrow \neg linx(\lambda x\,.\,E_1\ x) \wedge \\
&\quad \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)).
\end{aligned}$$

The following lemma guarantees that compilation preserves run-time immediate entailment:

Lemma 6.40 *Let* $D_1 \vee D_2 \setminus D$*: for every ground substitution* $\Gamma \vdash \theta : \Phi$, $\Gamma; \mathcal{D} \vdash [\theta](D_1 \vee D_2) \gg Q$ *iff* $\Gamma; \mathcal{D} \vdash [\theta]D \gg Q$.

Proof: $(\rightarrow)$. By induction on the structure of $\pi :: D_1 \vee D_2 \setminus D$.

Case: π ends in $\vee\top D, \vee\mathrm{AtAt} \not\doteq$: trivially true.

Case: π ends in $\vee\bot D, \vee\mathrm{At}\bot$: immediate.

Case:

$$\pi = \frac{}{\forall(q\ \vec{N}_1 \leftarrow G_1) \vee \forall(q\ \vec{N}_2 \leftarrow G_2) \setminus \forall(q\ \vec{N}_1 \leftarrow (\vec{N}_1 \doteq \vec{N}_2) \wedge G_1 \wedge G_2)}\ \vee\mathrm{atm}$$

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta](\forall(q\ \vec{N}_1 \leftarrow G_1) \vee \forall(q\ \vec{N}_2 \leftarrow G_2)) \gg q\ \vec{N}$ By hypothesis
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta](\forall(q\ \vec{N}_1 \leftarrow G_1)) \gg q\ \vec{N}$ and $[\theta](\forall(q\ \vec{N}_2 \leftarrow G_2)) \gg q\ \vec{N}$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_1]\vec{N}_1 \doteq \vec{N}$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_1]G_1$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_2]\vec{N}_2 \doteq \vec{N}$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_2]G_2$ By inversion
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_1]\vec{N}_1 \doteq [\theta \cdot \sigma_2]\vec{N}_2$ By replacement
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta \cdot \sigma_1 \cdot \sigma_2]\vec{N}_1 \doteq [\theta \cdot \sigma_1 \cdot \sigma_2]\vec{N}_2 \wedge [\theta \cdot \sigma_1 \cdot \sigma_2]G_1 \wedge [\theta \cdot \sigma_1 \cdot \sigma_2]G_2$
By composition of ground substitution
$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta](\forall(q\ \vec{N}_1 \leftarrow (\vec{N}_1 \doteq \vec{N}_2) \wedge G_1 \wedge G_2)) \gg q\ \vec{N}$ By rule

Case: π ends in $\vee\mathrm{At}\wedge, \vee\mathrm{At}\vee, \vee\wedge, \vee\vee$: by an immediate appeal to the IH.

$(\leftarrow)$: similarly. □

We remark that, using again terminating programs, the above Lemma covers immediate denial as well.

We now extend the effect of the above compilation on a program:

Definition 6.41

$$\begin{array}{rcll}
(\top)^{\vee_D} & = & \top & \\
(\bot)^{\vee_D} & = & \bot & \\
(\forall(Q \leftarrow G))^{\vee_D} & = & \forall(Q \leftarrow (G)^{\vee_G}) & \\
(D_1 \wedge D_2)^{\vee_D} & = & (D_1)^{\vee_D} \wedge (D_2)^{\vee_D} & \\
(D_1 \vee D_2)^{\vee_D} & = & D & \textit{where } (D_1)^{\vee_D} \vee (D_2)^{\vee_D} \setminus D \\
(X)^{\vee_G} & = & X & X \in \{\top, \bot, Q, \textit{(dis)eq}\} \\
(G_1 \wedge G_2)^{\vee_G} & = & (G_1)^{\vee_G} \wedge (G_2)^{\vee_G} & \\
(G_1 \vee G_2)^{\vee_G} & = & (G_1)^{\vee_G} \vee (G_2)^{\vee_G} & \\
(D \rightarrow G)^{\vee_G} & = & (D)^{\vee_D} \rightarrow (G)^{\vee_G} & \\
(\forall x{:}a.\, G)^{\vee_G} & = & \forall x{:}a.\, (G)^{\vee_G} &
\end{array}$$

Theorem 6.42 (Elimination of $\vee$) *For every ground substitution* $\Gamma \vdash \theta : \Phi$

1. $\mathcal{S} :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]G$ *iff* $(\mathcal{S})^{\vee}) :: \Gamma; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} ([\theta]G)^{\vee_G}$*;*
2. $\mathcal{I} :: \Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]D \gg Q$ *iff* $(\mathcal{I})^{\vee} :: -; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} ([\theta]\mathcal{D})^{\vee_D} \gg Q$.

Proof: $(\rightarrow)$. By a straightforward mutual induction on the structure of the given derivations: we show the crucial case.

Case: $(\mathcal{I})^{\vee}$ ends in $\gg\vee$:

$\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]D_1 \gg Q$ and $\Gamma; \mathcal{D} \vdash_{\mathcal{P}} [\theta]D_2 \gg Q$ By sub-derivation
$\Gamma; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} ([\theta]D_1)^{\vee_D} \gg Q$ and $\Gamma; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} ([\theta]D_2)^{\vee_D} \gg Q$ By IH 2
$\Gamma; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} ([\theta]D_1)^{\vee_D} \vee ([\theta]D_2)^{\vee_D} \gg Q$ By rule $\gg\vee$
$\Gamma; ([\theta]\mathcal{D})^{\vee_D} \vdash_{\mathcal{P}} [\theta](D_1 \vee D_2)^{\vee_D} \gg Q$ By Lemma 6.40

$(\leftarrow)$. Similarly. □

The same remark w.r.t. termination applies.

We finish up this Chapter by completing our two running example, where in a final pass we have also:

- Renamed negative predicates with more suggestive names.
- Removed irrelevant occurrences of $\top$.
- Hidden the irrelevant augmentation in the final negative clause.
- Brought clauses to the 'core' languages syntax, as in LF.

Example 6.43 *Concluding Example 6.25 and 6.39 the final definition of* $\neg$`closed` *is:*

$$\begin{array}{rcl} oplam & : & \forall E\!:\!exp.\, open\ (lam\ E) \leftarrow \forall x\!:\!exp.\ open\ (E\ x). \\ opap1 & : & \forall E_1, E_2\!:\!exp.\, open\ (app\ E_1\ E_2) \leftarrow open\ E_1. \\ opap2 & : & \forall E_1, E_2\!:\!exp.\, open\ (app\ E_1\ E_2) \leftarrow open\ E_2. \end{array}$$

where we have hidden the irrelevant assumption closed x in `oplam` *and renamed '*$\neg$`closed`*' into* `open`*.*

Example 6.44 *The final definition of* $\neg$`linear` *and in turn* $\neg$`linx` *is:*

$$\begin{array}{rcl} \neg linapp1 & : & \neg linear(app\ F\ G) \leftarrow \neg linear(F). \\ \neg linapp2 & : & \neg linear(app\ F\ G) \leftarrow \neg linear(G). \\ \neg linlam1 & : & \neg linear(lam\ \lambda x\,.\,E\ x) \leftarrow \neg linx(\lambda x\,.\,E\ x). \\ \neg linlam2 & : & \neg linear(lam\ \lambda x\,.\,E\ x) \\ & & \quad \leftarrow (\forall y\!:\!exp.\,(\neg linx(\lambda x\,.\,y) \wedge linear(y)) \rightarrow \neg linear(E\ y)). \\ \neg linxap1 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^1)). \\ \neg linxap2 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x)^0). \\ \neg linxap3 & : & \neg linx(\lambda x\,.\,app\ (F\ x^1)\ (G\ x^0)) \leftarrow \neg linx(\lambda x\,.\,F\ x^1). \\ \neg linxap4 & : & \neg linx(\lambda x\,.\,app\ (F\ x^0)\ (G\ x^1)) \leftarrow \neg linx(\lambda x\,.\,G\ x^1). \\ \neg linxlm & : & \neg linx(\lambda x\,.\,lam(\lambda y\,.\,E\ x\ y)) \leftarrow \\ & & \quad (\forall y\!:\!exp.\,\neg linx(\lambda x\,.\,y) \rightarrow \neg linx(\lambda x\,.\,E\ x\ y)). \end{array}$$

6.10 Summary

In this Chapter we have given a complement algorithm for a significant fragment of third order Hereditary Harrop Formulae, by adapting the idea of *elimination* of negation introduced in [ST84] for Horn logic. This has the neat effect that negation and its problems are *eliminated*, i.e. we avoid any extension to the (meta) language. This has entailed finding a middle ground between the Closed World Assumption usually associated with negation and the Open World Assumption typical of logical frameworks. Our solution is to restrict the set of programs we deem deniable in a novel way, so as to enforce a *Regular Word Assumption* (RWA): we define a class of programs whose dynamic assumptions extend the current database in a specific regular way. Technically, this regularity under dynamic extension is calibrated so as to ensure that static and dynamic clauses never *overlap*. This property extends to the negative program; in a sense, we maintain a distinction between static and dynamic information, but at a much finer level, i.e. *inside* the definition of a predicate. The resulting fragment is very rich, as it captures the essence of the usage of hypothetical and parametric judgments in a logical framework; namely, that they are intrinsically combined to represent *scoping* constructs in the object language.

Chapter 7

Conclusions and Future Work

The importance of higher-order logical frameworks and logic programming languages that depend on intuitionistic logic, open-world assumptions (changing contexts), and lambda-abstractions is becoming more apparent and their use more widespread. Recent research is attempting to increase their expressivity, while preserving the conciseness and elegance of their representation techniques. The issue of negation has to be appreciated in that context. A good understanding of negation in such settings will significantly enhance the expressive strengths of such specification languages.

In this dissertation we have presented a solution to this long-standing issue in logical frameworks endowed with a logic programming interpretation. The solution offered by our approach has the net (and neat) effect that negation and its problems are *eliminated*, i.e. we avoid any extension to the (meta) language.

Although the transformational approach to negation has been investigated in traditional logic programming, this is a novel approach to addressing negative information in the higher-order intuitionistic setting. Many of the techniques from Horn programs do not carry over directly, so creative solutions had to be found to adapt the idea to the higher-order setting. In particular, we had to:

1. Formulate a strict λ-calculus to obtain closure under term complement of the source language.
2. Find a notion of negation normal forms which is compatible with the operational semantics required by Hereditary Harrop Formulae (HHF).
3. Introduce the *Regular World Assumption* (RWA) as a way to reconcile the intrinsic tension between the *Closed World Assumption*, associated with negation, and the *Open World Assumption* typical of languages with embedded implication.

Elimination of negation is particularly tuned to logical frameworks: although the problems connected with negation are analogous in logical frameworks and logic programming, the solution does not need to be the same. In this sense our approach is not a panacea. For example, it is definitely non appropriate in presence of even a small database of facts. Although the typing discipline goes a long way to limit the combinatorial explosion of negative facts, often a different approach is more fruitful; consider a small database recording the age of some people. It would be painful to negate say $\texttt{age k s}^{34}\texttt{(0)}$ by inferring $\texttt{non-age k 0}\ldots\texttt{non-age k s}^{33}\texttt{(0), non-age k s}^{35}\texttt{(X)}$. In this case the use of some form of *constructive* negation is preferable, possibly in the form of disequations, i.e. $\texttt{non-age k Z} \leftarrow \texttt{Z} \neq \texttt{s}^{34}\texttt{(0)}$. This area, to date, has not been explored at all in the higher-order case.

Before discussing how to extend our approach beyond some of the current limitations, we take on again the main technical restriction[1], i.e. to *complementable* programs as defined in Figure 6.7. As we have argued, the key to a successful and implementable pairing of negation and hypothetical judgments is to keep separate at any time static and dynamic information in a program. We have achieved this by requiring every assumption to be parametric, a property which is preserved by the negative program. The eigenvariable condition enforced by the operational semantics of HHF together with the rigidity restriction does the rest.

[1] We discuss the issue of local variables later, see Subsection 7.1.4.

We have argued that this restriction is a most natural one w.r.t. the intended application; in our experience, it covers the overwhelming majority of actual Twelf and λProlog code. This of course does not mean that there are no useful programs which lie outside of this class; it is indeed entirely possible to use hypothetical judgments in isolation from parametric ones. We have noted in Section 5.4 how sometimes this can be avoided by resorting to a finer notion of context such as the one offered in linear logic. Otherwise, it is always possible to eliminate the offending implications by introducing an explicit management of contexts. For instance, we can revisit Example 5.1 and transform

$$impi \quad : \quad nd(A\ imp\ B) \leftarrow (nd(A) \rightarrow nd(B)).$$

into:

$$\begin{array}{rcl} impi' & : & nd(A\ imp\ B) \leftarrow nd2(A, B). \\ impi1 & : & nd2(A, A). \\ impi2 & : & nd2(C, A\ imp\ B) \leftarrow nd2(C, B). \\ impi3 & : & nd2(C, A\ imp\ B) \leftarrow nd2(A, B). \end{array}$$

Although this goes against the spirit of logical frameworks based on intuitionistic logic programming which owes part of its success to the capability of representing object-logics contexts via the meta-level scoping mechanism of embedded implication, this is not unheard of, and it is actually proposed (for different reasons) in [MM97]. Besides, this transformation will of course be localized only to those predicates we need to complement.

7.1 Lifting Restrictions

We now address some of the restrictions which can in fact be lifted; in doing so, we will consider an example which is not currently treated:

Example 7.1 *Consider the following extension of the* `copy` *clauses to a third and fourth order constructs, respectively* `callcc` *and* `reset`*:*

$$\begin{array}{rcl} callcc & : & ((exp \rightarrow exp) \rightarrow exp) \rightarrow exp. \\ reset & : & (((exp \rightarrow exp) \rightarrow exp) \rightarrow exp) \rightarrow exp \\ cpcallcc & : & cp\ (callcc\ \lambda c{:}exp \rightarrow exp.\, E\ c)\ (callcc\ \lambda d{:}exp \rightarrow exp.\, F\ d) \\ & & \quad \leftarrow (\forall c{:}exp \rightarrow exp. \\ & & \qquad (\forall x{:}exp.\, \forall y{:}exp.\, cp\ (c\ x)\ (c\ y) \leftarrow cp\ x\ y) \\ & & \qquad\quad \rightarrow cp\ (E\ c)\ (F\ c)). \\ cpreset & : & cp\ (reset\ \lambda f{:}(exp \rightarrow exp) \rightarrow exp.\, E\ f)\ (reset\ \lambda g{:}(exp \rightarrow exp) \rightarrow exp.\, F\ g) \\ & & \quad \leftarrow (\forall f{:}(exp \rightarrow exp) \rightarrow exp. \\ & & \qquad (\forall c, d{:}exp \rightarrow exp. \\ & & \qquad\quad cp\ (f\ c)\ (f\ d)) \\ & & \qquad\qquad \leftarrow (\forall x, y{:}exp.\, cp\ x\ y \rightarrow cp\ (c\ x)\ (d\ y)) \\ & & \qquad\qquad\quad \rightarrow cp\ (E\ f)\ (F\ f)) \end{array}$$

7.1.1 Parameters Restrictions

The restriction to parameters of base type seems to be the less complicated to lift, if we still require them to occur only in *head* position. We thus would redefine the class of Shallow Parameter Expressions (*SPE*) as:

$$SPE \quad e_x \quad ::= \quad x : A \mid \lambda x \,.\, e_x \mid (e_x\ M)^k$$

This will allow to complement the third-order *cpcallcc* clause:

$$\begin{array}{l} c{:}exp \to exp \vdash \mathrm{Not}_{\alpha}(\forall x,y{:}exp.\, cp\ (c\ x)\ (c\ y) \leftarrow cp\ x\ y) = \\ \quad \forall(\neg cp\ (c\ x)\ (lam\ E) \wedge \neg cp\ (c\ x)\ (app\ E_1\ E_2) \wedge \\ \quad \neg cp\ (lam\ F)\ (c\ y) \wedge \neg cp\ (app\ F_1\ F_2)\ (c\ y) \wedge \\ \quad \neg cp\ (c\ x)\ (c\ y) \leftarrow \neg cp\ x\ y). \end{array}$$

The condition for parameter to occur at head position instead is orthogonal, since it is a sufficient condition for maintaining non-overlapping between dynamic and static clauses; that is, of course, the main technical idea behind the exclusivity proof. A clause can indeed be *complementable*, i.e. every assumption is parametric, but if the eigenvariable does not occur in head position in the assumption, then the non-overlapping requirement may not be immediately verifiable. If we can detect by static program analysis or by any other means that no overlapping will result, then those clauses too can be promoted.

7.1.2 Extension to Any Order

We currently treat only the third-order case, that is we allow HHF which only make Horn assumptions. In this way, generally speaking, judgments on goals are only trivially recursive with the ones on clauses, since the latter would not make any new assumption. Allowing arbitrary assumptions requires instead this recursion to be unbounded. In general, most of the times we simply need to modify those judgments by passing around their contexts. To be concrete, let us consider for instance the issue of *schema extraction*; in the n-ary case, schemata ought to be *hereditarely closed*. In fact, if a HHF belongs to a schema, the latter ought to take into account any further assumption that the former may yield. The aforementioned modifications of the extraction judgments in Figure 6.5 will do the trick:

$$\frac{\Gamma; \mathcal{D} \vdash G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \Gamma; \mathcal{D} \vdash D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\Gamma; \mathcal{D} \vdash (G \to D) \overset{D}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{D}{\Longrightarrow}\to$$

$$\dots$$

$$\frac{\Gamma; (\mathcal{D} \wedge D) \vdash G \overset{G}{\Longrightarrow} \mathcal{S}_1 \qquad \Gamma; \mathcal{D} \vdash D \overset{D}{\Longrightarrow} \mathcal{S}_2}{\Gamma; \mathcal{D} \vdash D \to G \overset{G}{\Longrightarrow} \mathcal{S}_1 \| \mathcal{S}_2} \overset{G}{\Longrightarrow}\to$$

For example we can extract the following schema from the extended definition of `cp`:

$$\begin{array}{lll} \mathrm{def}(cp) & \overset{D}{\Longrightarrow} & x : exp; cp\ x\ x \| \\ & & c : exp \to exp; (\forall x,y{:}exp.\, cp\ x\ y \to cp\ (c\ x)\ (c\ y)) \| \\ & & f : (exp \to exp) \to exp; (\forall c,d{:}exp \to exp.\, cp\ (f\ c)\ (f\ d) \leftarrow \\ & & \qquad\qquad (\forall x,y{:}exp.\, cp\ x\ y \to cp\ (c\ x)\ (c\ y))) \end{array}$$

In this case only the schema alternative induced by the `cpreset` clause needs to be hereditarely closed with `x:exp;cp x x`; but this is already provided by the (schema extracted from the) `cplam` clause.

Similar changes apply to the other judgments; in particular, augmentation is generalized, so that goals which can make dynamic assumptions will be recursively augmented as well. Formally:

$$\frac{\Gamma; \mathcal{D} \vdash aug_{\mathrm{D}}(D) = D^a \qquad \Gamma; (\mathcal{D} \wedge D) \vdash aug_{\mathrm{G}}(G) = G^a}{\Gamma; \mathcal{D} \vdash aug_{\mathrm{G}}(D \to G) = D^a \to G^a}\, aug_{\mathrm{G}} \to$$

We can therefore complement a fourth-order clause such as `cpreset`, where, for the sake of readability, we have not expanded the calls to $\mathrm{Not}_{\alpha}(D)$:

$$\neg cpreset \ :\ \neg cp\ (reset\ \lambda f{:}(exp \to exp) \to exp.\, E\ f)\ (reset\ \lambda g{:}(exp \to exp) \to exp.\, F\ g)$$

$$
\begin{aligned}
\leftarrow (\forall f\!:\!(exp \rightarrow exp) \rightarrow exp.\\
&(\forall c, d\!:\!exp \rightarrow exp.\\
&\quad (\forall x, y\!:\!exp.\, cp\ x\ y \wedge \mathrm{Not}_{\alpha}(cp\ x\ y) \rightarrow \neg cp\ (c\ x)\ (d\ y))\\
&\qquad \rightarrow \neg cp\ (f\ c)\ (f\ d) \wedge \mathrm{Not}_{\alpha}(cp\ (f\ c)\ (f\ d)\\
&\qquad\quad \rightarrow \neg cp\ (E\ f)\ (F f))
\end{aligned}
$$

7.1.3 Open Queries

One of the more widespread criticism to negation-as-failure (*NF*) is that it cannot return answer substitutions; in fact, if it is not restricted to ground goals, it may return unsound solutions, as detailed in Section 1.4. On the other hand, one of the bright spots of elimination of negation is the *positivization* of negative information, which can now be queried as ordinary (HHF) clauses. In particular, negative open queries are simply positive queries to the complement program and thus will return a witness if one exists. However, open queries are not accounted for in the theory we have developed so far. Indeed, we have used programs just as *decision procedures*, i.e. such that for every ground goal only yes/no answers can be returned. This does not need to be the case; open queries can be allowed in so far as enough 'directionality' is retained in the query. In fact, if we can split the arguments of a query in an *input* and *output* sequence so that termination (in the input elements) can still be determined, the proof of exclusivity will go through, provided that the no-overlapping requirement holds. However, this is almost infallibly the case, because parameters tend to occur in input position; in fact, as we have already remarked, parametric judgments are typically used to traverse scoping constructs and most of the times termination can be proven w.r.t. those terms.

The idea is to give *modes* to predicate definitions: let $q : A_1 \rightarrow \ldots \rightarrow A_n \rightarrow \mathbf{o}$ and $\mathrm{mode}(q) = m_1, \ldots, m_n$ be a *mode* for q, where each m_i is either $+$ (input) or $-$ (output) [RP96]. More complex moding systems have been proposed in the literature on traditional logic programming, but this one may be pragmatically adequate for our purposes. Now, given m_q consisting of a sequence of $m \geq 1$ +'s and $n \geq 0$ -'s, if we can show that $\mathrm{def}(q)$ is terminating w.r.t. m_q and, in case it is a non-Horn definition, if $\mathrm{def}(q)$ is parametric in at least one position i such that $1 \leq i \leq m$, then the query $\forall \overline{x_m}.\, \exists \overline{y_n}.\, q\ \overline{x_m}\ \overline{y_n}$ is allowed. This can be recursively extended as usual to any goal (and clause). To make the discussion more concrete, let us return to the `copy` example. The results in Chapter 6 guarantees the following, for an appropriate context schema $\Gamma; \mathcal{D}$:

- $\forall x\!:\!exp. \forall y\!:\!exp$ either $\Gamma; \mathcal{D} \vdash cp\ x\ y$ or $\Gamma; \mathcal{D} \vdash \neg cp\ x\ y$.
- $\forall x\!:\!exp. \forall y\!:\!exp$ it is not the case $\Gamma; \mathcal{D} \vdash cp\ x\ y$ and $\Gamma; \mathcal{D} \vdash \neg cp\ x\ y$.

But of course we may be interested in *finding* a y such that the above holds. By noting that `copy` can be shown to be terminating only with one term instantiated, we can consider say x as input and y as output. Additionally, we can check that the non-overlapping condition still holds in this case; thus we can establish:

- $\forall x\!:\!exp$ either $\exists y\!:\!exp$ such that $\Gamma; \mathcal{D} \vdash cp\ x\ y$ or $\Gamma; \mathcal{D} \vdash \neg cp\ x\ y$.
- $\forall x\!:\!exp$ it not the case that $\exists y\!:\!exp$ such that $\Gamma; \mathcal{D} \vdash cp\ x\ y$ and $\Gamma; \mathcal{D} \vdash \neg cp\ x\ y$.

7.1.4 Local Variables Revisited

Allowing non-ground queries has sometimes the side-effect of overcoming the strict restriction to programs without local variables. In certain cases, in fact, we can lift the local variable to the head of the clause (and possibly embed it in a constructor); thus, the variable is no more (intrinsically) existentially quantified in the body and will not cause any problem during complementation; moreover, we do not change the arity of the predicate definition, although the latter too is possible. If we can still prove the program terminating, that is the local variable plays no role in the termination argument, then we can use that predicate definition leaving the ex-local variable as an output value only. For example, we can revisit the program for type-checking in the simply typed calculus; if we change application to allow the type of the argument to the function, that is:

$$\begin{array}{rcl} app & : & exp \to exp \to tp \to exp. \\ ofapp & : & of\ (app\ E_1\ E_2\ T_1)\ T_2 \\ & & \quad \leftarrow of\ E_1\ (arrow\ T_1\ T_2) \\ & & \quad \leftarrow of\ E_2\ T_1 \\ oflam & : & of\ (lam\ E)\ (arrow\ T_1\ T_2) \\ & & \quad \leftarrow (\forall x{:}exp.\, of\ x\ T_1 \to of\ (E\ x)\ T_2). \end{array}$$

Then we can apply (a slight modification of) our complement algorithm, yielding:

$$\begin{array}{rcl} \neg app1 & : & \neg of\ (app\ E_1\ E_2\ T_1)\ T_2 \\ & & \quad \leftarrow \neg of\ E_1\ (arrow\ T_1\ T_2). \\ \neg app2 & : & \neg of\ (app\ E_1\ E_2\ T_1)\ T_2 \\ & & \quad \leftarrow of\ E_1\ (arrow\ T_1\ T_2) \\ & & \quad \leftarrow \neg of\ E_2\ T_1. \\ \neg aug_{\mathrm{D}}(oflam) & : & \neg of\ (lam\ E)\ (arrow\ T_1\ T_2) \\ & & \quad \leftarrow (\forall x{:}exp.\, of\ x\ T_1 \to \\ & & \qquad\quad (\forall T{:}tp.\, \neg of\ x\ T \leftarrow T \doteq\!\!\!\!/\ T_1) \to \\ & & \qquad\qquad \neg of\ (E\ x)\ T_2). \end{array}$$

Of course, this lifting of local variables is not sound in general; in the former case, it is doable because T_1 remains in output position, as (possibly higher-order) mode analysis guarantees. On the other hand, for example, it will not work for predicates as graph reachability:

$$\begin{array}{l} path(X,Y) \leftarrow edge(X,Y). \\ path(X,Y) \leftarrow edge(X,Z) \wedge path(Z,Y). \end{array}$$

Here, the local variable Z becomes an input argument to the recursive call for `path`. It is our opinion that the issue of local variables during complementation does not have a universal solution. Approaches which embrace the extensional nature of universal quantification brought in by the negation of existential quantifiers [BMPT90, ABT90] are not satisfactory and robust enough to carry over to logical frameworks with intensional universal quantification. On the other end of the spectrum, the idea to simply keep those universally quantified predicates as constraints, such as in [BLLM94] will pay some run-time overhead. We instead conjecture that (sound) variable lifting may be enough for a sufficiently interesting class of programs w.r.t. our intended application. In fact, it goes along well with the rigidity restriction we have already to enforce to obtain exclusivity of complementation.

7.2 Extensions

The techniques we have described in this dissertation applies to a framework such as L_λ; we briefly comment on how to extend elimination of negation to richer languages.

7.2.1 Beyond Patterns

Term complementation and intersection require higher-order patterns; both λProlog and Twelf allow any (well-typed) terms, although they are dealt with differently. Since we have (dis)equations in our language, we can apply an idea analogous to normalization of input variables 6.4.1 and replace occurrence of non-patterns with fresh variables in those predicates we intend to complement, and constrain those variable to be equal to the replaced term. This is close to what happens in the Twelf implementation and allow the constraints handling mechanism to work its magic. If the framework is not constraint-oriented, it is well-known how to compile away the offending non-pattern occurrence via appropriate substitution predicates implemented through the `copy` clauses [Lia95].

7.2.2 Richer Type Theories

The approach we have chosen is tailored to satisfy the requirements of more complex logical frameworks than L_λ; thus, we now mark some observations on how elimination of negation can be extended to those type theories.

Polymorphism

Different degrees (in the λ-cube) of polymorphism have been advocated as a feature in logical frameworks. Of course, the more expressive the type theory, the more complicated its meta-theory. Pfenning has given an algorithm for patterns unification and generalization in the Calculus of Constructions [Pfe91b]. Even if we stick to the fully applied case, term complementation may be fairly difficult to achieve in general. Languages such as λProlog instead offer the more manageable prenex (ML-like) polymorphism. In this case, a version of term complementation able to deal with polymorphic constructors, such as a polymorphic *cons*, should be feasible. In many ways, the problems are analogous to negation in presence of predicate quantification (see next entry 7.2.3), namely the tension between a static operation such as complementation and the possible instantiations offered by polymorphism.

Dependent Types

Almost all the design decisions we have taken while addressing the issue of negation in a logical framework have been motivated by the ease to extend the latter to a framework such as LF/ *Twelf*. The very idea to allow negation by elimination owes to the requirment to preserve the adequacy of the extraordinary representation power of dependent types, while at the same time avoiding to interfere with the underlying logic programming engine, which makes *Twelf* a unique unified meta-language for the theory and meta-theory of deductive systems. While we fall short in this dissertation to addressing dependent types directly, we believe that the machinery we have developed is robust enough for this extension. One novel problem that we can already foresee is related to the interaction between term complementation and *empty* types. In the simply-typed fragment we can assume every type to be inhabited, but this property is obviously undecidable in the more powerful setting. This turns out to be problematic when complementing variables; at first sight, it not clear what the complement of $\Gamma \vdash \mathrm{Not}(E\ \overline{x_n}) : a\ M$ would be, where the latter may be empty. One possibility is to restrict variable complementation at type $a\ M$, perhaps such that M has no internal structure at all, i.e. it is empty or just a term variable; this would cover all the examples in this dissertation. Another one could be 'approximating' complementation in the simply-typed fragment and then sift out the resulting complement set, in view of dependencies.

7.2.3 Predicate Quantification

In the logic programming community the term 'higher-order' is usually identified with the possibility of quantifying on predicates, in the effort to simulate the first-class functions capability of functional programming languages. This is a sometimes a source of misunderstandings (let me mention *Hilog* [CKW93] which has a higher-order syntax and allows arbitrary terms to appear in places where predicates, functions and atomic formulas occur in predicate calculus, but whose semantics is strictly first-order) and it may be overrated, at least as far as logical frameworks are concerned, as the success of frameworks such as those based on LF testify. Nevertheless, this is a useful, though not essential, feature and has been utilized for example in implementation of proof-carrying code with λProlog [AF99]. Although we have not investigated the issue in depth, it seems that clause complementation can be sometimes applied in this extended sense. For instance, let `some` be a predicate of type $(nat \to \mathbf{o}) \to natlist \to \mathbf{o}$, which selects the first element in a list of numbers for which a given predicate $P : nat \to \mathbf{o}$ holds:

$$\begin{array}{rcl} shd & : & some\ (\lambda x{:}nat.\, P\ x)\ (cons\ Y\ Ys) \\ & & \quad \leftarrow (P\ Y). \\ stl & : & some\ (\lambda x{:}nat.\, P\ x)\ (cons\ Y\ Ys) \\ & & \quad \leftarrow some\ (\lambda x{:}nat.\, P\ x)\ Ys. \end{array}$$

The application of our algorithm would yield:

$$\begin{array}{rcl} \neg snil & : & \neg some\ (\lambda x\!:\!nat.\, P\ x)\ nil \\ \neg stl & : & \neg some\ (\lambda x\!:\!nat.\, P\ x)\ (cons\ Y\ Ys) \\ & & \leftarrow \mathrm{Not_G}(P\ Y) \wedge \neg some\ (\lambda x\!:\!nat.\, P\ x)\ Ys. \end{array}$$

Since we cannot foresee the structure of the goal $(P\ Y)$ at compile-time, we delay the computation of $\mathrm{Not_G}(P\ Y)$ until the instantiation is known. Of course, we cannot allow unrestricted instantiations, but we need to restrict P to conform to the possible context schema. This will prevent goals such as $some\ (\lambda x : nat.\, \neg even(x) \to even(s(s(x)))\ (cons\ 0\ nil)$, which will destroy exclusivity.

Of course, the above example is too simple-minded in at least one respect; we did not consider term complementation on terms with some internal logical structure. The only reason the `some` predicate was complementable is because the predicate occurring inside a term is simply a variable, making term complement trivial. In the general case, clause heads can contain arbitrary complex terms of type **o**. The approach we have developed so far does not seem to capture those phenomena except in the simplest form.

7.3 Implementation Issues

A strict logical framework can be directly implemented with very minor adaptations of well-known techniques used for its linear cousins such as *Lolli* [Hod94, CHP97] and *LLF* [Cer96]. Although we argue that strictness is a useful and ubiquitous concept which deserves to be offered as a *primitive* notion in a logical framework, this is not the only choice.

In fact, Girard would be quick to point out that it is not necessary to take strictness as a primitive at all, since linear logic is flexible enough to express the notion of 'must occur' already. Indeed, strict implication can be embedded into linear logic by simply defining $A \xrightarrow{1} B$ as $A \multimap A \to B$. This is of course true, but notice that while this translation will indeed retain provability, it not faithful to the structure of proofs. Since in a logical framework such as LF we are also concerned with the structure of terms, this embedding is not adequate. For example, the strict term $\lambda x^1.\, c\ x^1\ x^1$ corresponds to both $\lambda x^g \lambda y^u.c\ x^g\ y^u$ and $\lambda x^g \lambda y^u.c\ y^u\ x^g$, where the $(\_)^g$ notation refers to linear abstraction and application. Even if we are just considering proof search (and not proof-terms) there are too many distinct derivations of $A \multimap A \to B$ when compared to $A \xrightarrow{1} B$. So when we take a theorem in strict logic, embed it, and run a logical frameworksuch as *LLF*, we incur into a fair amount of additional non-determinism. On the other hand, the strict λ-calculus captures exactly the right properties in an elegant way and can be developed from first principles. In summary, the linear λ-calculus here does not apply; even though it is clearly possible to compile strict functions to linear ones, this compilation preserves only truth, but not the structure of proofs.

Moreover, as far as negation is concerned, we can safely remain in an intuitionistic setting. The drawback is that we have to decorate source programs whose clause heads (hereditarely) contain partially applied terms with appropriate occurrences of the `vacuous` predicate we have mentioned in Section 2.5. The latter, under negation, is transformed in the `strict` one. Strict unification does not need to be considered, since every term remains in the fully-applied fragment.

For example, if we are encoding a term say $\ulcorner \Lambda x.\, e \urcorner$ with the side condition that $x \notin FV(e)$, we usually represent e with a pattern variable E, which does *not* depend on x. For example, reconsider clause `linxap1`:

$$linxap1 \quad : \quad linx(\lambda x\,.\, app\ (E_1\ x)\ E_2) \leftarrow linx(\lambda x\,.\, E_1\ x).$$

The latter may be rewritten as:

$$linxap1' \quad : \quad linx(\lambda x\,.\, app\ (E_1\ x)\ (E_2\ x)) \leftarrow vacuous(\lambda x\,.\, E_2\ x) \wedge linx(\lambda x\,.\, E_1\ x).$$

where $vacuos(\lambda x\,.\, E_2\ x)$ enforces that x does not occur in $E_2\ x$. The negation transformation will convert those annotations in (pre-compiled) `strict` ones. Pursuing further this example, the complement of `linxap1'` will include:

$$\mathrm{Not_D}(linxap1) \quad : \quad \neg linx(\lambda x\,.\, app\ (E_1\ x)\ (E_2\ x)) \leftarrow strict(\lambda x\,.\, E_2\ x) \vee \neg linx(\lambda x\,.\, E_1\ x).$$

This kind of decoration of programs is a relatively small price to pay compared to the hassle of implementing a strict calculus, only for the purpose of allowing full clause complementation; consider for example the lack so far of a crucial ingredient such as the type reconstruction algorithm (although one for a related calculus is presented in Wright's dissertation [Wri92]). Moreover, for every signature, the definition of the predicate `vacuous` is completely trivial, while the one for `strict` is type-directed and may be automatically inferred, in the style of Miller's `copy` clause [Mil89b]. On the other hand, this approach is less workable when lifting the restriction to *ground* goals; in the presence of open queries, in fact, those annotations would induce an undesirable 'generate-and-test' operational semantics: for example, the `vacuous` predicate would generate vacuous terms to be then checked for, in this case, linearity. It may be possible to internalize the strictness annotations as *boolean constraints* [HP97], in analogy to the linear case; if so, that would work very well with logical frameworks such as *Twelf(X)* [Vir99], which have a declarative notion of constraints.

7.4 Additional Topics

7.4.1 Higher-Order Program Algebra

Several authors have investigated the algebra of logic programs to address modularity and meta-level issues [O'K85, MP88, SW92]. In particular in [MPRT90a] the authors describe a program algebra for Horn clauses without local variables under clause complement, set union and intersection. Since they interpret negation as finite failure, exhaustivity does not hold. In fact, what they call a "constructive version of a boolean algebra" is exactly what Rasiowa [BR57] calls "quasi pseudo Boolean algebras", that is distributive lattices which satisfy the axioms of strong negation restricted to Horn logic.

We can extend the idea of a logic program algebra to a significant fragment of (third-order) HHF. In our case we take equality ($\simeq$) as operational equivalence under appropriate run-time contexts [Har92]; for every program $\mathcal{P}_1, \mathcal{P}_2$ (seen as conjunction of predicate definitions) which satisfy a common context schema $\mathcal{S}$, for every run-time context $\Gamma; \mathcal{D} < \mathcal{S}$ and for every ground G:

$$\mathcal{P}_1 \simeq \mathcal{P}_2 \quad \text{iff} \quad (\Gamma; \mathcal{D} \vdash_{\mathcal{P}_1} G \text{ iff } \Gamma; \mathcal{D} \vdash_{\mathcal{P}_2} G)$$

Then we can organize the set of (finite) programs into a boolean algebra under union '$\wedge$' (lub), intersection '$\vee$' (glb), complement '$\mathrm{Not_D}$', empty program '$\top$' (zero) and universal program '$\perp$' (one). Corollary 6.34 confirms that negation is boolean. Moreover, the rules for $\mathrm{Not_G}, \mathrm{Not_D}$ have been engineered to respect De Morgan's laws.

It could be interesting to study the applicability of those ideas to modularity of higher-order logic programs, in particular to the modular construction of knowledge-based systems; for example a user could collect in a module $\mathcal{P}_1$ all the positive knowledge about a predicate and in another, say $\mathcal{P}_2$, all the negative ones (that is, the cases where the predicate does not hold). This is useful when dealing with defaults and exceptions. The system would be able to compose those modules via boolean manipulations.

7.4.2 Strict and Vacuous Variables

It is our contention that the strict λ-calculus we have introduced to formulate term complementation has an independent interest in the investigation of sub-structural logics. Not only our types system is simpler and (we claim) more elegant than the ones presented in the literature (reviewed in Section 3.3), but the introduction of the notion of *vacuous* variables can be useful in a variety of contexts, beyond strictness analysis. In fact, Pfenning [Pfe00b] has suggested some unexpected usage of those variables in type theory; in particular, if we equip a linear λ-calculus with vacuous variables, this will permit more programs under type assignment. For example consider the function $\lambda x \,.\, \lambda y \,.\, x \otimes (\lambda w \,.\, y)\ x$, which is traditionally considered *not* linear, since x appears twice; nevertheless, the second occurrence is vacuous since it is the argument of a constant function. By accounting for the latter, we can give it the linear type $A \multimap B \multimap (A \otimes B)$. This carries over to the study of *explicit substitutions* [ACCL91] in resource-conscious λ-calculi: for example, in [GPR98] and refined in [CdPR99], the authors propose a system of explicit substitutions for intuitionistic linear logic over unit, lollipop, tensor and bang, where variables can either be linear or intuitionistic. The calculus is not optimal for several reasons, to start with the need for commutative conversions. One technical issue which could be improved is the substitution "extension operator" which accounts for term to be substituted and comes in three flavors: intuitionistic, linear and 'used' linear, to mark a term which has already been linearly substituted. More in general vacuous (or *hidden*) variables go together with the modality of *hidden types*, $M : [A]$, " M is a *past* term of type A" [DPar], which as the adjoint operation of the traditional $S4$ necessity operator, is a promising tool in the study of staged computation [DPar] and computational irrelevance. We plan to develop such a calculus and verify its usefulness in those areas.

Bibliography

[AAB92] J. M. Dunn A. Anderson and N.D. Belnap. *Entailment. The Logic of Relevance and Necessity*, volume 2. Princeton University Press, 1992.

[AAB+95] Johan Andersson, Stefan Andersson, Kent Boortz, Mats Carlsson, Hans Nilsson, Thomas Sjöland, and Johan Widen. SICStus prolog user's manual. Technical Report T93-01, Swedish Institute of Computer Science, 1995.

[AB75] A. Anderson and N.D. Belnap. *Entailment. The Logic of Relevance and Necessity*, volume 1. Princeton University Press, 1975.

[AB90] Krzysztof R. Apt and Marc Bezem. Acyclic programs. In Peter Warren, David H.D.; Szerdei, editor, *Proceedings of the 7th International Conference on Logic Programming (ICLP '90)*, pages 617–633, Jerusalem, June 1990. MIT Press.

[AB94] K. Apt and R. Bol. Logic programming and negation. *Journal of Logic Programming*, 19/20:9–72, May/July 1994.

[ABT90] D. Pedreschi A. Brogi, P. Mancarella and F. Turini. Universal quantification by case analysis. In *Proc. ECAI-90*, pages 111–116, 1990.

[ABW88] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Acz77] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, Amsterdam, 1977.

[AF99] Andrew W. Appel and Amy P. Felty. Lightweight lemmas in lambda prolog. In Danny De Schreye, editor, *Proceedings of the International Conference on Logic Programming (ICLP'99)*, pages 411–425, Las Cruces, New Mexico, December 1999. MIT Press.

[Apt90] Krzysztof R. Apt. Logic programming. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 10, pages 493–574. The MIT Press, New York, N.Y., 1990.

[Bar80] H. P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*. North-Holland, 1980.

[BDLM00] Bugliesi, Delzanno, Liquori, and Martelli. Object calculi in linear logic. *JLC: Journal of Logic and Computation*, 10, 2000.

[Bel74] N.D. Belnap. Functions which really depend on their argument. Manuscript, 1974.

[Bel93] N. D. Belnap. Life in the undistributed middle. In K. Dosen and P. Schroeder-Heister, editors, *Substructural Logics*, pages 31–42. Oxford University Press, 1993.

[BF93] Clement A. Baker-Finch. Relevance and contraction: A logical basis for strictness and sharing analysis. Technical Report ISE RR 34/94, University of Canberra, 1993.

[BLLM94] P. Bruscoli, F. Levi, G. Levi, and M. C. Meo. Compilative constructive negation in constraint logic programs. In Sophie Tison, editor, *Proc. Trees in Algebra and Programming - CAAP'94, 19th International Colloquium*, volume 787, pages 52–76. Springer, 1994.

[BM90] Anthony J. Bonner and L. Thorne McCarty. Adding negation-as-failure to intuitionistic logic programming. In Saumya Hermenegildo Manuel Debray, editor, *Proceedings of the 1990 North American Conference on Logic Programming*, pages 671–693, Austin, TX, October 1990. MIT Press.

[BMPT87] R. Barbuti, Paolo Mancarella, Dino Pedreschi, and Franco Turini. Intensional negation of logic programs. In *Proceedings ofTapsoft87*, volume 259 of *LNCS*, pages 96–110, Austin, TX, October 1987. Springer Verlag Press.

[BMPT90] Roberto Barbuti, Paolo Mancarella, Dino Pedreschi, and Franco Turini. A transformational approach to negation in logic programming. *Journal of Logic Programming*, 8:201–228, 1990.

[BMV89] A. J. Bonner, L. T. McCarty, and K. Vadaparty. Expressing Database Queries with Intuitionistic Logic. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 831–850, Cleveland, Ohio, USA, 1989.

[Bol88] A. W. Bollen. *Conditional Logic Programming*. PhD thesis, Australian National University, 1988.

[Bol90] A. W. Bollen. Relevant logic programming. *Journal of Automated Reasoning*, 7(4):563–586, December 1990.

[Bon91] A.J. Bonner. *Hypothetical Reasoning in Deductive Databases*. PhD thesis, Rutgers University, October 1991.

[Bon94] A.J. Bonner. Hypothetical reasoning with intuitionistic logic. In R. Demolombe and T. Imielinski, editors, *Non-Standard Queries and Answers*, volume 306 of *Studies in Logic and Computation*, pages 187–219. Oxford University Press, 1994.

[BR57] A. Bialynicki-Birula and H. Rasiowa. On the representation of quasi-boolean algebras. *Bulletin de L'académie Polonaise des Sciences*, 5:259–261, 1957.

[CdPR99] Iliano Cervesato, Valeria de Paiva, and Eike Ritter. Explicit substitutions for linear logical frameworks: Preliminary results. In A. Felty, editor, *Proceedings of the Worskshop on Logical Frameworks and Meta-Languages — LFM'99*, Paris, France, 28 September 1999.

[Cer] Iliano Cervesato. Propositional linear logical framework. Manuscript.

[Cer96] Iliano Cervesato. *A Linear Logical Framework*. PhD thesis, Dipartimento di Informatica, Università di Torino, February 1996.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.

[Cha88] D. Chan. Constructive negation based on the completed databases. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth International Conference and Symposium on Logic Programming*, pages 111–125, Seattle, Washington, August 15-19 1988.

[Cha89] D. Chan. An Extension of Constructive Negation and its Application in Coroutining. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 477–496, Cleveland, Ohio, USA, 1989.

[CHP97] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 1997. To appear in a special issue on Proof Search in Type-Theoretic Languages, D. Galmiche, editor.

[Chu41] A. Church. *The Calculi of Lambda Conversion.* Princeton University Press, 1941.

[Chu51] A. Church. The weak theory of implication. In *Kontrolliertes.* Karl Albert, 1951.

[CKW93] Weidong Chen, Michael Kifer, and David S. Warren. Hilog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[CL89] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3–4):371–425, March–April 1989.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[Col84] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS-84)*, pages 85–99, Tokyo, Japan, November 1984. ICOT.

[Com88] H. Comon. *Unification et disunification. Théories et applications.* Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988.

[Com91] H. Comon. Disunification: a survey. In J-L. Lassez and G.Plotkin, editors, *Computational Logic.* MIT Press, Cambridge, MA, 1991.

[Com98] H. Comon. About proofs by consistency. *Lecture Notes in Computer Science*, 1379:136–??, 1998.

[CP96] Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, pages 264–275, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[dB80] N.G. de Bruijn. A survey of the project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, 1980.

[dB91] N.G. de Bruijn. A plea for weaker frameworks. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 40–67. Cambridge University Press, 1991.

[DFH$^+$93] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

[DM97] Raymond Dowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In Glynn Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, June 1997.

[DPar] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, To appear.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, April 1997. Springer-Verlag LNCS. An extended version is available as Technical Report CMU-CS-96-172, Carnegie Mellon University.

[Dun92] P. M. Dung. Declarative semantics of hypothetical logic programming with negation as failure. In E. Lamma and P. Mello, editors, *Proceedings of the Third International Workshop on Extensions of Logic Programming (ELP '92)*, volume 660 of *LNAI*, pages 45–58, Bologna, Italy, February 1992. Springer Verlag.

[Eri92] Lars-Henrik Eriksson. A finitary version of the calculus of partial inductive definitions. In L.-H. Eriksson, L. Hallnäs, and P. Schroeder-Heister, editors, *Proceedings of the Second International Workshop on Extensions of Logic Programming*, pages 89–134, Stockholm, Sweden, January 1992. Springer-Verlag LNAI 596.

[Eri93] Lars-Henrik Eriksson. *Finitary Partial Inductive Definitions and General Logic*. PhD thesis, Department of Computer and System Sciences, Royal Institute of Technology, Stockholm, 1993.

[Fag97] François Fages. Constructive negation by pruning. *Journal of Logic Programming*, 32(2):85–118, August 1997.

[FBM93] G. Levi F. Bruscoli, F. Levi and M.C. Meo. Intensional negation in constraint logic programming. In D. Sacca, editor, *Proc. GULP93*, pages 359–373, 1993.

[FRTW88] Norman Foo, Anand Rao, Andrew Taylor, and Adrian Walker. Deduced relevant types and constructive negation. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 126–139, Seatle, 1988. ALP, IEEE, The MIT Press.

[Gab85] Dov M. Gabbay. N-Prolog: An extension of Prolog with hypothetical implications II. Logical foundations and negation as failure. *Journal of Logic Programming*, 2(4):251–283, December 1985.

[Gab91] Dov Gabbay. Modal provability foundations for negation by failure, in extensions of logic programming. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming: International Workshop, Tübingen FRG, December 1989*, volume 475 of *Lecture Notes in Artificial Intelligence*, pages 179–222. Springer-Verlag, 1991.

[GdQ92] Dov M. Gabbay and Ruy J. G. B. de Queiroz. Extending the Curry-Howard interpretation to linear, relevant and other resource logics. *The Journal of Symbolic Logic*, 57(4):1319–1365, December 1992.

[GH78] J. Guttag and J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10(1):27–52, 1978.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David H.D. Szerdei Peter Warren, editor, *Proceedings of the 7th International Conference on Logic Programming (ICLP '90)*, pages 579–597, Jerusalem, June 1990. MIT Press.

[GMR92] L. Giordano, A. Martelli, and G. Rossi. Extending Horn clause logic with implication goals. *Theoretical Computer Science*, 95(1):43–74, March 1992.

[GO98] Laura Giordano and Nicola Olivetti. Negation as failure and embedded implication. *Journal of Logic Programming*, 36(2):91–147, August 1998.

[GPR98] Neil Ghani, Valeria De Paiva, and Eike Ritter. Linear explicit subsitutions. Technical Report CSR-98-2, University of Birmingham, School of Computer Science, March 1998.

[GR84] Dov M. Gabbay and U. Reyle. N-Prolog: An extension of Prolog with hypothetical implications I. *Journal of Logic Programming*, 1(4):319–355, December 1984.

[GR87] Jean H. Gallier and Stan Raatz. Hornlog: A graph-based interpreter for general Horn clauses. *Journal of Logic Programming*, 4(2):119–155, June 1987.

[GS86] Dov M. Gabbay and Marek J. Sergot. Negation as inconsistency I. *Journal of Logic Programming*, 3(1):1–35, April 1986.

[Hal91] L. Hallnas. Partial inductive definitions. *Theoretical Computer Science*, 87(1):115–147, July 1991.

[Har89] J. Harland. A Kripke-like Model for Negation as Failure. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 626–644, Cleveland, Ohio, USA, 1989.

[Har91a] James Harland. A clausal form for the completion of logic programs. In Koichi Furukawa, editor, *Proceedings of the 8th International Conference on Logic Programming*, pages 711–725. MIT, June 1991.

[Har91b] James Harland. *On Hereditary Harrop Formulae as a Basis for Logic Programming.* PhD thesis, Edinburgh, January 1991.

[Har92] James Harland. On normal forms and equivalence for logic programs. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 146–160, Washington, USA, 1992. The MIT Press.

[Har93] James Harland. Success and failure for hereditary Harrop formulae. *Journal of Logic Programming*, 17(1):1–29, October 1993.

[Hel77] G. Helmann. Completeness of the normal typed fragment of the λ-system u. *Journal of Philosophical Logic*, 1977.

[Hey56] A. Heyting. *Intuitionism, an Introduction.* North-Holland, Amsterdam, 3 edition, 1956.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HM90] Joshua S. Hodas and Dale Miller. Representing objects in a logic programming language with scoping constructs. In Peter Warren, David H.D.; Szerdei, editor, *Proceedings of the 7th International Conference on Logic Programming (ICLP '90)*, pages 511–528, Jerusalem, June 1990. MIT Press.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[HP96] M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. In *Proc. Seventh International Conference on Rewriting Techniques and Applications (RTA'96)*, pages 138–152. Springer LNCS 1103, 1996.

[HP97] J. Harland and D. Pym. Resource-distribution via Boolean constraints. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 222–236, Berlin, July 13–17 1997. Springer.

[HSH90] L. Hallnas and P. Schroeder-Heister. A proof-theoretic approach to logic programming: Clauses as rules. *Journal of Logic and Computation*, 1(2):635-660, October 1990.

[HSH91] L. Hallnas and P. Schroeder-Heister. A proof-theoretic approach to logic programming: Programs as definitions. *Journal of Logic and Computation*, 1(5):261-283, October 1991.

[Isa98] Isabelle. System home page, October 1998. Version 98-1.

[Jen91] Thomas P. Jensen. Strictness Analysis in Logical Form. In John Hughes, editor, *Functional Programming Languages and Computer Architectures*, volume 523 of *Lecture Notes in Computer Science*, pages 352–366, Harvard, Massachusetts, USA, 1991. Springer, Berlin.

[JLLM91] M.J. Maher J-L. Lassez and K. Marriot. Elimination of negation in term algebras. In A. Tarlecki, editor, *Mathematical Foundations of Computer Science*, pages 1–16, Berlin, jul 1991. Springer-Verlag.

[Kah87] Gilles Kahn. Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pages 22–39. Springer-Verlag LNCS 247, 1987.

[KL87] Claude Kirchner and Pierre Lescanne. Solving disequations. In *Proceedings, Symposium on Logic in Computer Science*, pages 347–352, Ithaca, New York, 22–25 June 1987. The Computer Society of the IEEE.

[KNW93] Keehand Kwon, Gopalan Nadathur, and Debra Sue Wilson. Implementing a notion of modules in the logic programming language λProlog. In E. Lamma and P. Mello, editors, *Proceedings of the Third International Workshop on Extensions of Logic Programming*, pages 359–393, Bologna, Italy, February 1993. Springer-Verlag LNAI 660.

[Kre92] Per Kreuger. GCLA II. a definitional approach to control. Technical Report R92:09, SICS, 1992.

[Lia95] Chuck Liang. *Object-Level Substitution, Unification and Generalization in Meta-Logic.* PhD thesis, University of Pennsylvania, August 1995.

[Lif99] Vladimir Lifschitz. Answer set planning. In D. De Schreye, editor, *Proceedings of the International Conference on Logic Programming (ICLP'99)*, pages 23–35. MIT Press, 1999.

[Llo93] J. W. Lloyd. *Foundations of Logic Programming, Second Extended Edition.* Springer-Verlag, 1993.

[LM87] J.-L. Lassez and K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, September 1987.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT press, Cambridge, Massachusetts, 1992.

[LT84] J. W. Lloyd and R. W. Topor. Making PROLOG more expressive. *LOGIC PROGRAM. (USA) ISSN: 0743-1066*, 1(3):225–40, October 1984.

[Lug94] D. Lugiez. Higher-order disunification: Some decidable cases. In J.-P. Jouannaud, editor, *Proceedings of the First International Conference on Constraints in Computational Logics*, pages 121–135, Munich, Germany, September 1994. Springer-Verlag LNCS 845.

[Lug95] D. Lugiez. Positive and negative results for higher-order disunification. *Journal of Symbolic Computation*, 20(4):431–470, October 1995.

[Mah88] M. Maher. Complete Axiomatizations of the Algebras of finite, infinite and rational Trees. In *Proc. third Annual Symposium on Logic in Computer Science*, pages 348–359. Computer Society Press, July 1988.

[Mak87] Johann A. Makowsky. Why Horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34:266–292, 1987.

[MAK91] L. Hallnass Martin Aronson, L-H. Eriksson and P. Kreuger. A survey of GCLA: a definitional approach to logic programming. In Peter Schroeder-Heister, editor, *Proceedings of the First International Workshop on Extensions of Logic Programming*, volume 1050 of *475*, pages 19–34, Berlin, March28–30 1991. Springer.

[Mal71] A.I. Mal'cev. Complete axiomatixation of classes of locally free algebrae of varioud type. In *The metamathematics of algebraic systems: Collected papers, 1936–1967*, pages 262–289. North-Holland, 1971.

[McC88a] L. Thorne McCarty. Clausal intuistionistic logic: I. Fixed point semantics. *Journal of Logic Programming*, 5(1):1–31, March 1988.

[McC88b] L. Thorne McCarty. Clausal intuitionistic logic II. Tableau proof procedure. *Journal of Logic Programming*, 5:93–132, 1988.

[Mil89a] Dale Miller. Lexical scoping as universal quantification. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 268–283, Lisbon, Portugal, June 1989. MIT Press.

[Mil89b] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Proceedings of the International Workshop on Extensions of Logic Programming*, pages 253–281, Tübingen, Germany, 1989. Springer-Verlag LNAI 475.

[Mil89c] Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1-2):79–108, January 1989.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Mil92] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[Mil94] Dale Miller. A multiple-conclusion meta-logic. In S. Abramsky, editor, *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 272–281, Paris, France, July 1994.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In Glynn Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, June 1997.

[MM00] Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction,. *Theoretical Computer Science*, 231(231):91–119, 2000.

[MMP88] Paolo Mancarella, Simone Martini, and Dino Pedreschi. Complete logic Programs with domain-closure Axiom. *Journal of Logic Programming*, 5:263–276, 1988.

[MN89] J. Maluszyński and T. Näslund. Fail Substitutions for Negation as Failure. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 461–476, Cleveland, Ohio, USA, 1989.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Mom92] Alberto Momigliano. Minimal negation and hereditary Harrop formulae. In Anil Nerode and Mikhail Taitslin, editors, *Proceedings of Logical Foundations of Computer Science (Tver '92)*, volume 620 of *LNCS*, pages 326–335, Berlin, Germany, July 1992. Springer.

[MP88] Paolo Mancarella and Dino Pedreschi. An algebra of logic programs. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1006–1023, Seattle, 1988. ALP, IEEE, The MIT Press.

[MP91] Spiro Michaylov and Frank Pfenning. Natural semantics and some of its meta-theory in Elf. In L.-H. Eriksson, L. Hallnäs, and P. Schroeder-Heister, editors, *Proceedings of the Second International Workshop on Extensions of Logic Programming*, pages 299–344, Stockholm, Sweden, January 1991. Springer-Verlag LNAI 596.

[MP93] Spiro Michaylov and Frank Pfenning. Higher-order logic programming as constraint logic programming. In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming*, pages 221–229, Newport, Rhode Island, April 1993. Brown University.

[MPRT90a] Paolo Mancarella, Dino Pedreschi, Marina Rondinelli, and Marco Tagliatti. Algebraic properties of a class of logic programs. In Saumya Debray and Manuel Hermenegildo, editors, *Proceedings of the 1990 North American Conference on Logic Programming*, pages 23–39, Austin, 1990. ALP, MIT Press.

[MPRT90b] Paolo Mancarella, Dino Pedreschi, Marina Rondinelli, and Marco Tagliatti. Algebraic properties of a class of logic programs. In Saumya Debray and Manuel Hermenegildo, editors, *Proceedings of the 1990 North American Conference on Logic Programming*, pages 23–39, Austin, 1990. ALP, MIT Press.

[Myc80] Alan Mycroft. The theory and practice of transforming call-by-need to call-by-value. In *Proceedings of the 4th International Symposium on Programming*, volume 83 of *Lecture Notes in Computer Science*, pages 269–281. Springer Verlag, 1980.

[Nai86] Lee Naish. *Negation and Control in Prolog*, volume 238 of *Lecture Notes in Computer Science*. Springer Verlag, 1986.

[Nel49] D. Nelson. Constructive falsity. *Journal of Symbolic Logic*, pages 16–26, 1949.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In G. Kahn, editor, *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–349, Amsterdam, The Netherlands, July 1991.

[Nip93] Tobias Nipkow. Orthogonal higher-order rewrite systems are confluent. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 306–317, Utrecht, The Netherlands, May 1993.

[NL92] Olivetti N. and Terracini L. N-Prolog and the equivalence of logic program. *Journal of Logic Language and Information*, 5:253-3392, 1992.

[NL95] Gopalan Nadathur and Donald W. Loveland. Uniform proofs and disjunctive logic programming. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 148–155, San Diego, California, June 1995. IEEE Computer Society Press. Available as Technical Report CS-1994-40, Department of Computer Science, Duke University, December 1994.

[NM88] Gopalan Nadathur and Dale Miller. An overview of λProlog. In Kenneth A. Bowen and Robert A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, August 1988. MIT Press.

[O'K85] R. A. O'Keefe. Towards an algebra for constructing logic programs. In *Proceedings of the International Symposium on Logic Programming*, pages 152–161. IEEE Computer Society, Technical Committee on Computer Languages, The Computer Society Press, July 1985.

[PE88] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1988.

[Pea90] David Pearce. Reasoning with negative information, II: Hard negation, strong negation and logic programs. In H. Pearce, D.; Wansing, editor, *Proceedings of the International Workshop on Nonclassical Logics and Information Processing*, volume 619 of *LNAI*, pages 63–79, Berlin, FRG, November 1990. Springer Verlag.

[Pet81] G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.

[Pfe89] Frank Pfenning. Elf: A language for logic definition and verified meta-programming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322, Pacific Grove, California, June 1989. IEEE Computer Society Press.

[Pfe91a] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe91b] Frank Pfenning. Unification and anti-unification in the Calculus of Constructions. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, Amsterdam, The Netherlands, July 1991.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pfe93] Frank Pfenning. Refinement types for logical frameworks. In Herman Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, pages 285–299, Nijmegen, The Netherlands, May 1993.

[Pfe00a] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 2000. In preparation.

[Pfe00b] Frank Pfenning. Reasoning about staged computation, Sept. 2000. Invited talk at the Workshop on Semantics, Applications and Implementation of Program Generation (SAIG), Montreal, Canada.

[PG86] David L. Poole and Randy Goebel. Gracefully adding negation and disjunction to Prolog. In Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, Lecture Notes in Computer Science, pages 635–641, London, 1986. Springer-Verlag.

[Plo71] G. D. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, 1971.

[PM90] Remo Pareschi and Dale Miller. Extending definite clause grammars with scoping constructs. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of Seventh International Conference on Logic Programming*, pages 373–389, Jerusalem, Israel, June 1990. MIT Press.

[Pym99] David J. Pym. On bunched predicate logic. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 183–192, Trento, Italy, July 1999. IEEE Computer Society Press.

[Rei78] R. Reiter. On closed world databases. In Gallaire and Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[RL92] David W. Reed and Donald W. Loveland. A comparison of three Prolog extensions. *Journal of Logic Programming*, 12(1-2):25–50, January 1992.

[Rob65] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[RP96] Ekkehard Rohwedder and Frank Pfenning. Mode and termination checking for higher-order logic programs. In Hanne Riis Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1058.

[Sch00] Carsten Schürmann. *Automating the meta theory of deductive systems*. PhD thesis, Carnegie Mellon University, 2000.

[SD94] Danny De Schreye and Stefaan Decorte. Termination of logic programs: the never-ending story. *The Journal of Logic Programming*, 19 & 20:199–260, May 1994.

[SH93] Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 222–232, Montreal, Canada, June 1993.

[She85] John C. Shepherdson. Negation as failure II. *The Journal of Logic Programming*, 2(3):185–202, Oktober 1985.

[She88] J. C. Shepherdson. Negation in logic progamming. In J. Minker, editor, *Found. of Deductive Databases and Logic Programming*, page 19. Morgan Kaufmann, San Mateo, CA, 1988.

[She89] J. C. Shepherdson. A sound and complete semantics for a version of negation as failure. *Theoretical Computer Science*, 65(3):343–371, July 1989.

[Sny91] Wayne Snyder. *A proof theory for general unification.* Birkhauser, Boston, 1991.

[SP98] Carsten Schürmann and Frank Pfenning. Automated theorem proving in a simple meta-logic for LF. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, pages 286–300, Lindau, Germany, July 1998. Springer-Verlag LNCS 1421.

[ST84] T. Sato and H. Tamaki. Transformational logic program synthesis. In *International Conference on Fifth Generation Computer Systems*, 1984.

[Stä92] R. F. Stärk. *The Proof Theory of Logic Programs with Negation.* PhD thesis, University of Berne, Switzerland, 1992.

[Sti88] Mark E. Stickel. A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[Stu95] Peter J. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118(1):12–33, April 1995.

[SW92] D. T. Sannella and L. A. Wallen. A calculus for the construction of modular Prolog programs. *Journal of Logic Programming*, 12(1-2):147–177, January 1992.

[Thi84] Jean Jacques Thiel. Stop losing sleep over incomplete data type specifications. In Ken Kennedy, editor, *Conference Record of the 11th Annual ACM Symposium on Principles of Programming Languages*, pages 76–82, Salt Lake City, UT, jan 1984. ACM Press.

[TMM89] Kuo Tsung-Min and Prateek Mishra. Strictness Analysis: A New Perspective Based on Type Inference. In *FPCA '89, Functional Programming Languages and Computer Architecture*, London, UK, September 11–13, 1989. ACM Press, New York.

[TS84] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In Sten-Åke Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming*, pages 127–138, Uppsala, 1984.

[Vir99] Roberto Virga. *Higher-Order Rewriting with Dependent Types.* PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 1999.

[Wad90] P. Wadler. Is there a use for linear logic. Technical report, University of Glasgow, 1990.

[Wal87] M. Wallace. Negation by constraints: A sound and efficient implementation of negation in deductive databases. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 253–263, San Francisco, August - September 1987. IEEE, Computer Society Press.

[Wan93] Heinrich Wansing. *The logic of information structures*, volume 681 of *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1993. Revision of the author's doctoral thesis (Fachbereich Philosophie und Sozialwissenschaften I of the Free University of Berlin).

[WBF93] D. A. Wright and C. A. Baker-Finch. Usage analysis with natural reduction types. In *Lecture Notes in Computer Science*, volume 724, pages 254–266, 1993.

[WM91] Larry Wos and William McCune. Automated theorem proving and logic programming: A natural symbiosis. *Journal of Logic Programming*, 11(1):1–53, July 1991.

[Wri91] D. A. Wright. A new technique for strictness analysis. In *TAPSOFT '91*. Springer-Verlag, New York, NY, 1991. Lecture Notes in Computer Science 494.

[Wri92] D. A. Wright. *Reduction types and intensionality in the lambda-calculus*. PhD thesis, University of Tasmania, September 1992.

[Wri96] David A. Wright. Linear, strictness and usage logics. In Michael E. Houle and Peter Eades, editors, *Proceedings of Conference on Computing: The Australian Theory Symposium*, pages 73–80, Townsville, January 29–30 1996. Australian Computer Science Communications.