Expert-System Development in Soar: A Tutorial

Erik Altmann and Gregg R. Yost*
June 1992
CMU-CS-92-151

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

* Digital Equipment Corporation
111 Locke Drive (LM02/K11)
Marlboro, MA 01752

Abstract

This is a tutorial for building an expert system in Soar using the TAQL programming language. It provides a self-contained reference for the end-to-end development of a Soar system that accomplishes a particular task. It presents a natural-language task description, a system design, and a sample implementation, including a documented code listing. It also discusses chunking (Soar's learning mechanism) in the context of the sample implementation.

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The research was also supported in part by Digital Equipment Corporation and the Natural Sciences and Engineering Research Council of Canada. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, DEC, the Government of Canada, or the US Government.

Keywords: Artificial intelligence, expert systems, machine learning, Soar, TAQL.

1. Introduction

This is a tutorial for building a learning expert system in the Soar problem solving architecture (Laird et al., 1990), using the TAQL programming language (Yost and Altmann, 1991; Yost, 1992). It provides a self-contained reference for end-to-end development of a Soar system, beginning with a natural-language task description and ending with a sample implementation of a system to accomplish the task. In between it illustrates a number of design and programming techniques (without claims to complete coverage of such techniques, or independence from personal style). It also discusses chunking (Soar's learning mechanism) in the context of the sample implementation.

Chapter 2 describes the task of the *shipment scheduling assistant*, which is to generate schedules that coordinate trucks, truck drivers, and shipments. Chapter 3 presents a design that outlines both the dynamic behavior of a system and the static relationship of its components. The last section of this chapter presents a detailed view of chunks learned by a sample implementation of this design. Chapter 4 presents the details of this implementation, including a documented code listing, a trace of the running system, and a listing of the chunks learned during the traced run.

For information about obtaining the code and other listings on-line, or to obtain this or other documents concerning Soar and TAQL, send electronic mail to soar-requests@cs.cmu.edu, or physical mail to The Soar Project, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

2. The Shipment Scheduling Assistant: Task Description

This description was derived from a formal description of a larger set of problems in the shipment scheduling domain (Filman, 1988a). The formal description is for the problem used as an example in a recent expert systems paper (Filman, 1988b).

2.1. Description

The Big Giant Trucking Company ships materials among cities in the Midwest. Customers contact the company and request that goods be transported from one city to another on a specified day. Big Giant's dispatchers collect the orders and create suitable delivery schedules.

A schedule consists of some number of *trips*, where each trip has an *itinerary*, a *truck*, and a *driver*. An itinerary is a list of cities and the highways that the driver should take from one city to the next. The itinerary also states what shipments, if any, the driver should pick up and deliver at each city (sometimes the driver will just pass through with no pickups or deliveries).

Producing a schedule is difficult because of the many constraints on the trips and the schedule as a whole. For example, each driver can drive only one trip, and union drivers can only drive trips that take less than eleven hours. The shipment scheduling assistant takes the set of itineraries on the schedule (which we assume were put together by a dispatcher), and tries to find an assignment of trucks and drivers to the itineraries that does not violate any constraints. If no such assignment is possible, the assistant informs the dispatcher, who must then revise the itineraries and try again.

The remainder of this description provides the information the assistant needs to try to find valid truck and driver assignments.

2.1.1. Drivers, trucks, cities, and highways

Tables 1, 2, and 5 define the driver, trucks, and highways used by Big Giant.

Big Giant serves the following cities:

- *In Illinois:* La Harpe, Oregon, Thayer, Utica, Viola, Yale, and Zion.
- *In Indiana:* Attica, Bloomington, Cook, Delphi, English, Fowler, Gary, Hebron, Indianapolis, Jasper, Kokomo, Mitchell, New Harmony, Paoli, Roselawn, Seymour, and Warsaw.

Name	Union	License Class
Brown	yes	3
Gray	no	1
Green	yes	3
White	no	2

Table 1: Drivers

2.1.2. Constraints

The constraints on schedules and trips are:

- Each driver can drive only one trip, and each trip has only one driver.
- Each truck can be used on only one trip, and each trip has only one truck.
- The maximum weight of a truck's load at any point during a trip cannot exceed the truck's rated weight limit. Big trucks can hold 32000 pounds, medium trucks 10000 pounds, and small trucks 5000 pounds.
- The maximum volume of a truck's load at any point during a trip cannot exceed the truck's rated volume limit. Big trucks can hold 1280 cubic feet, medium trucks 640 cubic feet, and small trucks 400 cubic feet.
- The driver and truck assigned to a trip must be in the trip's origin city to begin with.
- The license class of a driver must be at least as great as the license class required by the truck he or she is assigned. Big trucks require class 3 licenses, medium trucks require at least class 2 licenses, and small trucks require at least class 1 licenses.
- A driver can only drive trips whose duration is less than his or her maximum allowable driving time. The duration of a trip is the sum of the driving times for each segment on the itinerary, plus the time needed for loading and unloading shipments during the trip. The driving time for a segment is the length of the road used for that segment divided by the estimated speed for that road (as determined by the weather and road grade, see Table 3). Union drivers can be on a trip for at most 11 hours, while non-union drivers can be on a trip for at most 12.5 hours.
- White cannot drive on any trip that passes through a city in Illinois (he is wanted for a crime there).

Name	Class
Cannonball	big
Piper	small
Queen Bee	medium
Traveler	medium

Table 2: Trucks

Road grade → Weather ↓	Primary	Secondary	Tertiary
Fair	60	55	50
Rain	55	50	35
Snow	45	40	30

Table 3: Estimated travel speed, given road grade and weather

2.2. Test case

This section presents a simple test case that you can use to partially test your scheduling assistant.

The weather throughout Big Giant's area of operations is rainy. Drivers Brown and Gray are in Gary, and drivers Green and White are in Indianapolis. Trucks Piper and Traveler are in Gary, and trucks Cannonball and Queen Bee are in Indianapolis.

The dispatcher's schedule has three trips. The shipments referred to in the trips are listed in Table 4.

- *Trip 1:* Starting in Gary, pick up the typewriter shipment and take highway U30 to Warsaw, followed by U31a to Kokomo, U31b to Indianapolis, and I74 to Attica. Deliver the typewriter shipment in Attica.
- *Trip 2:* Starting in Gary, take U41a to Cook, I80b to Utica, and I80a to Viola. Pick up the carpet shipment in Viola. Then, take I80a back to Utica, and I80b to Cook. In Cook, deliver the carpet shipment and pick up the newsprint shipment.
- *Trip 3:* Starting in Indianapolis, take I70b to Yale, then take U41e to Attica.

One valid solution for this test case is to assign Gray/Piper to trip 1, Brown/Traveler to trip 2, and Green/Cannonball to trip 3. Piper is the only truck in Gary that Gray is licensed to drive, leaving Traveler (the only other truck in Gary) for Brown. White cannot drive trip 3, because it passes through Illinois, so Green must do it.

Material	Origin	Destination	Weight	Volume	Loading Time	Unloading Time
Bicycles	Roselawn	Bloomington	500.0	100.0	0.2	0.25
Books	Oregon	Mitchell	1000.0	50.0	0.2	0.25
Carpet	Viola	Cook	500.0	100.0	0.2	0.25
Computers	Seymour	Thayer	1000.0	150.0	0.2	0.25
Newsprint	Cook	Indianapolis	6000.0	400.0	0.2	0.25
Refrigerators	Kokomo	Warsaw	9000.0	600.0	0.2	0.25
Toys	La Harpe	Oregon	1000.0	100.0	0.2	0.25
Typewriters	Gary	Attica	1000.0	200.0	0.2	0.25

Table 4: Shipments

Name	Connects	Grade	Length
I64a	New Harmony, Jasper	primary	60.0
I64b	English, Jasper	tertiary	30.0
I65	Seymour, Indianapolis	primary	60.0
I70a	Thayer, Yale	primary	150.0
I70b	Indianapolis, Yale	primary	90.0
I74	Indianapolis, Attica	primary	60.0
I80a	Viola, Utica	primary	100.0
I80b	Cook, Utica	primary	90.0
I90	Oregon, Gary	secondary	100.0
I94	Zion, Gary	primary	60.0
S125	La Harpe, Thayer	tertiary	80.0
S25	Delphi, Attica	tertiary	40.0
S26	Delphi, Kokomo	tertiary	30.0
S37a	Bloomington, Indianapolis	secondary	50.0
S37b	Bloomington, Mitchell	tertiary	30.0
S37c	Paoli, Mitchell	tertiary	10.0
S37d	Paoli, English	tertiary	10.0
U231	Cook, Hebron	tertiary	20.0
U24	La Harpe, Fowler	secondary	180.0
U30	Gary, Warsaw	secondary	70.0
U31a	Kokomo, Warsaw	secondary	70.0
U31b	Kokomo, Indianapolis	primary	40.0
U41a	Cook, Gary	secondary	20.0
U41b	Cook, Roselawn	secondary	20.0
U41c	Fowler, Roselawn	secondary	30.0
U41d	Fowler, Attica	secondary	30.0
U41e	Yale, Attica	secondary	90.0
U41f	Yale, New Harmony	secondary	70.0
U50	Mitchell, Seymour	tertiary	30.0
U51	Utica, Oregon	secondary	40.0
U67	Viola, La Harpe	secondary	50.0

Table 5: Highways

3. Designing a Soar Solution

3.1. Introduction

This chapter discusses how to design a Soar system that solves the shipment scheduling task. The guidance toward a particular implementation increases as the chapter progresses, allowing novice Soar users to choose a stepping-off point, or to stick with the chapter to the end to gain exposure to one particular set of design and programming methods. The particular implementation, together with a trace and a chunk listing, is given in Chapter 4.

Section 3.2 presents a one-page analysis of the test case from Chapter 2, including a summary of the various constraints on the solution.

Section 3.3 begins with an analysis of these constraints and a procedure for solving the task. It then outlines the dynamic behavior and static structure of a Soar system. The dynamic and static designs are depicted graphically, and then tied together in a Soar pseudo-trace. The last subsection presents a detailed design, in textual form.

Section 3.4 introduces chunking, backtracing, and the problem of learning from exhaustion. It also discusses methods of manipulating data to prevent overgeneral chunks, including a case-study of how certain conditions come to be included in a particular chunk.

3.2. Notes on the Test Case

The description of the three trips in the test case (Section 2.2, page 5) is reproduced below. Relevant notes, compiled from the various tables, are made under each trip. A summary of constraints, and a specification for the solution, are given afterwards.

- *Trip 1:* Starting in Gary, pick up the typewriter shipment and take highway U30 to Warsaw, followed by U31a to Kokomo, U31b to Indianapolis, and I74 to Attica. Deliver the typewriter shipment in Attica.

  Notes: There are four segments. The *time* needed to drive them, and to load and unload the single shipment, adds up to 3.7 hours, so any driver will do. Similarly, the *size* of the carpet shipment (volume 200, weight 1000) provides no constraint: any truck will do.

  Drivers available are Gray and Brown, and *trucks* available are Piper (small) and Traveler (medium). Possible assignments: Gray/Piper, Brown/Traveler, Brown/Piper. (Gray is licensed only for Piper.)

  Brown/Piper will cause the schedule to fail, because Trip 2 also starts in Gary and Gray/Traveler is illegal.

- *Trip 2:* Starting in Gary, take U41a to Cook, I80b to Utica, and I80a to Viola. Pick up the carpet shipment in Viola. Then, take I80a back to Utica, and I80b to Cook. In Cook, deliver the carpet shipment and pick up the newsprint shipment.

  Notes: There are six segments, counting loading the newsprint as the beginning of a segments that is otherwise null. Drive *time* adds up to 6.5 hours, so any driver will do. The *size* of the shipments provides one constraint: the *weight* of the newsprint shipment (6000) requires a medium truck (Traveler).

  Drivers available are Gray and Brown, and *trucks* available are Piper (small) and Traveler (medium). Possible assignments: Gray/Piper, BrownTraveler, Brown/Piper. (Gray is licensed only for Piper.)

  Any assignment involving Piper will fail on the last segments because Piper's weight limit will not accommodate the newspaper shipment.

- *Trip 3:* Starting in Indianapolis, take I70b to Yale, then take U41e to Attica.

  Notes: There are two segments. The *time* needed to drive them is 3.4 hours, so any driver will do. There are no shipments, so *size* provides no constraint.

  Drivers available are Green and White, and *trucks* available are Queen Bee and Cannonball. Possible assignments: all four pairs.

  Any assignment involving White will fail on the first segment because Yale is in Illinois.

Global constraints:	<Brown, Piper, Trip 1> means Trip 2 cannot be covered.
Local constraints:	<X, Piper, Trip 2> will fail because Piper is too small. <White, X, Trip 3> will fail because White will be arrested in Yale.
Solution:	<Gray, Piper, Trip 1>, <Brown, Traveler, Trip 2>, <Green, X, Trip 3>

3.3. Problem Space Design

3.3.1. Task analysis

A first step in developing a solution is to look at the source of difficulty in the task, which in this case is the set of constraints listed on page 4.

The constraints fall into three categories:

1. *Immediate* constraints — These can be checked immediately, based on the information in the tables given with the task description. For instance, a series of table lookups can tell us that a particular driver is in the city in which a trip originates, or is licensed to drive a particular class of truck.

2. *Local* constraints — These must be met for a particular assignment of a truck and a driver to a trip to be successful (the constraints are *local* to a particular assignment).

   Checking local constraints requires some computation beyond looking up information in the tables. For instance, the load in the truck at any one time is a function of how the shipments are spread out across the trip, and checking that the truck's capacity is not exceeded requires simulating the effects of loading, unloading, and driving.

   Violations of local constraints surface as the inability to complete a particular trip.

3. *Global* constraints — These must hold across assignments for the entire schedule to be successful (they are *global* with respect to particular assignments).

   For instance, consider the possibility of assigning Brown to Piper for Trip 1. This assignment is successful, but leaves no remaining assignments for Trip 2, because Gray is not licensed for Traveler. Thus the schedule as a whole will fail if it includes the first assignment.

   Global constraint violations surface as a situation in which there are trips left over but no assignments for them.

These categories help to outline a procedure for solving the task, which is shown in Figure 1. Step 1 generates assignments that satisfy the immediate constraints. Steps 2 and 3 check the local and global constraints. Step 4 detects when the task is solved, and Step 5 recursively invokes the procedure when it is not.

The procedure in Figure 1 can be mapped onto problem spaces as follows. If we associate each assignment with an operator, each such operator presents an opportunity to check the local constraints associated with that assignment. The effect of such an operator would be to simulate a trip with a particular truck and driver, and generate a state in which either the schedule is updated with that assignment, or in which a failure of the assignment (due to local constraints being violated) is indicated. We can call this the *simulate-trip* operator. The operator itself requires a sequence of steps, namely simulating the loading and unloading of shipments and the driving of segments of the trip. Simulate-trip can be applied by a separate problem space that executes these steps.

The immediate constraints can be incorporated in the proposal of simulate-trip operators. This

1. Generate a set of possible assignments for unassigned trips that satisfy all immediate constraints, and choose one assignment.
2. Check the local constraints on that assignment, by simulating the trip. If a constraint fails, backtrack to some previous instance of Step 1 where alternatives remain, and choose one. If there are no previous instances where alternatives remain, halt with failure.
3. If no local constraints are violated, check whether any global constraints have been violated by checking if there are trips left over but no assignments for them. If a global constraint has been violated, backtrack to some previous instance of Step 1 or halt with failure (as in Step 2).
4. If no local or global constraints are violated, check whether we have a successful schedule, by checking whether all the trips have assignments. If so, halt with success.
5. Mark the trip as having been assigned, update the pool of available trucks and drivers, and recurse to Step 1.

Figure 1: Procedure that solves the shipment scheduling task

will reduce the number of operators generated (and therefore the search involved in solving the task). Further reduction in search comes if we base successive operator proposals on the updated set of trips and the updated pool of available trucks and drivers (all of which shrink with each successful assignment). These updates can be carried out by simulate-trip itself, upon successful completion of a simulation. This updating also allows testing for success by detecting when no more unassigned trips remain, and testing for violations of global constraints by detecting when trips are left over but no assignments are left.

The states in this scheme will have to represent several kinds of information. First is that from the tables presented in Chapter 2. This information implicitly represents many of the immediate constraints, and therefore must be available when proposing simulate-trip operators. Second is the dynamic information that changes from trip to trip. This includes the sets of unassigned trips, available drivers, and available trucks; the schedule, as it grows with each successful assignment; and indications of local and global constraint violations. Third is the dynamic information that changes only from segment to segment within a trip. This includes resources such as available drive time and truck capacity.

Finally, we need a mechanism for backtracking when constraints fail. One simple mechanism is depth-first search, in which the selection at the most recent instance of Step 1 is changed, unless there are no more alternatives, at which point the procedure backtracks to the next most recent instance of Step 1.

3.3.2. Dynamic behavior

Figure 2 shows an example trace of the procedure above as mapped onto problem spaces, with an emphasis on how depth-first search works in Soar. Spaces are depicted as triangles, states as circles, operators as lines emanating from states, impasses as down arrows, and subgoal results are up arrows. The trace is annotated with examples of how each step of the procedure is realized as a problem-space operation. (Figure 3 shows a corresponding pseudo-trace that resembles actual Soar output.) The problem spaces shown are the *task* space, which contains the simulate-trip operator, and the *selection* space, which implements depth-first search (usually referred to as *lookahead* search in the context of Soar).

In the top instance of the task space (T1), a set of simulate-trip operators is proposed (per Step 1). These tie, resulting in an impasse. The selection space (S1) is selected automatically for the subgoal (by Soar's default productions). In the selection space, the default productions create an *evaluate-object* operator for each tied alternative. The evaluate-object operators are given indifferent preferences, so one is selected (at random or in a prescribed pattern; see the Soar *user-select* command). Evaluate-object itself impasses, and in the resulting subgoal the default productions cause the task space to be selected (T2). This subgoal is referred to as the *evaluation subgoal*. The operator being evaluated is then selected automatically, under the assumption that it might generate a state that either succeeds or fails (or might otherwise yield information that allows the operator to be evaluated). This automatically-selected operator is referred to as the *lookahead operator*.

In the left-most evaluation subgoal (T2), the lookahead operator causes a local constraint violation. (The space that applies simulate-trip is omitted, but see Figure 3.) The violation is detected (per Step 2), and results in a failure evaluation for evaluate-object. T2 exits, and a second evaluate-object operator is selected.

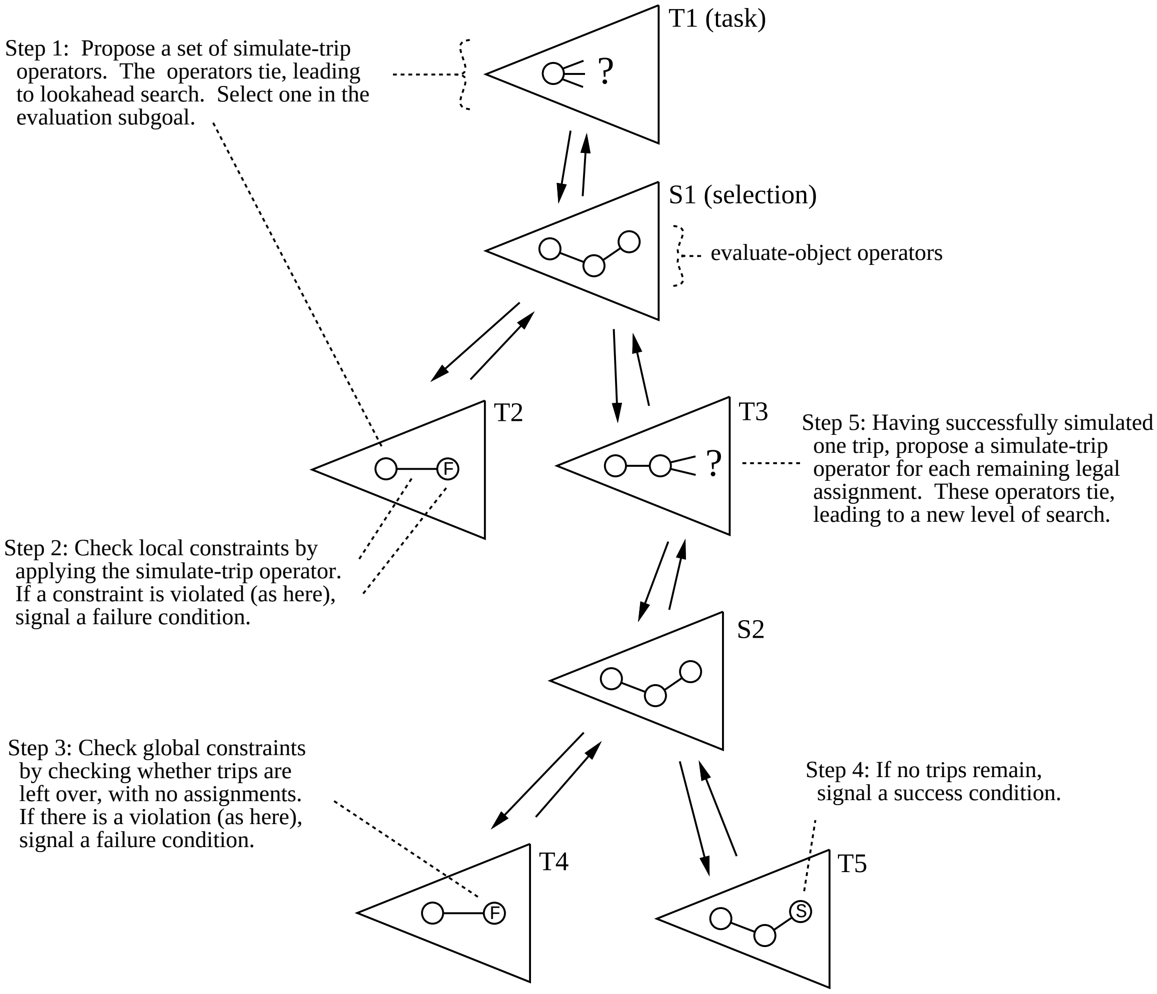

Figure 2: Trace of lookahead search

The lookahead operator in T3 yields neither constraint violations nor success. A new set of simulate-trip operators is proposed (based on the updated sets of unassigned trips and available trucks and drivers). The resulting tie leads to a new instance of the selection space (S2). Below S2, in T4, the simulation leads to a global constraint violation (per Step 3). As with a local constraint violation, this generates a failure evaluation, causing T4 to exit and another alternative to be evaluated.

In T5, the lookahead operator violates no local constraints, and leads to a state in which only one new simulate-trip operator is proposed. That operator is selected, and results in a state in which no trips are left. The space exits with a success evaluation (per Step 4). The default productions propagate success up to T1 (and would propagate it further if there were more levels). This breaks the tie in T1, with the lookahead operator from T3 being selected. If chunking was turned on, the success of the lookahead operator in T5 will have been learned as a chunk that selects that operator after the first selection in T1, leading directly to a successful state in T1.

For further details on lookahead search, consult the Soar manual. The depth-first behavior described here is only one of many weak methods that can arise from the *universal weak method* (Laird, 1984), depending on how much and what kinds of evaluation knowledge are available. For example, hill-climbing arises if any state can be evaluated, and not just those that represent failures or successes.

3.3.3. Static structure

The previous section outlined a procedure for solving the shipment scheduling task, and described at a high level how that procedure could be cast in terms of states and operators. This section describes in more detail a set of Soar problem spaces for performing the task. The first part of the description is a diagram (Figure 3) that shows the static problem space structure, and the second part (beginning on page 16) is a textual specification. Both the diagram and the specification correspond directly to the code presented in Chapter 4, so ambiguities and other confusions can be resolved by looking there.

Figure 3 shows the operators in each space, and the information that flows between spaces. The name of each space appears at its top right-hand corner, and the operators appear in the interior of the triangle. The lines connecting spaces denote impasses.

The remainder of this section traces through the figure, showing how the spaces work together. Note that the diagram represents the *static* structure of the system. This explains how the task space can be connected to two superspaces: it responds to impasses in both, at different times. All the impasses represented in the diagram are operator no-changes; the diagram omits the operator tie impasse that occurs in the task space (on simulate-trip operators), which leads to the selection space. This interaction is described in Figure 2.

The *do-task* operator in the top space is applied in the task space. To carry out the task, operators in the task space need access to the information from the tables, and initial values for the trip, truck, and driver sets. When the task space exits, it returns the complete schedule to the top space as the solution to the task.

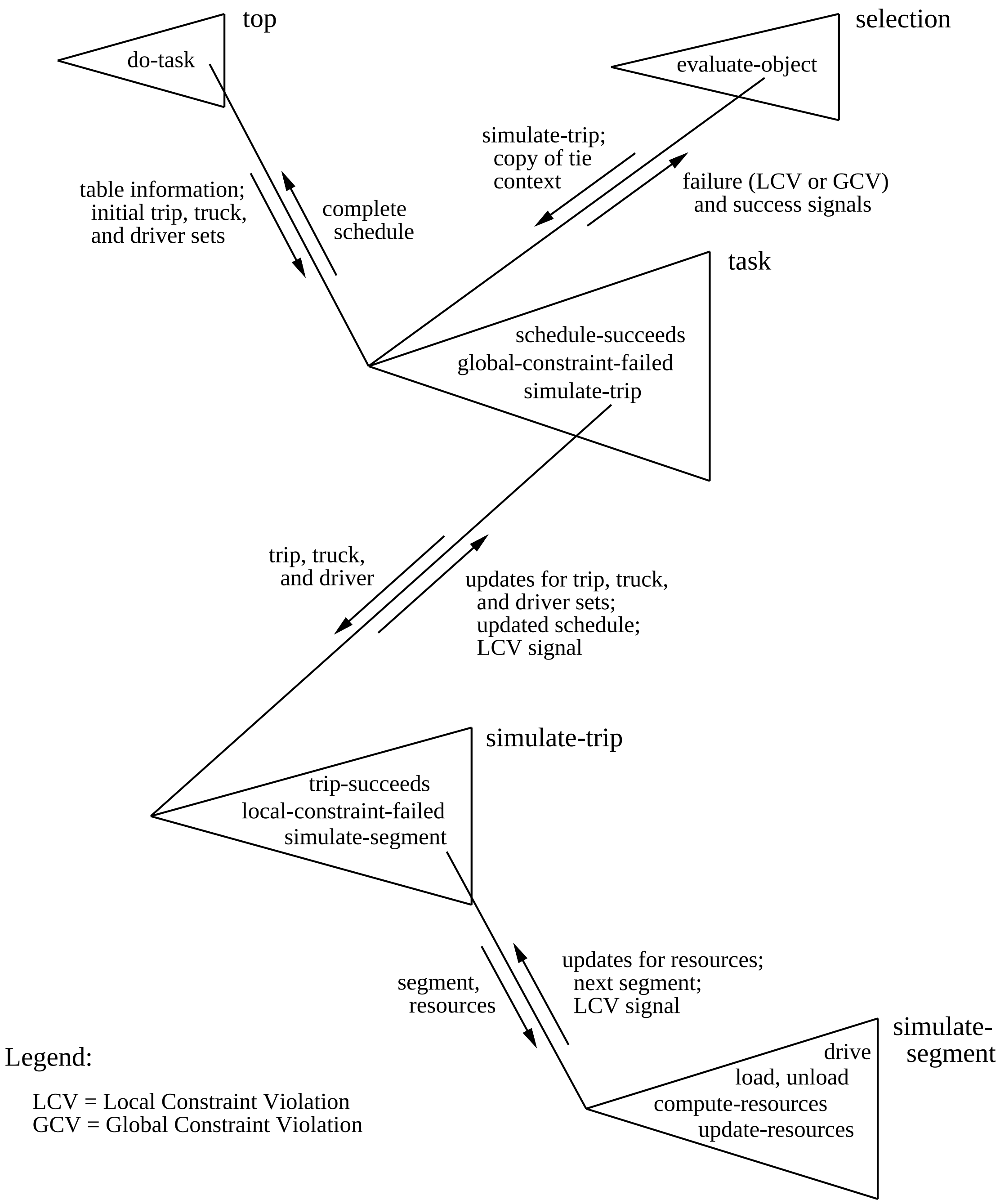

Figure 3: Static problem-space structure

The simulate-trip operator in the task space is applied in the simulate-trip space. Simulate-trip takes the particular assignment and simulates the trip. As part of its result, the space updates the trip, truck, and driver sets by removing those involved in the trip it just simulated. It also returns an indication of whether any local constraints were violated during the simulation. If not, it returns the schedule as updated with the addition of the trip just simulated.

If a simulation generates a local constraint violation, the task space passes the violation signal directly up to the selection space above, if the task space occurs in an evaluation subgoal. The default productions in the selection space act on the failure by terminating the evaluate-object operator and selecting a new one. However, if the simulation is successful, a new simulate-trip operator is selected in the task space.

If after a simulation the task space finds that all trips have been assigned to, the *schedule-succeeds* operator is proposed. This returns a success signal to the selection space if this is an evaluation subgoal, or the complete schedule to the top space. If, on the other hand, there comes a point where there are still trips to be assigned to but no simulate-trip operators proposed, then the *global-constraint-failed* operator is selected. This returns a failure signal to the selection space.

In the simulate-trip space, the *simulate-segment* operator is selected once for each segment of the trip. If after any segment a resource (such as truck capacity) is found to have run out, the *local-constraint-failed* operator is selected. This operator returns a failure signal to the task space, causing the simulate-trip operator to terminate (thereby terminating the simulate-trip subgoal). The task space then terminates as well, as described above. If, on the other hand, all the segments are simulated with no violations, the *trip-succeeds* operator is selected. This operator updates the trip, truck, and driver sets in the task space, and returns the updated schedule.

The simulate segment operator, when selected, impasses, because simulating a segment involves several operations. These occur in the simulate-segment space. The operations for each segment consist of loading any shipments at the beginning of the segment, followed by the drive to the next location, followed by unloading shipments to be delivered at that location. Loading and unloading affect the volume and weight capacities remaining for that trip, and driving affects the time the driver has been on the road. When the segment is complete, the *compute-resources* operator computes the remaining resources and checks if any have run out, and if so signals a local constraint violation on the superstate. If not, the update-resources operator updates the resources on the superstate, in preparation for the next segment.

The Soar pseudo-trace in Figure 4 brings together both static and dynamic aspects of the system. The labels for task and selection spaces from Figure 2 appear along the right margin. The space and operator names are carried over from Figure 3. The impasse types are also given.

```
G: g1
P: p1 (top-space)
S: s1
O: o1 (do-task)
==>G: g2 (operator no-change)
   P: p2 (task)                                                          T1
   S: s2
   ==>G: g3 (operator tie)
      P: p3 (selection)                                                  S1
      S: s3
      O: o3 (evaluate-object (simulate-trip ((gray) (piper) (trip2))))
      ==>G: g4 (operator no-change)
         P: p2 (task)                                                    T2
         S: d1 (duplicates s3)
         O: c1 (simulate-trip (gray) (piper) (trip2))
         ==>G: g5 (operator no-change)
            P: p4 (simulate-trip)
            S: s4
            O: o4 (simulate-segment segment1)
            ...                                     ...intermediate segments
            O: o20 (simulate-segment segment6)
            ==>G: g21 (operator no-change)
               P: p21 (simulate-segment)
               S: s21
               O: o21 (load (newsprint 6000.0))
               O: o22 (drive)
               O: o23 (compute-resources)
               O: o24 (update-resources)
            O: o25 (local-constraint-failure weight-limit-exceeded)
         Evaluation of operator c1 (simulate-trip) is failure
      O: o26 (evaluate-object (simulate-trip ((green) (cannonball) (trip3))))
      ==>G: g22 (operator no-change)
         P: p2  (task)                                                   T3
         S: d2  (duplicates s3)
         O: c2  (simulate-trip (green) (cannonball) (trip3))
            ...                                 ...simulation is successful
            O: o27 (trip-succeeds)
         ==>G: g23 (operator tie)
            P: p23 (selection)                                           S2
            S: s22
            O: o28 (evaluate-object (simulate-trip ((brown) (piper) (trip1))))
            ==>G: g24 (operator no-change)
               P: p2  (task)                                             T4
               S: d3  (duplicates d2)
               O: c3  (simulate-trip (brown) (piper) (trip1))
               ...                                     ...intermediate trips
               O: o40 (global-constraint-violation trips-left-over)
               Evaluation of operator c3 (simulate-trip) is failure
            O: o41 (evaluate-object (simulate-trip ((gray) (piper) (trip1))))
            ==>G: g25 (operator no-change)
               P: p2  (task)                                             T5
               S: d4  (duplicates d2)
               O: c4  (simulate-trip (gray) (piper) (trip1))
               ...
               O: o50 (simulate-trip (brown) (traveler) (trip2))
               ...
               O: o70 (schedule-succeeds)
               Evaluation of operator c4 (simulate-trip) is success
               ...                           ...success propagates up to T1
```

Figure 4: Pseudo-trace corresponding to Figures 2 and 3

3.3.4. A detailed design

This section specifies the operators in more detail, giving their arguments (in parentheses) and their semantics in terms of conditions, effects, and termination conditions. Where appropriate, information about state contents and operator search control is also given.

- **top-space**[1]

  State (also called the *top state*): Contains the invariant information from the tables, which is used to initialize dynamic values like the trip, truck, and driver sets.

  Operators:

  1. **do-task**

     Conditions: Proposed without conditions.

     Also called the *task operator*. Applied in the task space.

     Effects:[2] Adds the completed schedule to the state.

     Termination: Automatic, when a successful or failed state is reached in the subgoal that applies it.[3]

[1] The space itself is provided by TAQL by default.

[2] The effects of an operator applied in a subspace are carried out by the subspace.

[3] This termination is provided by TAQL's runtime support.

- **task**

  State: Contains trip, truck, and driver sets, either initialized from the top state (when below the top space), or duplicated from another instance of the task space (when below the selection space).

  Operators:

  1. **simulate-trip** (trip, truck, driver)

     Conditions: Reads the current state's trip, truck, and driver sets. Checks that the assignment satisfies the immediate constraints that the driver and the truck be in the city where the trip begins, and that the driver is licensed for the truck.

     Applied in the simulate-trip space, which checks local constraints by simulating the trip with that truck and driver.

     Effects: Updates the trip, truck, and driver sets by removing its arguments from them. Updates the current schedule if no local constraints are violated.

     Termination: Terminates (1) if the subspace updates the current schedule with the simulated trip, or (2) if the subspace signals a local constraint violation.

  2. **schedule-succeeds** (complete-schedule)

     Conditions: Proposed when the trip set is empty.

     Effects: Signals success in the superspace, either via evaluation knowledge (during lookahead search) or the goal test (when the superspace is the top space). If the superspace is the top space, returns complete-schedule.

     Termination: By higher decision.[4]

  3. **global-constraint-failed**

     Conditions: Proposed without conditions, but made worst. Consequently, selected only when there are trips left over (inhibiting schedule-succeeds) but no simulate-trip operators are proposed. This condition implies a global constraint violation.

     Effects: Signals failure in the superspace, either through evaluation knowledge (during lookahead search) or the goal test (when the superspace is the top space).

     Termination: By higher decision.

[4] When information is returned that resolves an impasse in the supercontext, Soar terminates all lower selections.

- **simulate-trip**

  State: Contains resources (initially the time for which the driver can be on the road, and the weight limit and volume of the truck), and the current segment of the trip being simulated. Initialized from the superoperator, and updated by simulate-segment.

  Operators:

  1. **simulate-segment** (segment, resources)

     Conditions: Reads the segment and resources from the state.

     Applied in the simulate-segment subspace, which loads, unloads, and drives as necessary for that segment. The subspace signals if a resource went negative, or any other local constraint was violated (such as White being arrested because the segment ends in Illinois).

     Effects: Changes the current segment to be the next segment, and updates the resources on the current state. Signals failure on the current state if there was a constraint violation (see local-constraint-failed).

     Termination: Terminates (1) when the current segment becomes the next segment, if there is no constraint violation; or (2) when a constraint violation occurs.

  2. **trip-succeeds** (assignment)

     Conditions: Proposed without conditions, but made worst. Consequently, selected only when there are no more simulate-segment operators and no constraint violations.

     Effects: Adds assignment to the current schedule on the superstate, allowing the superoperator to terminate. Also updates the trip, truck, and driver sets on the superstate.

     Termination: By higher decision.

  3. **local-constraint-failed** (failed-assignment)

     Conditions: Proposed if simulate-segment signals failure, and selected immediately when proposed.

     Effects: Signals failure in the supercontext, allowing the assign operator to terminate. Returns failed-assignment to the superstate, so we can figure out what happened.

     Termination: By higher decision.

- **simulate-segment**

  State: Contains resources (weight, volume, time), initialized from the superoperator.

  Search control: A segment begins with a load (if any), and ends with an unload (if any). Therefore, for correctness with respect to the truck's weight limit and volume, load must be selected before unload.

  Operators:

  1. **load** (volume, load-time, weight)

     Conditions: Proposed if there is a shipment to load in the beginning location.

     Effects: Updates weight, volume, and time, on the current state.

     Termination: By direct application.[5]

  2. **unload** (volume, unload-time, weight)

     Conditions: Proposed if there is a shipment to unload at the end location.

     Effects: Updates weight, volume, time.

     Termination: By direct application.

  3. **drive** (time)

     Conditions: Proposed without conditions (segments always involve driving).

     Effects: Updates time.

     Termination: By direct application.

  4. **compute-resources**

     Conditions: Proposed without conditions, but made worst. Consequently, selected when all other operations have been carried out.

     Effects: Augments the state with a record of the remaining resources, Indicates in the record whether resources went negative, or whether any other local constraints were violated (such as White being arrested in Illinois).

     Termination: By direct application.

  5. **update-resources** (resource-record)

     Conditions: Proposed when the resource record has been computed.

     Effects: Returns resource-record to the superstate. Changes the current segment to be the next segment on the superstate, allowing the superoperator to terminate.

     Termination: By higher decision.

[5] When an operator is applied entirely by an apply-operator TC, edits in the TC terminate the operator.

3.4. Notes on Chunking

Section 3.4.1 discusses some general aspects of chunking in Soar. Section 3.4.2 examines some of the code from Chapter 4 in light of how it affects chunking. Novice Soar users are advised to skim this section for future reference, rather than read it for understanding immediately. Section 3.4.3 briefly discusses a sample expensive chunk.

3.4.1. Correctness, generality, and backtracing

There are two aspects to good chunks: correctness and generality. These aspects trade off: incorrect chunks are often that way because they are overgeneral. There is usually an identifiably optimal degree of generality, in which the chunk conditions are the weakest that guarantee that the knowledge contained in the action is correct. This applies to both forms of chunks learned in the sample implementation (operator application chunks, which modify the state, and search control chunks, which create preferences for operators).

Chunk conditions are created by a dependency analysis on subgoal results, called *backtracing*. Backtracing is a form of operator regression (Mitchell et al., 1986). In Soar terms, backtracing involves tracing through working memory elements that were added and deleted from working memory during processing in the subgoal, beginning with the subgoal result and working back to the working memory elements that it depends on.

Figure 5 shows a simple example of subgoal-result generation and backtracing. The initial state is created from supercontext augmentations *aug1* and *aug2*. Augmentations *aug3* and *aug4* are copied down directly from the supercontext. Operator 1 tests only the state, and generates augmentation *aug5*. Operator 2 generates *aug6* from aug3 and aug4. Operator 3 generates *aug7* from aug5 and aug3. Finally, Operator 4 generates *aug8*, the subgoal result, from aug7. Backtracing follows this dependency path backwards, starting with aug8. Aug6, and before that aug4, are irrelevant to the subgoal result, and therefore are not included in the chunk conditions. Note that every operator must test the state, either to edit the state directly or to reach the objects that will be edited, and therefore backtracing will always reach the augmentations used to generate the initial state (aug1 and aug2, in the figure).

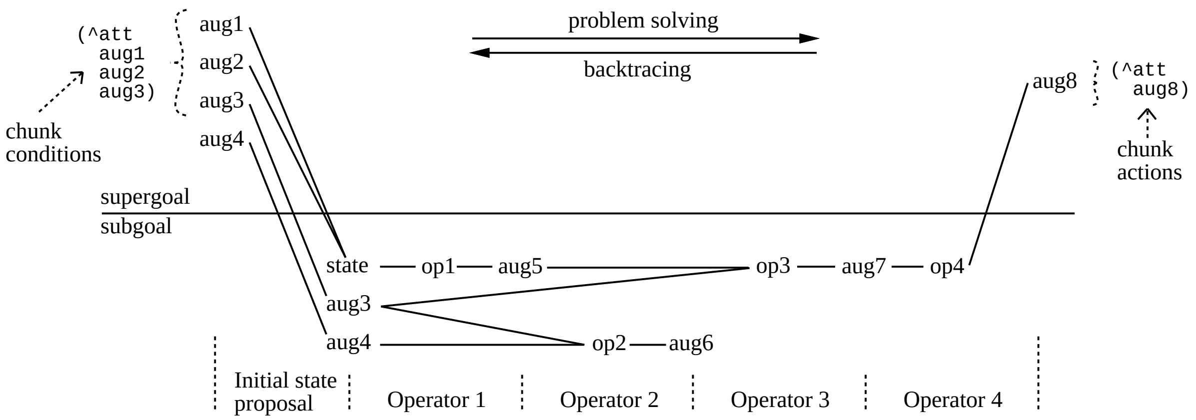

Figure 5: Simple backtracing example

Productions that create desirability preferences (other than the require and prohibit preferences, which are special cases) are not backtraced through, even if they bring about a particular operator selection. The underlying premise is that search control affects only the efficiency and not the correctness of problem-solving (Rosenbloom et al., 1987). Consequently, encoding various forms of knowledge in terms of search control rather than conditions on operator proposal is a powerful mechanism for modulating the generality of chunks.

3.4.2. Preventing incorrect chunks

One of the main causes of incorrect chunks is the *exhaustion* problem. One way this can arise is if an operator is proposed with very weak conditions, and made worst so that it will be selected only when the set of other operators is exhausted. This is a convenient programming technique, as illustrated by several operators in the sample implementation (global-constraint-failed, selected when there are neither simulate-trip operators nor a schedule-succeeds operator proposed; trip-succeeds, selected when all simulate-segment operators are exhausted; and compute-resources, selected when the operations for a particular segment are exhausted). The drawback relates to subgoal results based on effects of that operator: the weak proposal conditions present little for backtrace to work with, so the conditions of the resulting chunk will be wildly overgeneral.

The sample implementation shows two responses to the exhaustion problem. The first response, which is to learn nothing, is illustrated in the TAQL constructs (TCs) in Figure 6 (they appear in the code on pages 33 and 34). The *global-constraint-failed* augmentation is added by task*ao*global-constraint-failed and detected by eo*task, which assigns the *novalue* evaluation for that case. This triggers a Soar feature (use of the *quiescence t* goal augmentation) that allows a production to return a result with no chunk being built. A novalue evaluation is interpreted as knowledge that this path is fruitless, with no corresponding knowledge (or desire to generate it) about how to assign credit.

```
(apply-operator task*ao*global-constraint-failed
  :space task
  :op (global-constraint-failed ^reason <reason>)
  (edit :what state
        :new (global-constraint-failed <reason>)))

(evaluate-object eo*task
  :space task
  :what lookahead-state
  ...
  :symbolic-value (novalue
                   :when ((state ^global-constraint-failed)))
  ...)
```

Figure 6: Novalue: backtracking without chunking

A second response is to make up for the lack of constraint in the proposal productions by adding conditions, in either the proposal or the application TC for an operator. This is illustrated in the TC in Figure 7 (which appears in the code on page 36). Sometimes these conditions arise naturally, as when it is necessary to bind information that will be used later. For example, Condition 1 binds the trip, truck, and driver elements that are removed by the edit clause of the TC.

```
(apply-operator simulate-trip*ao*trip-succeeds
  :space simulate-trip
  :op (trip-succeeds ^assignment <new>)

  :when ((operator (car <new>)                                    Condition 1
                  ^trip <trip> ^driver <driver> ^truck <truck>))

  ;; the chunks that update the schedule in the supercontext will
  ;; loop unless we're careful, because one chunk rejects the
  ;; current head of the list, and a second adds the new head.
  ;; we can prevent looping by making the chunks specific to the
  ;; current head of the list.  <current-trip-name> matches a
  ;; constant, and that constant will appear in the chunk:

  :when ((superstate ^current-schedule <current>)
         (operator (car <current>) ^trip <current-trip>)
         (trip <current-trip> ^name <current-trip-name>))        Condition 2

  (edit :what superstate

        ;; push the new trip onto the head of the schedule list:
        :replace (current-schedule :by <new>)

        :remove (trip <trip>)
        :remove (driver <driver>)
        :remove (truck <truck>)))
```

Figure 7: Conditions that bind constants

Frequently, however, building hazard-free chunks requires a comprehensive approach, combining an appropriate representation of the data with correct manipulations at run-time. The Soar manual (beginning on page 119) discusses data representation issues, among them techniques for achieving maximally general (but still correct) chunks. The remainder of this section presents methods of manipulating data to affect chunk generality.

Returning to Figure 7, Condition 2 uses *constant binding* to ensure that chunks built from replacing the head of a list do not cause problems. Constant binding exploits the fact that Soar replaces object identifiers in chunks with variables but leaves constants in place. Thus, binding a constant makes the chunk specific to the name of the element pushed on the list. If this were not done, the chunk built from severing the pointer to the first element of the list would fire whenever that pointer were set.

Condition 2 also has a less obvious effect. Consider the two simulate-trip operators in the trace excerpt in Figure 8 (the excerpt begins on page 44). As the second simulation (*o380*) is being carried out, a constraint violation occurs. The default productions convert the violation into a failure evaluation (for the first simulate-trip operator, since it is the one being evaluated). The failure evaluation results in a search-control chunk (*p946*) that in the future will reject operators similar to *c374* under similar circumstances.

Figure 9 shows p946 (with some of the more ghastly conditions excised; the full chunk appears in the chunk listing on page 52). The effect of Condition 2 is to cause the brown/traveler/trip1 assignment to be rejected specifically when the previous assignment involved *trip3*. There is no

```
34   O: o344 (((brown) (traveler) (trip1) simulate-trip) evaluate-object)
     ...
38      O: c374 ((brown) (traveler) (trip1) simulate-trip)
        ...
68         O: o419 (trip-succeeds)
        ...
69      O: o380 ((gray) (piper) (trip2) simulate-trip)
        ...
112        O: o938 (weight-limit-exceeded local-constraint-failed)
        ...
        Evaluation of operator c374 (simulate-trip) is failure
Build:p946
```

Figure 8: Lookahead leading to search-control chunk p946

clear purpose for this condition, in terms of knowledge usefully captured by the chunk. In fact, this condition makes the chunk overspecific, since the relevant information is only that *trip2* cannot be completed with a small truck (such as *piper*).

Questions of usefulness aside, the trip3 test illustrates how chunking can be affected by

```
(sp p946
   (goal <g2> ^desired <d4> ^problem-space <p1> ^state <d2>
              ^operator <o1> +)
   (problem-space <p1> [...] ^name task)
   (state <d2> ^dummy-att* true ^driver <d3> ^truck <t4> ^trip <t3>
               ^current-schedule <l1>)
   (driver <d3> ^drive-time 12.5 ^name gray)
   (truck <t4> ^volume 400 ^weight-limit 5000 ^type small ^name piper)
   (trip <t3> ^name trip2 ^first-segment <s1>)                          trip2
   (segment <s1> ^name segment1 ^trip trip2 ^source gary ^next-segment <s2>)
   (segment <s2> ^name segment2 ^trip trip2 ^next-segment <s3>)
   (segment <s3> ^name segment3 ^trip trip2 ^next-segment <s7>)
   (segment <s7> ^load-shipment carpet -^load-shipment NIL ^name segment4
                 ^trip trip2 ^next-segment <s8>)
   (segment <s8> ^unload-shipment carpet -^unload-shipment NIL
                 ^name segment5 ^trip trip2 ^next-segment <s9>)
   (segment <s9> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                 ^trip trip2)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)                                              trip3
   (operator <o1> [...] ^name simulate-trip                             <o1>
                  ^trip <t5> ^truck <t1> ^driver <d1>)
   (trip <t5> ^name trip1)                                              trip1
   (truck <t1> ^volume 640 ^weight-limit 10000)
   (driver <d1> ^drive-time 11)
   (goal <g1> ^object NIL ^state <s6>)
   (state <s6> ^shipment <s5> <s4> ^license <l2> ^city <c2>)
   (shipment <s5> ^name newsprint ^weight 6000.0 ^volume 400.0
                  ^load-time 0.2)
   (shipment <s4> ^name carpet ^weight 500.0 ^volume 100.0 ^unload-time 0.25
                  ^load-time 0.2)
   (license <l2> ^truck-type small ^holder gray)
   (city <c2> ^name gary ^truck piper ^driver gray)
 -->
   (goal <g2> ^operator <o1> -))                                        <o1> -
```

Figure 9: The search-control chunk p946

manipulating the data. Figure 10 shows how backtracing comes to include trip3 in the chunk conditions. The simulate-trip*ao*trip-succeeds TC (of Figure 7) replaces the current-schedule augmentation while testing trip3. In Figure 10, the new current-schedule augmentation is *lxxx* (*l* because we know it is a list object, and *xxx* because unlike with operators we can not determine its identifier from the trace). This creates a dependency of lxxx on trip3. When lxxx is tested during the application of o938 (the simulate-trip*ao*local-constraint-failed TC is on page 37), the dependency is extended to the subgoal result. When the chunk is built, o938 acts as a bridge from the result to lxxx, and o419 acts as a bridge from lxxx to trip3. Trip3 was copied directly from the higher task context, so backtracing includes it in the chunk conditions. The general technique for ensuring that a particular condition (based on a supercontext augmentation) is included in a chunk is to ensure that each operator in the subgoal extends the dependency, by creating an augmentation while testing a previous augmentation that depends on the one from the supercontext.

Note that the augmentation created by one operator to be tested by the next need not be meaningful beyond the function of extending the dependency. In a variant known locally as *beading*, the augmentations are simply new identifiers, each replaced by the next. Backtracing then strings up the identifiers like beads, backtracing through whatever augmentations were tested in creating them.[6]

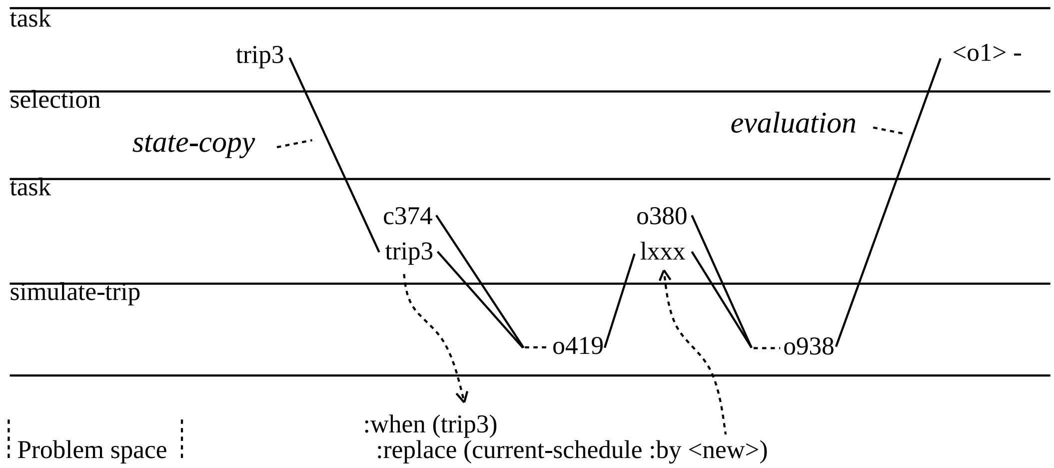

Figure 10: Backtracing to the trip3 condition

A second instance of constant binding, shown in Figure 11, does provide a useful constraint on p946. During the application of o380 (between decision cycles 69 and 112 in Figure 8), Condition 3 causes *trip2* to be included in the backtrace and eventually in the chunk conditions (Figure 9). This test is semantically relevant to the constraint that small trucks are not able to complete this trip.

For completeness, we also account for the appearance of *trip1* (the third and last trip name) in the conditions of p946. The trip1 test occurs because of the simulate-trip*ao*local-constraint-failed TC (page 37), which applies o938. The relevant condition (corresponding to Condition 2)

[6]The bead metaphor is due to Rick Lewis.

```
(apply-operator simulate-segment*ao*compute-resources
  :space simulate-segment
  :op compute-resources
  ...
  ;; add the updated resources to the object:
  (edit :what (:none resources <resources>)
    ...
    ;; add the trip, to make sure that the segment and trip stay
    ;; tied together in the chunks that update the resources.
    ;; if we don't do this, the chunks could transfer
    ;; incorrectly to the same segment of another trip.

    :new (trip <trip-name>
           :when ((segment &lt;segment&gt; ^trip <trip-name>)))        Condition 3
    ...
  )
```

Figure 11: Conditions that prevent overgeneral search-control chunks

tests the name of the last trip pushed onto the current schedule (trip1, by o419). Note that o938 accesses trip1 through lxxx, extending the dependency of the subgoal result on trip3 as described at the top of the previous page.

3.4.3. Expensive chunks

The problem of *expensive chunks* (Tambe et al., 1990) is a manifestation in Soar of the utility problem in explanation-based learning (Minton, 1985). It can arise whenever a representation uses sets (multi-attributes), such as that of the top state of the sample implementation.

The chunk in Figure 12 implements the task operator (the chunk also appears on page 56). It reads the top-state and creates the solution (the actions are elided). It is expensive to match because of the combinatorics in the <s4> condition (second from the top). The *^license* test can match 3^3 ways, the *^city* test 2^2 ways, the *^driver* test 3^3 ways, etc. The match cost is proportional to the product of these terms.

For this task, a set-based representation is a great convenience. However, the general solution to expensive chunks is to use structured representations, such as lists.

```
(sp p1548
      (goal <g1> ^object NIL ^state <s4> ^problem-space <p1> ^operator <o1>)
      (state <s4> ^dummy-att* true ^license <l3> <l2> <l1> ^city <c2> <c1>
                               ^driver <d2> <d1> <d3> ^truck <t4> <t3> <t5>
                               ^trip <t6> <t2> <t1>)
      (problem-space <p1> ^name top-space)
      (license <l3> ^truck-type small ^holder gray)
      (city <c2> ^name gary ^truck piper traveler ^driver gray brown)
      (license <l2> ^truck-type big ^holder green)
      (city <c1> ^name indy ^truck cannonball ^driver green)
      (license <l1> ^truck-type medium ^holder brown)
      (driver <d2> ^name gray ^drive-time 12.5)
      (truck <t4> ^type small ^name piper ^volume 400 ^weight-limit 5000)
      (trip <t6> ^name trip1 ^first-segment <s3>)
      (segment <s3> ^source gary)
      (driver <d1> ^drive-time 11 ^name green)
      (truck <t3> ^volume 1280 ^weight-limit 32000 ^type big ^name cannonball)
      (trip <t2> ^name trip3 ^first-segment <s2>)
      (segment <s2> ^source indy)
      (driver <d3> ^drive-time 11 ^name brown)
      (truck <t5> ^volume 640 ^weight-limit 10000 ^type medium ^name traveler)
      (trip <t1> ^name trip2 ^first-segment <s1>)
      (segment <s1> ^source gary)
      (operator <o1> ^name do-task ^control-stuff* <c3>)
      (control-stuff* <c3> ^edit-from-subgoal-enabled* true)
 -->
      ...)
```

Figure 12: An expensive chunk

4. Sample Implementation in Soar

4.1. Introduction

This chapter presents a sample implementation for the shipment scheduling assistant. This implementation serves as a reference for the design given in Chapter 3.

Section 4.2 is a program listing, organized by problem space. Section 4.3 is an execution trace, followed by statistics on the run and the source code. Section 4.4 is a listing of the chunks generated during the run, useful for cross-referencing with the trace, and as a reference for the discussion of learning in the previous chapter. For information on obtaining on-line versions of these files, send mail to soar-requests@cs.cmu.edu.

4.2. Program Listing

```
;;;; -*- Mode: TAQL -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File            : /afs/cs.cmu.edu/user/altmann/taql/truck/truck.taql
;;;; Author          : Erik Altmann
;;;; Created On      : Sat Sep 28 20:56:38 1991
;;;; Last Modified By: Erik Altmann
;;;; Last Modified On: Wed Jun  3 20:14:15 1992
;;;; Update Count    : 113
;;;; Soar Version    : 5.2.1
;;;; TAQL Version    : 3.1.4
;;;;
;;;; PURPOSE
;;;;    Soar implementation of the Shipment Scheduling Assistant,
;;;; a.k.a. the Trucking Task.  Runs with chunking on.  See trace.txt
;;;; for a trace, and chunks.soar for the corresponding chunks.
;;;;
;;;; TABLE OF CONTENTS
;;;;    In terms of spaces:  top-space, task, simulate-trip, and
;;;; simulate-segment.  To reach the code for SPACE, search for
;;;; "ps*SPACE", which will locate the propose-space TAQL construct
;;;; for that space.
;;;;
;;;; (C) Copyright 1991, Carnegie Mellon University, all rights reserved.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; object/attribute pairs for Soar trace.  the value of each attribute
;; will be printed.

(trace-attributes '(;; attributes of the simulate-trip operator (in the task space):
                    (operator driver)
                    (operator truck)
                    (operator trip)

                    ;; attributes of operators in the simulate-trip space:
                    (operator segment)      ; simulate-segment
                    (operator reason)       ; trip-succeeds

                    ;; attributes of operators in the simulate-segment space:
                    (operator shipment-name)
                    (operator volume)
                    (operator load-time)
                    (operator unload-time)
                    (operator weight)
                    (operator time)
                    ))
```

```
;; ----------------------
;; top-space:
;; ----------------------

;; the top context contains the task state and the task operator.  the
;; task-state contains all the static information for the task.  the
;; task operator is bare.

;; the space is proposed automatically by taql's run-time support, so
;; there is no ps*top-space tc.

(propose-task-state pts*task

  :new (weather rain)

  :new (driver
        ((driver ^name green ^union yes ^drive-time 11))
        ((driver ^name white ^union no ^drive-time 12.5))
        ((driver ^name brown ^union yes ^drive-time 11))
        ((driver ^name gray ^union no ^drive-time 12.5)))

  ;; the hierarchy of licenses is implicit in the ^holders:
  ;; everyone holds type 1, fewer hold type 2, fewest hold type 3:

  :new (license
        ((license ^name class3 ^truck-type big
                  ^holder green + &, brown + &))
        ((license ^name class2 ^truck-type medium
                  ^holder green + &, brown + &, white + &))
        ((license ^name class1 ^truck-type small
                  ^holder green + &, brown + &, white + &, gray + &)))

  :new (truck
        ((truck ^name cannonball ^type big ^weight-limit 32000 ^volume 1280))
        ((truck ^name piper ^type small ^weight-limit 5000 ^volume 400))
        ((truck ^name traveler ^type medium ^weight-limit 10000 ^volume 640))
        ((truck ^name queen-bee ^type medium ^weight-limit 10000 ^volume 640)))

  :new (city
        ((city ^name gary ^state indiana
               ^driver brown + &, gray + &
               ^truck piper + &, traveler + &))
        ((city ^name indy ^state indiana
               ^driver green + &, white + &
               ^truck cannonball + &, queen-bee + &))
        ((city ^name utica ^driver nil ^truck nil ^state illinois))
        ((city ^name viola ^driver nil ^truck nil ^state illinois))
        ((city ^name warsaw ^driver nil ^truck nil ^state indiana))
        ((city ^name kokomo ^driver nil ^truck nil ^state indiana))
        ((city ^name attica ^driver nil ^truck nil ^state indiana))
        ((city ^name cook ^driver nil ^truck nil ^state indiana)))

  :new (shipment
        ((shipment ^name typewriters ^weight 1000.0 ^volume 200.0
                   ^load-time .2 ^unload-time .25))
        ((shipment ^name carpet ^weight 500.0 ^volume 100.0
                   ^load-time .2 ^unload-time .25))
        ((shipment ^name newsprint ^weight 6000.0 ^volume 400.0
                   ^load-time .2 ^unload-time .25)))
```

```
;; trip 1:
:new (trip
      ((trip ^name trip1 ^first-segment <s1>)
       (segment <s1> ^name segment1 ^road u30 ^source gary ^dest warsaw
                ^load-shipment typewriters
                ^unload-shipment nil
                ^trip trip1
                ^next-segment <s2>)
       (segment <s2> ^name segment2 ^road u31a ^source warsaw ^dest kokomo
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip1
                ^next-segment <s3>)
       (segment <s3> ^name segment3 ^road u31b ^source kokomo ^dest indy
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip1
                ^next-segment <s4>)
       (segment <s4> ^name segment4 ^road i74 ^source indianapolis ^dest attica
                ^load-shipment nil
                ^unload-shipment typewriters
                ^trip trip1)))

;; trip 2:
:new (trip
      ((trip ^name trip2 ^first-segment <s1>)
       (segment <s1> ^name segment1 ^road u41a ^source gary ^dest cook
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip2
                ^next-segment <s2>)
       (segment <s2> ^name segment2 ^road i80b ^source cook ^dest utica
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip2
                ^next-segment <s3>)
       (segment <s3> ^name segment3 ^road i80a ^source utica ^dest viola
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip2
                ^next-segment <s4>)
       (segment <s4> ^name segment4 ^road i80a ^source viola ^dest utica
                ^load-shipment carpet
                ^unload-shipment nil
                ^trip trip2
                ^next-segment <s5>)
       (segment <s5> ^name segment5 ^road i80b ^source utica ^dest cook
                ^load-shipment nil
                ^unload-shipment carpet
                ^trip trip2
                ^next-segment <s6>)
       ;; note the several nil fields.  trip2 ends oddly, with doing a load:
       (segment <s6> ^name segment6 ^road nil ^source cook ^dest nil
                ^load-shipment newsprint
                ^unload-shipment nil
                ^trip trip2)))

;; trip 3:
:new (trip
      ((trip ^name trip3 ^first-segment <s1>)
       (segment <s1> ^name segment1 ^road i70b ^source indy ^dest yale
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip3
                ^next-segment <s2>)
       (segment <s2> ^name segment2 ^road u41e ^source yale ^dest attica
                ^load-shipment nil
                ^unload-shipment nil
                ^trip trip3)))
```

```
  :new (road
        ((road ^name u30 ^grade secondary ^length 70.0))
        ((road ^name u31a ^grade secondary ^length 70.0))
        ((road ^name u31b ^grade primary ^length 40.0))
        ((road ^name i74 ^grade primary ^length 60.0))

        ((road ^name u41a ^grade secondary ^length 20.0))
        ((road ^name i80b ^grade primary ^length 90.0))
        ((road ^name i80a ^grade primary ^length 100.0))

        ((road ^name i70b ^grade primary ^length 90.0))
        ((road ^name u41e ^grade secondary ^length 90.0))))

;; entailments that encode maximum speeds for various road grades,
;; given the weather.  the speeds augment the corresponding road
;; objects.
;;
;;   entailments for fair and snow are missing, but aren't necessary
;;   for the given task.

(augment a*top-space*speed-when-rainy-on-primary
  :space top-space
  :when ((top-state ^weather rain ^road <r>))
  :what (:none road <r>)
  :new (speed 55)
  :when ((road <r> ^grade primary)))

(augment a*top-space*speed-when-rainy-on-secondary
  :space top-space
  :when ((top-state ^weather rain ^road <r>))
  :what (:none road <r>)
  :new (speed 50)
  :when ((road <r> ^grade secondary)))

(augment a*top-space*speed-when-rainy-on-tertiary
  :space top-space
  :when ((top-state ^weather rain ^road <r>))
  :what (:none road <r>)
  :new (speed 35)
  :when ((road <r> ^grade tertiary)))

;; once the speeds are computed, the time to traverse each road can be
;; computed, given the length.  the times also augment the road
;; object.

(augment a*top-space*time-from-length-and-speed
  :space top-space
  :when ((state ^road <r>)
         (road <r> ^length <l> ^speed <s>))
  :what (:none road <r>)
  :new (time (compute <l> / <s>)))

;; propose doing the scheduling task:

(propose-task-operator pto*task
  :op do-task)
```

```
;;; ----------------------
;;; the task space
;;; ----------------------

;; applies the schedule operator from the top-space, by trying various
;; assignments of trucks and drivers to trips.

(propose-space ps*task
  :function (apply operator do-task)
  :space task)

(propose-initial-state pis*task
  :space task

  ;; create a set each of the available drivers and trucks and the
  ;; trips that have to be covered.  we'll update this set as
  ;; assignments are made (simulate-trip*ao*trip-succeeds).

  :copy (driver truck trip)

  ;; initialize the current-schedule.  use a dummy simulate-trip
  ;; operator, so that the various conditions that test the name of
  ;; the last assignment on the current-schedule for chunking purposes
  ;; will work without a special case for the empty list.
  ;;
  ;;   note: below, "list" is a data macro.  this clause creates a
  ;;   list whose car is the dummy operator.  the syntax used is the
  ;;   third form of ACTION-SPEC (see the TAQL manual, p. 52).

  :new ((^current-schedule (list <dummy>))
        (operator <dummy> ^name simulate-trip ^trip <dummy-trip>)
        (trip <dummy-trip> ^name dummy)))
```

```
;; generate an assignment (a <driver, truck, trip> tuple), and propose
;; simulating the trip.
;;
;;   the conditions of the operator proposal generate the assignments
;;   that embody the immediate constraints, like the truck and the
;;   driver being in the first city of the trip.  these constraints
;;   are easy to test, given the structure of the information on
;;   the top state.

(propose-operator task*po*simulate-trip
  :space task
  :when (;; bind a driver, truck, and city:
         (state ^driver <driver> ^truck <truck> ^trip <trip>)

         ;; constrain the driver and truck to be in the same city:
         (top-state ^city <city>)
         (city <city> ^name <city-name> ^truck <truck-name> ^driver <driver-name>)
         (truck <truck> ^name <truck-name>)
         (trip <trip> ^name <trip-name> ^first-segment &lt;segment&gt;)
         (segment &lt;segment&gt; ^source <city-name>)

         ;; constrain the driver to be licensed to drive the truck:
         (top-state ^license <license>)
         (driver <driver> ^name <driver-name>)
         (truck <truck> ^type <truck-type>)
         (license <license> ^truck-type <truck-type> ^holder <driver-name>))

  :op (simulate-trip ^driver <driver> ^truck <truck> ^trip <trip>))

;; simulate-trip is applied in a subgoal, by the simulate-trip space.
;; it succeeds if no local constraints are violated, such as resources
;; going negative.  the termination conditions are in the next two TCs.
;;
;;   note: the simulate-trip operator itself is as an easy way to
;;   represent an assignment, so we build up the schedule by adding
;;   simulate-trip operators to a list, called ^current-schedule.

;; simulate-trip succeeds when the simulate-trip operator itself has
;; been added to the current schedule (by the subgoal).  terminate the
;; operator when this happens:

(apply-operator task*ao*simulate-trip*trip-succeeded
  :space task
  :op simulate-trip
  :terminate-when ((state ^current-schedule <schedule>) ; bind "cons cell"
                   (operator (car <schedule>) ^trip <trip>) ; bind its "car"
                   (operator ^trip <trip>)))

;; terminate simulate-trip if the subgoal signals an local constraint
;; violation.  local-constraint-failed is detected as a failure
;; condition by eo*task (below).
;;
;;   note: multiple :terminate-when clauses in one TC are conjunctive,
;;   so we need two apply-operator TCs to represent the two conditions.

(apply-operator task*ao*simulate-trip*constraint-violation
  :space task
  :op simulate-trip
  :terminate-when ((state ^local-constraint-failed)))
```

```
;; when there are no trips left, we're done:

(propose-operator task*po*schedule-succeeds
  :space task
  :when ((state ^current-schedule <last-assignment> - ^trip))
  :op (schedule-succeeds ^complete-schedule <last-assignment>
                         ^reason no-trips-left)

  ;; bind the name of the previous trip.  this causes the search
  ;; control chunks that arise from detecting the success of this trip
  ;; to include the previous trip name in their conditions:

  :when ((operator (car <last-assignment>) ^trip <last-trip>))
  )

;; we know that the current schedule is complete, so put that
;; information on the state.  this signals success (via eo*task, below).

(apply-operator task*ao*schedule-succeeds
  :space task
  :op (schedule-succeeds ^complete-schedule <schedule>)
  (edit :what state
        :new (complete-schedule <schedule>)))

;; be prepared for a global constraint violation.  propose an operator
;; that represents such a violation occurring, then make it worst.  if
;; at some point there are trips left but no assignments for them,
;; then neither simulate-trip nor schedule-succeeds will be proposed,
;; so global-constraint-failed will be selected by default.
;;
;;   note: this illustrates a general programming technique, which is
;;   to use the selection of an operator with a worst preferences to
;;   implicitly represent the knowledge that each of a set of options
;;   has been exhausted.  however, because such a selection is not
;;   based on any explicit knowledge that can be backtraced through,
;;   chunks learned from results of such operators can be vastly
;;   overgeneral.  one solution is to not learn from such results (see
;;   eo*task).

(propose-operator task*po*global-constraint-failed
  :space task
  :when ((state ^trip))
  :op (global-constraint-failed ^reason trips-level-over))

(prefer task*p*global-constraint-failed*worst
  :space task
  :op global-constraint-failed
  :value worst)

(apply-operator task*ao*global-constraint-failed
  :space task
  :op (global-constraint-failed ^reason <reason>)
  (edit :what state
        :new (global-constraint-failed <reason>)))
```

```
;; signal task success and local and global failures.  this TC causes
;; a lookahead instance of the task space to exit, causing another
;; simulate-trip alternative to be tried out.

(evaluate-object eo*task
  :space task
  :what lookahead-state

  ;; succeed when there's a complete schedule (generated by
  ;; task*ao*schedule-succeeds).  Soar's default productions
  ;; propagate success to the top of the lookahead-search stack,
  ;; causing Soar to learn search-control chunks that remember the
  ;; sequence of simulate-trip operators.

  :symbolic-value (success
                  :when ((state ^complete-schedule)))

  ;; backtrack if there's been a global-constraint-failed.  the novalue
  ;; evaluation tells Soar not to learn anything.  we don't know
  ;; which assignment caused the problem or why, so there's nothing
  ;; useful to learn (see note at task*po*global-constraint-failed).

  :symbolic-value (novalue
                  :when ((state ^global-constraint-failed)))

  ;; backtrack (and learn that the last assignment was a bad choice)
  ;; if there's a local-constraint-failed (created by
  ;; task*ao*simulate-trip*constraint-violation).

  :symbolic-value (failure
                  :when ((state ^local-constraint-failed))))

;; goal-test knowledge, to detect success and failure when not in
;; lookahead search.  replicates the knowledge in eo*task.

(goal-test-group gtg*task*success
  :space task
  :group-type success
  :when ((superspace ^name top-space)
         (state ^complete-schedule)))

(goal-test-group gtg*task*failure
  :space task
  :group-type failure
  :when ((superspace ^name top-space))

  ;; disjunctive failure test.
  :test (failure
         :when ((state ^global-constraint-failed)))
  :test (failure
         :when ((state ^local-constraint-failed))))

;; return the complete schedule to the top state.
;;
;;   note: with this TC, Soar builds a chunk that applies the task
;;   operator (proposed by pto*task) after learning.  without this TC,
;;   no chunk will be learned for the task operator (TAQL's runtime
;;   support automatically terminates the task operator when a
;;   successful or failed state is reached in the space that applies
;;   it); after learning, the task operator will impasse, and chunks
;;   learned during lookahead search will guide Soar directly to a
;;   solution in the subspace without further impasses.

(result-superstate rs*task*success
  :space task
  :group-type success
  (edit :what superstate
        :when ((state ^complete-schedule <schedule>))
        :new (complete-schedule <schedule>)))
```

```
;;; --------------------------
;;; the simulate-trip space
;;; --------------------------

;; applies the simulate-trip operator from the schedule space.

;; if the simulation succeeds, trip-succeeds pushes the superoperator
;; onto the current-schedule list on the superstate.  if it fails (due
;; to a local constraint failure), local-constraint-failed pushes this
;; assignment onto the current schedule, and returns that as the value
;; of ^local-constraint-failed.

(propose-space ps*simulate-trip
  :function (apply operator simulate-trip)
  :space simulate-trip)

;; copy the trip/truck/driver set down from the superstate, for
;; convenience.  establish the volume and weight-limit of the truck,
;; and the maximum time the driver can drive, as initial values of
;; those resources.

(propose-initial-state pis*simulate-trip
  :space simulate-trip
  :when ((superoperator ^truck <truck> ^trip <trip> ^driver <driver>)
         (driver <driver> ^drive-time <drive-time>)
         (truck <truck> ^volume <volume> ^weight-limit <weight-limit>))
  :new (driver <driver>)
  :new (truck <truck>)
  :new (trip <trip>)
  :new (initial-resources ((resources ^current-time <drive-time>
                                      ^current-volume <volume>
                                      ^current-weight <weight-limit>))))

;; simulate the driver driving the truck over each segment of the trip
;; in turn.  after each segment, the simulate-segment operator updates
;; the resources.

(propose-operator simulate-trip*po*simulate-segment
  :space simulate-trip
  :select-once-only                   ; so first segment is only done once

  ;; propose doing the first segment, using the initial resources:
  :op (simulate-segment ^segment <first> ^resources (list <res>)
                       :when ((state ^trip <trip> ^initial-resources <res>)
                              (trip <trip> ^first-segment <first>)))

  ;; at each segment, propose doing the next one, using the current
  ;; resources:
  :op (simulate-segment ^segment <next> ^resources <res>
                       :when ((operator ^segment <current>)
                              (segment <current> ^next-segment <next>)
                              (state ^current-resources <res>))))

;; operator simulate-segment is applied in a subgoal, in the
;; simulate-segment space.  the space returns a resource object,
;; pushing it onto the list ^current-resources (creating the list if
;; it has to).  the resource object specifies the segment it was
;; constructed in, allowing us to test we can terminate the operator.
;; the resource object also specifies ^failed true if any resource
;; (volume, weight-limit, time) ran out during the segment, or if
;; other constraints were violated (see
;; simulate-segment*ao*compute-resources).

(apply-operator simulate-trip*ao*simulate-segment
  :space simulate-trip
  :op (simulate-segment ^segment &lt;segment&gt;)
  :terminate-when ((state  ^current-resources <res>)
                   (resources (car <res>) ^segment <segment-name>)
                   (segment &lt;segment&gt; ^name <segment-name>)))
```

```
;; be prepared to declare the simulation successful.  propose
;; trip-succeeds and make it worst, so that it's selected only
;; when there are no segments left to drive over.
;;
;;   in the proposal, create a new list whose car is the current
;;   assignment, and whose cdr is the existing schedule from the
;;   superspace.  make this new list an argument to the operator,
;;   which can return it directly to the superstate (becoming the
;;   updated schedule).

(propose-operator simulate-trip*po*trip-succeeds
  :space simulate-trip

  ;; bind the superoperator, and the current schedule:

  :when ((superstate ^current-schedule <last-assignment>)
         (supergoal ^operator <so>))

  ;; put the updated schedule on the operator, so that the operator
  ;; application TC can return the updated schedule to the
  ;; superstate.  (updating and returning requires two steps, so
  ;; doing the update here allows the apply-operator TC to return
  ;; the result.)

  :op (trip-succeeds ^assignment (cons <so> <last-assignment>)))

(prefer simulate-trip*p*trip-succeeds*worst
  :space simulate-trip
  :op trip-succeeds
  :value worst)

;; if the assignment succeeds, return the new schedule to the
;; supercontext, and update the superstate's trip/truck/driver sets.

(apply-operator simulate-trip*ao*trip-succeeds
  :space simulate-trip
  :op (trip-succeeds ^assignment <new>)

  ;; bind the information we need to update the trip/truck/driver sets:

  :when ((operator (car <new>)
                   ^trip <trip> ^driver <driver> ^truck <truck>))


  ;; the chunks that update the schedule in the supercontext will
  ;; loop unless we're careful, because one chunk rejects the
  ;; current head of the list, and a second adds the new head.
  ;; we can prevent looping by making the chunks specific to the
  ;; current head of the list.  <current-trip-name> matches a
  ;; constant, and that constant will appear in the chunk:

  :when ((superstate ^current-schedule <current>)
         (operator (car <current>) ^trip <current-trip>)
         (trip <current-trip> ^name <current-trip-name>))

  (edit :what superstate

        ;; push the new trip onto the head of the schedule list:
        :replace (current-schedule :by <new>)

        ;; trip is taken care of:
        :remove (trip <trip>)

        ;; driver and truck are used up:
        :remove (driver <driver>)
        :remove (truck <truck>)))
```

```
;; the simulation fails if a resource went negative.  the value of
;; ^reason is printed in the trace (see trace-attributes).

(propose-operator simulate-trip*po*local-constraint-failed
  :space simulate-trip

  ;; as in simulate-task*po*trip-succeeds:
  :when ((supergoal ^operator <so>)
         (superstate ^current-schedule <last-assignment>))
  :op (local-constraint-failed ^failed-assignment (cons <so> <last-assignment>)
                               ^reason <reason>)

  ;; failure condition:
  :when ((state ^current-resources <res>)
         (resources (car <res>) ^failed true ^reason <reason>)))


;; if local-constraint-failed is proposed, select it immediately.

(prefer simulate-trip*p*local-constraint-failed*require
  :space simulate-trip
  :op local-constraint-failed
  :value require)

;; if the simulation fails, return local-constraint-failed to the
;; supercontext.  don't bother updating the trip/truck/driver sets,
;; because the supercontext will signal a failure condition and exit

(apply-operator simulate-trip*ao*local-constraint-failed
  :space simulate-trip
  :op (local-constraint-failed ^failed-assignment <schedule>)

  ;; to prevent chunks that will loop in other cases (as in
  ;; simulate-trip*ao*trip-succeeds):
  :when ((superstate ^current-schedule <current>)
         (operator (car <current>) ^trip <current-trip>)
         (trip <current-trip> ^name <current-trip-name>))

  ;; failure termination condition on the superoperator:
  (edit :what superstate
        :new (local-constraint-failed <schedule>)))
```

```
;; ----------------------------
;; the simulate-segment space
;; ----------------------------

;; applies the simulate-segment operator from the simulate-trip space.
;;
;; the space succeeds if the segment can be completed with the volume
;; and weight currently available in the truck, and if the time left
;; the driver is enough to cover loading, driving, and unloading.  it
;; fails otherwise.  the return value is a resources object, pushed
;; onto the ^current-resources list of the superstate.  the list is
;; created if it isn't there.  if the segment fails, the resources
;; object is flagged with ^failed true and a ^reason.
;;
;; the compute-resources operator represents the oddball constraint
;; that white can't drive through illinois.

(propose-space ps*simulate-segment
  :function (apply operator simulate-segment)
  :space simulate-segment)

;; the arguments to the superoperator are the segment to do and the
;; resources to do it with.  unpack the resources so we can modify
;; them locally.  unpack the segment for convenience.

(propose-initial-state pis*simulate-segment
  :space simulate-segment
  :when ((superoperator ^segment <seg> ^resources <res>)
         (resources (car <res>)
                    ^current-volume <volume> ^current-time <time> ^current-weight <weight>))
  :new (segment <seg>)
  :new (current-volume <volume>)
  :new (current-time <time>)
  :new (current-weight <weight>))

;; when the ^shipment-name of a segment is non-nil, then propose loading
;; that shipment:

(propose-operator simulate-segment*po*load
  :space simulate-segment
  :select-once-only                     ; load only once per segment
  :op (load ^shipment-name <shipment-name> ^volume <volume> ^load-time <load-time>
          ^weight <weight>)
  :when ((top-state ^shipment <sh>)
         (shipment <sh> ^name <shipment-name> ^weight <weight>
                  ^volume <volume> ^load-time <load-time>))
  :when ((state ^segment <seg>)
         (segment <seg> - ^load-shipment nil ^load-shipment <shipment-name>)))

;; load the shipment by adjusting the resources:

(apply-operator simulate-segment*ao*load
  :space simulate-segment
  :op (load ^weight <weight> ^volume <volume> ^load-time <load-time>)
  :when ((state ^current-weight <current-weight> ^current-volume <current-volume>
               ^current-time <current-time>))
  (edit :what state
        :replace (current-weight :by (compute <current-weight> - <weight>))
        :replace (current-volume :by (compute <current-volume> - <volume>))
        :replace (current-time :by (compute <current-time> - <load-time>))))
```

```
;; similarly for unloading.  for correctness, loading has to be done before
;; unloading (see the explanation above simulate-segmentc*load*unload*better).

(propose-operator simulate-segment*po*unload
  :space simulate-segment
  :select-once-only                     ; unload only once per segment
  :op (unload ^shipment-name <shipment-name> ^volume <volume>
             ^unload-time <unload-time> ^weight <weight>)
  :when ((top-state ^shipment <sh>)
         (shipment <sh> ^name <shipment-name> ^weight <weight>
                  ^volume <volume> ^unload-time <unload-time>))
  :when ((state ^segment <seg>)
         (segment <seg> - ^unload-shipment nil ^unload-shipment <shipment-name>)))

(apply-operator simulate-segment*ao*unload
  :space simulate-segment
  :op (unload ^weight <weight> ^volume <volume> ^unload-time <unload-time>)
  :when ((state ^current-weight <current-weight> ^current-volume <current-volume>
               ^current-time <current-time>))
  (edit :what state
        :replace (current-weight :by (compute <current-weight> + <weight>))
        :replace (current-volume :by (compute <current-volume> + <volume>))
        :replace (current-time :by (compute <current-time> - <unload-time>))))

;; adjust the time resource by the length of time it takes to drive
;; the segment.

(propose-operator simulate-segment*po*drive
  :space simulate-segment
  :select-once-only                     ; drive only once per segment
  :op (drive ^time <time>)
  :when ((top-state ^road <road>)
         (state ^segment <seg> ^road <road-name>)
         (segment <seg> - ^dest nil)
         (road <road> ^name <road-name> ^time <time>))) ; entailed

(apply-operator simulate-segment*ao*drive
  :space simulate-segment
  :op (drive ^time <time>)
  :when ((state ^current-time <current-time>))
  (edit :what state
        :replace (current-time :by
                              (compute <current-time> - <time>))))

;; be prepared to update resources when no more resource-consuming
;; actions remain.

(propose-operator simulate-segment*po*compute-resources
  :space simulate-segment
  :op compute-resources)

(prefer simulate-segment*p*compute-resources
  :space simulate-segment
  :op compute-resources
  :value worst)

;; update resources by building a new resource object on the current state.

(apply-operator simulate-segment*ao*compute-resources
  :space simulate-segment
  :op compute-resources

  ;; bind information required to build the resource object:
  :when ((state ^current-weight <weight>
               ^current-volume <volume>
               ^current-time <time>
               ^segment &lt;segment&gt;))
```

```
;; create a new resource object and attach it to the current state:
:bind <resources>
(edit :what state
      :new (resources <resources>))

;; add the updated resources to the object:
(edit :what (:none resources <resources>)

      ;; add the segment, so that when the resource object is returned
      ;; to the superstate, the superoperator will be able to tell
      ;; when to terminate (its segment parameter will be the same as
      ;; the segment of the head of the current-resources list):

      :new (segment <segment-name>
                    :when ((segment &lt;segment&gt; ^name <segment-name>)))

      ;; add the trip, to make sure that the segment and trip stay
      ;; tied together in the chunks that update the resources.
      ;; if we don't do this, the chunks could transfer
      ;; incorrectly to the same segment of another trip.

      :new (trip <trip-name>
                 :when ((segment &lt;segment&gt; ^trip <trip-name>)))

      ;; add the resources:

      :new (current-weight <weight>)
      :new (current-volume <volume>)
      :new (current-time <time>))

;; failure conditions arise when a resource has fallen below zero:

(edit :what (:none resources <resources>)
      :new (failed true)
      :new (reason weight-limit-exceeded)
      :when ((state ^current-weight < 0.0)))

(edit :what (:none resources <resources>)
      :new (failed true)
      :new (reason volume-exceeded)
      :when ((state ^current-volume < 0.0)))

(edit :what (:none resources <resources>)
      :new (failed true)
      :new (reason time-exceeded)
      :when ((state ^current-time < 0.0)))

;; this is the failure condition in which white is wanted for a
;; crime in illinois.  the chunks built are more specific than
;; they need to be, because they include all sorts of other
;; resources that have nothing to do with failure condition.

(edit :what (:none resources <resources>)
      :new (failed true)
      :new (reason whites-a-criminal-in-illinois)
      :when ((top-state ^city <city>)
             (city <city> ^name <city-name> ^state illinois)
             (state ^segment &lt;segment&gt;)
             (superstate ^driver <driver>)
             (driver <driver> ^name white)))

)            ; end of simulate-segment*ao*compute-resources
```

```
;; when the resources object is built, propose returning it:

(propose-operator simulate-segment*po*update-resources
  :space simulate-segment
  :op (update-resources ^resources <res>)
  :when ((state ^resources <res>)))

;; push the resource object onto the list of current resources.  if
;; the list has not been created yet, create it.

(apply-operator simulate-segment*ao*update-resources
  :space simulate-segment
  :op (update-resources ^resources <res>)

  ;; bind the name of the current segment (see below):
  :when ((state ^segment &lt;segment&gt;)
         (segment &lt;segment&gt; ^name <segment-name>))

  (edit :what superstate

        ;; if the current-resources list exists already:
        :replace (current-resources
                  :by (cons <res> <res-list>)
                  :when ((superstate ^current-resources <res-list>)
                         ;; prevent looping chunks:
                         (resources (car <res-list>) - ^segment <segment-name>)))

        ;; if there's no current-resources list:
        :new (current-resources (list <res>)
                                :when ((superstate - ^current-resources)))))

;; for a given segment, ^load-shipment non-nil means load at the source
;; city, and ^unload-shipment non-nil means unload at the destination.
;; so for correctness wrt weight-limit and volume, loads have to come first.

(compare c*load*unload*better
  :space simulate-segment
  :op1 load
  :op2 unload
  :relation better)

;; other than that, it doesn't matter what order resources are
;; consumed in, but it doesn't hurt to do it in a sensible one.

(compare c*load*drive*better
  :space simulate-segment
  :op1 load
  :op2 drive
  :relation better)

(compare c*drive*unload*better
  :space simulate-segment
  :op1 drive
  :op2 unload
  :relation better)
```

```
;; ------------------------
;; soar hacks:
;; ------------------------

;; this production replaces a soar default production that monitors
;; evaluations of lookahead operators.  it prints the id of the
;; operator copy, rather than the source of that copy.  this is an
;; improvement, because the copy shows up in the trace, so the id is
;; useful for reference.

(sp default*monitor*operator*evaluation
    (goal <top> ^object nil -^verbose false)
    (goal <g> ^object <sg> ^state <s>)
    (state <s> ^tried-tied-operator <obj>)
    (goal <sg> ^operator <so>)
    (operator <so> ^type evaluation ^evaluation <e>)
    (evaluation <e> ^ << numeric-value symbolic-value >> <n>)
    (<class> <obj> ^name <name>)
    -->
    (tabstop <tab>)
    (write2 (crlf) (tabto <tab>) "  Evaluation of " <class> " " <obj> " ("
        <name> ") is " <n>))

;;; end of code
```

4.3. Execution Trace

```
;; -*- Mode: Indented-Text -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; File            : /afs/cs.cmu.edu/user/altmann/taql/truck/trace.txt
;; Author          : Erik Altmann
;; Created On      : Wed Jun  3 20:25:47 1992
;; Last Modified By: Erik Altmann
;; Last Modified On: Wed Jun  3 20:26:34 1992
;; Update Count    : 1
;;
;; PURPOSE
;;  Trace of a sample implementation of the Trucking Task, aka the
;; Shipment Scheduling Assistant.  The source code is in truck.taql, and
;; chunks produced from this trace are in chunks.soar.
;;
;; TABLE OF CONTENTS
;;   The trace, followed by print-stats and taql-stats.
;;
;; Copyright 1992, Carnegie Mellon University.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Starting Soar5 ...
Allegro CL 3.1.12.2 [DECstation] (11/19/90)
Copyright (C) 1985-1990, Franz Inc., Berkeley, CA, USA

Soar (Version 5, Release 2)
Created August 26th, 1991
Bugs and questions should be sent to Soar-bugs@cs.cmu.edu
The current bug-list may be obtained by sending mail to
Soar-bugs@cs.cmu.edu with the Subject: line "bug list"
This software is in the public domain.

This software is made available AS IS, and Carnegie Mellon
University and the University of Michigan make no warranty
about the software or its performance.

See (soarnews) for news.
; Loading /usr0/altmann/.soar-init.lisp.
; Loading /usr/misc/.Soar5/bin/Soar5.latest.patches.lisp.
<cl>
<cl> (load "/usr/misc/.Soar5/lib/taql/load.lisp")
; Loading /usr/misc/.Soar5/lib/taql/load.lisp.

Disabling selected default productions: #*#*#*#*#*#*#*#*#**#*#*#**#*#*
Loading TAQL support productions:
*****************************************************************************************



TAQL 3.1.4
Created July 15, 1991

Bug reports should be sent to Soar-bugs@cs.cmu.edu.
Send comments on TAQL to Gregg.Yost@cs.cmu.edu or Erik.Altmann@cs.cmu.edu.


t
<cl> (load "/afs/cs.cmu.edu/user/altmann/taql/truck/truck.taql")
; Loading /afs/cs.cmu.edu/user/altmann/taql/truck/truck.taql.
***************************************************************************************************
****#(excised DEFAULT*MONITOR*OPERATOR*EVALUATION)
t
<cl> (user-select)  ;; for documentation
first
<cl> (learn on)     ;; turn learning on
Learn status: on all-goals print trace
```

```
on
all-goals
print
trace
<cl> (run)

Learn status: on all-goals print trace

0   G: g1
1   P: p4 (top-space)
2   S: s73
3   O: o69 (do-task)
4   ==>G: g76 (operator no-change)
5      P: p83 (task)
6      S: s104
7      ==>G: g130 (operator tie)
8         P: p131 (selection)
9         S: s138
10        O: o143 (((green) (cannonball) (trip3) simulate-trip) evaluate-object)
11        ==>G: g171 (operator no-change)
12           P: p83 (task)
13           S: d181
14           O: c182 ((green) (cannonball) (trip3) simulate-trip)
15           ==>G: g212 (operator no-change)
16              P: p219 (simulate-trip)
17              S: s230
18              O: o232 ((segment1) simulate-segment)
19              ==>G: g238 (operator no-change)
20                 P: p245 (simulate-segment)
21                 S: s255
22                 O: o257 (compute-resources)
23                 O: o272 (update-resources)
Build:p277
24              O: o278 ((segment2) simulate-segment)
25              ==>G: g280 (operator no-change)
26                 P: p287 (simulate-segment)
27                 S: s297
28                 O: o299 (compute-resources)
29                 O: o314 (update-resources)
Build:p319
Build:p320
30              O: o235 (trip-succeeds)
Build:p323
Build:p324
Build:p325
Build:p326
Build:p327
31           ==>G: g331 (operator tie)
32              P: p332 (selection)
33              S: s339
34              O: o344 (((brown) (traveler) (trip1) simulate-trip) evaluate-object)
35              ==>G: g363 (operator no-change)
36                 P: p83 (task)
37                 S: d373
38                 O: c374 ((brown) (traveler) (trip1) simulate-trip)
39                 ==>G: g396 (operator no-change)
40                    P: p403 (simulate-trip)
41                    S: s414
42                    O: o416 ((segment1) simulate-segment)
43                    ==>G: g422 (operator no-change)
44                       P: p429 (simulate-segment)
45                       S: s439
46                       O: o441 (typewriters 200.0 0.2 1000.0 load)
47                       O: o443 (compute-resources)
48                       O: o464 (update-resources)
Build:p469
49                    O: o470 ((segment2) simulate-segment)
50                    ==>G: g474 (operator no-change)
51                       P: p481 (simulate-segment)
```

```
52                        S: s491
53                        O: o493 (compute-resources)
54                        O: o508 (update-resources)
Build:p513
Build:p514
55                     O: o515 ((segment3) simulate-segment)
56                     ==>G: g519 (operator no-change)
57                        P: p526 (simulate-segment)
58                        S: s536
59                        O: o538 (compute-resources)
60                        O: o553 (update-resources)
Build:p558
Build:p559
61                     O: o560 ((segment4) simulate-segment)
62                     ==>G: g562 (operator no-change)
63                        P: p569 (simulate-segment)
64                        S: s579
65                        O: o581 (typewriters 200.0 0.25 1000.0 unload)
66                        O: o583 (compute-resources)
67                        O: o604 (update-resources)
Build:p609
Build:p610
68                     O: o419 (trip-succeeds)
Build:p613
Build:p614
Build:p615
Build:p616
Build:p617
69                  O: o380 ((gray) (piper) (trip2) simulate-trip)
70                  ==>G: g621 (operator no-change)
71                     P: p628 (simulate-trip)
72                     S: s639
73                     O: o641 ((segment1) simulate-segment)
74                     ==>G: g647 (operator no-change)
75                        P: p654 (simulate-segment)
76                        S: s664
77                        O: o666 (compute-resources)
78                        O: o681 (update-resources)
Build:p686
79                     O: o687 ((segment2) simulate-segment)
80                     ==>G: g691 (operator no-change)
81                        P: p698 (simulate-segment)
82                        S: s708
83                        O: o710 (compute-resources)
84                        O: o725 (update-resources)
Build:p730
Build:p731
85                     O: o732 ((segment3) simulate-segment)
86                     ==>G: g736 (operator no-change)
87                        P: p743 (simulate-segment)
88                        S: s753
89                        O: o755 (compute-resources)
90                        O: o770 (update-resources)
Build:p775
Build:p776
91                     O: o777 ((segment4) simulate-segment)
92                     ==>G: g781 (operator no-change)
93                        P: p788 (simulate-segment)
94                        S: s798
95                        O: o800 (carpet 100.0 0.2 500.0 load)
96                        O: o802 (compute-resources)
97                        O: o823 (update-resources)
Build:p828
Build:p829
98                     O: o830 ((segment5) simulate-segment)
99                     ==>G: g834 (operator no-change)
100                       P: p841 (simulate-segment)
101                       S: s851
102                       O: o853 (carpet 100.0 0.25 500.0 unload)
```

```
103                      O: o855 (compute-resources)
104                      O: o876 (update-resources)
Build:p881
Build:p882
105                   O: o883 ((segment6) simulate-segment)
106                   ==>G: g885 (operator no-change)
107                      P: p892 (simulate-segment)
108                      S: s902
109                      O: o904 (newsprint 400.0 0.2 6000.0 load)
110                      O: o906 (compute-resources)
111                      O: o931 (update-resources)
Build:p936
Build:p937
112                   O: o938 (weight-limit-exceeded local-constraint-failed)
Build:p943
113                O: e365 (failure final evaluate-state)
Build:p945
                   Evaluation of operator c374 (simulate-trip) is failure
Build:p946
115             O: o346 (((gray) (piper) (trip1) simulate-trip) evaluate-object)
116             ==>G: g948 (operator no-change)
117                P: p83 (task)
118                S: d958
Firing 119:915 p946
119                O: c959 ((gray) (piper) (trip1) simulate-trip)
120                ==>G: g981 (operator no-change)
121                   P: p988 (simulate-trip)
122                   S: s999
123                   O: o1001 ((segment1) simulate-segment)
124                   ==>G: g1007 (operator no-change)
125                      P: p1014 (simulate-segment)
126                      S: s1024
127                      O: o1026 (typewriters 200.0 0.2 1000.0 load)
128                      O: o1028 (compute-resources)
129                      O: o1049 (update-resources)
Build:p1054
130                   O: o1055 ((segment2) simulate-segment)
131                   ==>G: g1059 (operator no-change)
132                      P: p1066 (simulate-segment)
133                      S: s1076
134                      O: o1078 (compute-resources)
135                      O: o1093 (update-resources)
Build:p1098
Build:p1099
136                   O: o1100 ((segment3) simulate-segment)
137                   ==>G: g1104 (operator no-change)
138                      P: p1111 (simulate-segment)
139                      S: s1121
140                      O: o1123 (compute-resources)
141                      O: o1138 (update-resources)
Build:p1143
Build:p1144
142                   O: o1145 ((segment4) simulate-segment)
143                   ==>G: g1147 (operator no-change)
144                      P: p1154 (simulate-segment)
145                      S: s1164
146                      O: o1166 (typewriters 200.0 0.25 1000.0 unload)
147                      O: o1168 (compute-resources)
148                      O: o1189 (update-resources)
Build:p1194
Build:p1195
149                   O: o1004 (trip-succeeds)
Build:p1198
Build:p1199
Build:p1200
Build:p1201
Retracting 150:1170 p946
Build:p1202
150                O: o966 ((brown) (traveler) (trip2) simulate-trip)
```

```
151               ==>G: g1206 (operator no-change)
152                  P: p1213 (simulate-trip)
153                  S: s1224
154                  O: o1226 ((segment1) simulate-segment)
155                  ==>G: g1232 (operator no-change)
156                     P: p1239 (simulate-segment)
157                     S: s1249
158                     O: o1251 (compute-resources)
159                     O: o1266 (update-resources)
Build:p1271
160                  O: o1272 ((segment2) simulate-segment)
161                  ==>G: g1276 (operator no-change)
162                     P: p1283 (simulate-segment)
163                     S: s1293
164                     O: o1295 (compute-resources)
165                     O: o1310 (update-resources)
Build:p1315
Build:p1316
166                  O: o1317 ((segment3) simulate-segment)
167                  ==>G: g1321 (operator no-change)
168                     P: p1328 (simulate-segment)
169                     S: s1338
170                     O: o1340 (compute-resources)
171                     O: o1355 (update-resources)
Build:p1360
Build:p1361
172                  O: o1362 ((segment4) simulate-segment)
173                  ==>G: g1366 (operator no-change)
174                     P: p1373 (simulate-segment)
175                     S: s1383
176                     O: o1385 (carpet 100.0 0.2 500.0 load)
177                     O: o1387 (compute-resources)
178                     O: o1408 (update-resources)
Build:p1413
Build:p1414
179                  O: o1415 ((segment5) simulate-segment)
180                  ==>G: g1419 (operator no-change)
181                     P: p1426 (simulate-segment)
182                     S: s1436
183                     O: o1438 (carpet 100.0 0.25 500.0 unload)
184                     O: o1440 (compute-resources)
185                     O: o1461 (update-resources)
Build:p1466
Build:p1467
186                  O: o1468 ((segment6) simulate-segment)
187                  ==>G: g1470 (operator no-change)
188                     P: p1477 (simulate-segment)
189                     S: s1487
190                     O: o1489 (newsprint 400.0 0.2 6000.0 load)
191                     O: o1491 (compute-resources)
192                     O: o1512 (update-resources)
Build:p1517
Build:p1518
193                  O: o1229 (trip-succeeds)
Build:p1521
Build:p1522
Build:p1523
Build:p1524
Build:p1525
194               O: o1526 (no-trips-left schedule-succeeds)
195               O: e950 (success final evaluate-state)
Build:p1534
                  Evaluation of operator c959 (simulate-trip) is success
Build:p1535
Build:p1536
Build:p1537
                  Evaluation of operator c182 (simulate-trip) is partial-success
Build:p1538
196    O: o106 ((green) (cannonball) (trip3) simulate-trip)
```

```
Firing 197:1573 p323
Firing 197:1573 p327
Firing 197:1573 p324
Firing 197:1573 p325
Firing 197:1573 p326
Firing 197:1575 p946
Firing 197:1575 p1535
197    O: o115 ((gray) (piper) (trip1) simulate-trip)
Firing 198:1580 p1198
Firing 198:1580 p1202
Firing 198:1580 p1201
Firing 198:1580 p1199
Firing 198:1580 p1200
Retracting 198:1582 p1535
Retracting 198:1582 p946
198    O: o112 ((brown) (traveler) (trip2) simulate-trip)
Firing 199:1587 p1521
Firing 199:1587 p1525
Firing 199:1587 p1522
Firing 199:1587 p1523
Firing 199:1587 p1524
199    O: o1542 (no-trips-left schedule-succeeds)
200    O: o102 (final-state)
       Space task succeeded in goal g76.
Build:p1548
Build:p1549
201 O: o6 (halt)
Applied task operator o69 (do-task).  Final state is s73.
End -- Explicit Halt
nil
<cl> (list-chunks "/afs/cs.cmu.edu/user/altmann/taql/truck/chunks.soar")
Copying chunks to file /afs/cs.cmu.edu/user/altmann/taql/truck/chunks.soar.

t
<cl> (print-stats)
Soar (Version 5, Release 2)
Created August 26th, 1991

Run statistics on June  3, 1992
Allegro CL 3.1.12.2 [DECstation] (11/19/90) DECstation  id: 385 Ultrix  TRICERATOPS.SOAR.CS.CMU.EDU

362 productions (6491 / 27250 nodes)
 69 chunks (69 / 362 productions)
109.383 seconds elapsed 25.353 seconds chunking overhead
202 decision cycles (541.50006 ms per cycle)
707 elaboration cycles (154.71428 ms per cycle)
    (3.5 e cycles/d cycle)
2040 production firings (53.61912 ms per firing)
     2.8854313 productions in parallel
9415 RHS actions after initialization (11.61795 ms per action)
392 mean working memory size (1041 maximum, 350 current)
3347 mean token memory size (9279 maximum, 1802 current)
21814 left tokens added, 19337 right tokens added, 41151 total tokens added
21369 left tokens removed, 17980 right tokens removed, 39349 total tokens removed
80500 token changes (1.358795 ms per change)
     (8.549278 changes/action)
nil
<cl> (taql-stats)
TAQL 3.1.4
Created July 15, 1991

TAQL statistics on June  3, 1992
Allegro CL 3.1.12.2 [DECstation] (11/19/90) DECstation  id: 385 Ultrix  TRICERATOPS.SOAR.CS.CMU.EDU

47 TCs (46 user, 1 default)
    compiled into 124 productions (101 user, 23 default)
t
<cl> ;; end of trace
```

4.4. Chunk Listing

```
;;;; -*- Mode: Soar -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File            : /afs/cs.cmu.edu/user/altmann/taql/truck/chunks.soar
;;;; Author          : Erik Altmann
;;;; Created On      : Wed Jun  3 20:26:56 1992
;;;; Last Modified By: Erik Altmann
;;;; Last Modified On: Wed Jun  3 20:27:22 1992
;;;; Update Count    : 1
;;;; Soar Version    : 5.2.1
;;;;
;;;; PURPOSE
;;;;    Chunks generated by a run of the sample implementation of the
;;;; Trucking Task,  aka Shipping Scheduling Assistant.  The source
;;;; code is in truck.taql.  The trace from which these chunks were
;;;; generated is in trace.txt.
;;;;
;;;; Copyright 1992, Carnegie Mellon University.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(sp p277
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> -^current-resources ^dummy-att* true)
   (operator <o1> ^name simulate-segment ^segment <s1> ^resources <l1>)
   (segment <s1> ^name segment1 ^trip trip3)
   (list <l1> ^car <r1>)
   (resources <r1> ^current-volume 1280 ^current-time 11
                   ^current-weight 32000)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr nil + ^type* list +)
   (resources <r2> ^segment segment1 + segment1 & ^trip trip3 + trip3 &
                   ^current-weight 32000 + 32000 &
                   ^current-volume 1280 + 1280 & ^current-time 11 + 11 &))

(sp p319
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 1280 ^current-time 11
                   ^current-weight 32000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p320
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 1280 ^current-time 11
                   ^current-weight 32000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2 ^trip trip3)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment2 + segment2 & ^trip trip3 + trip3 &
                   ^current-weight 32000 + 32000 &
                   ^current-volume 1280 + 1280 & ^current-time 11 + 11 &))

(sp p323
   (goal <g1> ^state <d4> ^operator <c1>)
   (state <d4> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
 -->
   (state <d4> ^current-schedule <l1> -))

(sp p324
   (goal <g1> ^state <d4> ^operator <c1>)
   (state <d4> ^dummy-att* true ^truck <t2> ^current-schedule <l1>)
   (truck <t2> ^volume 1280 ^weight-limit 32000)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (operator <c1> ^name simulate-trip ^truck <t2> ^driver <d1> ^trip <t1>)
   (driver <d1> ^drive-time 11)
 -->
   (state <d4> ^truck <t2> -))

(sp p325
   (goal <g1> ^state <d4> ^operator <c1>)
   (state <d4> ^dummy-att* true ^driver <d3> ^current-schedule <l1>)
   (driver <d3> ^drive-time 11)
   (list <l1> ^car <d2>)
   (operator <d2> ^trip <d1>)
   (trip <d1> ^name dummy)
   (operator <c1> ^name simulate-trip ^driver <d3> ^truck <t1> ^trip <t2>)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
 -->
   (state <d4> ^driver <d3> -))

(sp p326
   (goal <g1> ^state <d4> ^operator <c1>)
   (state <d4> ^dummy-att* true ^current-schedule <l1> ^trip <t2>)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (operator <c1> ^name simulate-trip ^trip <t2> ^driver <d1> ^truck <t1>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
 -->
   (state <d4> ^trip <t2> -))

(sp p327
   (goal <g1> ^state <d4> ^operator <c1>)
   (state <d4> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
 -->
   (state <d4> ^current-schedule <l2> &, <l2> +)
   (list <l2> ^type* list + ^cdr <l1> + ^car <c1> +))

(sp p469
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> -^current-resources ^dummy-att* true)
   (operator <o1> ^name simulate-segment ^segment <s3> ^resources <l1>)
   (segment <s3> ^load-shipment typewriters -^load-shipment NIL
              ^name segment1 ^trip trip1)
   (list <l1> ^car <r1>)
   (resources <r1> ^current-volume 640 ^current-time 11
                   ^current-weight 10000)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
               ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr nil + ^type* list +)
   (resources <r2> ^segment segment1 + segment1 & ^trip trip1 + trip1 &
                   ^current-weight 9000.0 + 9000.0 &
                   ^current-volume 440.0 + 440.0 &
                   ^current-time 10.8 + 10.8 &))

(sp p513
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p514
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2 ^trip trip1)
 -->
```

```
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment2 + segment2 & ^trip trip1 + trip1 &
                   ^current-weight 9000.0 + 9000.0 &
                   ^current-volume 440.0 + 440.0 &
                   ^current-time 10.8 + 10.8 &))

(sp p558
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p559
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3 ^trip trip1)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment3 + segment3 & ^trip trip1 + trip1 &
                   ^current-weight 9000.0 + 9000.0 &
                   ^current-volume 440.0 + 440.0 &
                   ^current-time 10.8 + 10.8 &))

(sp p609
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment typewriters -^unload-shipment NIL
                 ^name segment4)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
                  ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p610
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 440.0
                   ^current-time 10.8 ^current-weight 9000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment typewriters -^unload-shipment NIL
                 ^name segment4 ^trip trip1)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
                  ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment4 + segment4 & ^trip trip1 + trip1 &
                   ^current-weight 10000.0 + 10000.0 &
                   ^current-volume 640.0 + 640.0 &
                   ^current-time 10.55 + 10.55 &))

(sp p613
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^current-schedule <l1> -))

(sp p614
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^truck <t3> ^current-schedule <l1>)
   (truck <t3> ^volume 640 ^weight-limit 10000)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <c1> ^name simulate-trip ^truck <t3> ^driver <d1> ^trip <t1>)
   (driver <d1> ^drive-time 11)
 -->
   (state <d2> ^truck <t3> -))

(sp p615
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^driver <d1> ^current-schedule <l1>)
   (driver <d1> ^drive-time 11)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^driver <d1> -))

(sp p616
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1> ^trip <t3>)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <c1> ^name simulate-trip ^trip <t3> ^driver <d1> ^truck <t1>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^trip <t3> -))

(sp p617
   (goal <g1> ^state <d2> ^operator <c2>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c2> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^current-schedule <l2> &, <l2> +)
   (list <l2> ^type* list + ^cdr <l1> + ^car <c2> +))

(sp p686
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> -^current-resources ^dummy-att* true)
   (operator <o1> ^name simulate-segment ^segment <s1> ^resources <l1>)
   (segment <s1> ^name segment1 ^trip trip2)
   (list <l1> ^car <r1>)
   (resources <r1> ^current-volume 400 ^current-time 12.5
                   ^current-weight 5000)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr nil + ^type* list +)
   (resources <r2> ^segment segment1 + segment1 & ^trip trip2 + trip2 &
                   ^current-weight 5000 + 5000 & ^current-volume 400 + 400 &
                   ^current-time 12.5 + 12.5 &))

(sp p730
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 400 ^current-time 12.5
                   ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p731
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 400 ^current-time 12.5
                   ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2 ^trip trip2)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment2 + segment2 & ^trip trip2 + trip2 &
```

```
                  ^current-weight 5000 + 5000 & ^current-volume 400 + 400 &
                  ^current-time 12.5 + 12.5 &))

(sp p775
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 400 ^current-time 12.5
                  ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p776
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 400 ^current-time 12.5
                  ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3 ^trip trip2)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment3 + segment3 & ^trip trip2 + trip2 &
                  ^current-weight 5000 + 5000 & ^current-volume 400 + 400 &
                  ^current-time 12.5 + 12.5 &))

(sp p828
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 400 ^current-time 12.5
                  ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment carpet -^load-shipment NIL ^name segment4)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p829
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 400 ^current-time 12.5
                  ^current-weight 5000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment carpet -^load-shipment NIL ^name segment4
                ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment4 + segment4 & ^trip trip2 + trip2 &
                  ^current-weight 4500.0 + 4500.0 &
                  ^current-volume 300.0 + 300.0 &
                  ^current-time 12.3 + 12.3 &))

(sp p881
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment5 ^current-volume 300.0
                  ^current-time 12.3 ^current-weight 4500.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment carpet -^unload-shipment NIL
                ^name segment5)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0
                 ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p882
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment5 ^current-volume 300.0
                  ^current-time 12.3 ^current-weight 4500.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment carpet -^unload-shipment NIL
                ^name segment5 ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0
                 ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment5 + segment5 & ^trip trip2 + trip2 &
                  ^current-weight 5000.0 + 5000.0 &
                  ^current-volume 400.0 + 400.0 &
                  ^current-time 12.05 + 12.05 &))

(sp p936
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment6 ^current-volume 400.0
                  ^current-time 12.05 ^current-weight 5000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment newsprint -^load-shipment NIL
                ^name segment6)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name newsprint ^weight 6000.0 ^volume 400.0
                 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p937
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment6 ^current-volume 400.0
                  ^current-time 12.05 ^current-weight 5000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name newsprint ^weight 6000.0 ^volume 400.0
                 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment6 + segment6 & ^trip trip2 + trip2 &
                  ^current-weight -1000.0 + -1000.0 &
                  ^current-volume 0.0 + 0.0 & ^current-time 11.85 + 11.85 &
                  ^failed true + true &
                  ^reason weight-limit-exceeded + weight-limit-exceeded &))

(sp p943
   (goal <g2> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip1)
   (operator <o1> ^name simulate-trip ^driver <d1> ^truck <t2> ^trip <t1>)
   (driver <d1> ^drive-time 12.5)
   (truck <t2> ^volume 400 ^weight-limit 5000)
   (trip <t1> ^first-segment <s1>)
   (segment <s1> ^name segment1 ^trip trip2 ^next-segment <s2>)
   (segment <s2> ^name segment2 ^trip trip2 ^next-segment <s3>)
   (segment <s3> ^name segment3 ^trip trip2 ^next-segment <s4>)
   (segment <s4> ^load-shipment carpet -^load-shipment NIL ^name segment4
                ^trip trip2 ^next-segment <s6>)
   (segment <s6> ^unload-shipment carpet -^unload-shipment NIL
                ^name segment5 ^trip trip2 ^next-segment <s9>)
   (segment <s9> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                ^trip trip2)
   (goal <g1> ^object NIL ^state <s8>)
   (state <s8> ^shipment <s7> <s5>)
   (shipment <s7> ^name newsprint ^weight 6000.0 ^volume 400.0
                 ^load-time 0.2)
   (shipment <s5> ^name carpet ^weight 500.0 ^volume 100.0 ^unload-time 0.25
                 ^load-time 0.2)
 -->
   (state <d2> ^local-constraint-failed <l2> &, <l2> +)
   (list <l2> ^type* list + ^cdr <l1> + ^car <o1> +))

(sp p945
   (goal <g3> ^problem-space <p1> ^state <s10> ^object <g1> ^operator <o1>)
   (problem-space <p1> ^name selection)
```

```
   (goal <g1> ^problem-space <p2>)
   (problem-space <p2> -^dont-copy-anything
                       -^dont-copy current-schedule trip truck driver
                                   dummy-att*
                       -^two-level-attributes current-schedule trip truck
                                              driver dummy-att*
                       -^all-attributes-at-level two -^one-level-attributes
                       ^name task)
   (state <s10> ^evaluation <e1>)
   (evaluation <e1> ^object <o2>)
   (operator <o2> -^dont-copy trip name truck driver ^name simulate-trip
                  ^trip <t5> ^truck <t1> ^driver <d1>)
   (trip <t5> ^name trip1)
   (truck <t1> ^volume 640 ^weight-limit 10000)
   (driver <d1> ^drive-time 11)
   (operator <o1> ^type evaluation ^attribute operator
                  ^default-state-copy yes ^default-operator-copy yes
                  ^object <o2> ^superproblem-space <p2> ^superstate <d3>
                  ^desired <d4>)
   (state <d3> ^dummy-att* true ^driver <d2> ^truck <t4> ^trip <t3>
               ^current-schedule <l1>)
   (driver <d2> ^drive-time 12.5 ^name gray)
   (truck <t4> ^volume 400 ^weight-limit 5000 ^type small ^name piper)
   (trip <t3> ^name trip2 ^first-segment <s1>)
   (segment <s1> ^name segment1 ^trip trip2 ^source gary ^next-segment <s2>)
   (segment <s2> ^name segment2 ^trip trip2 ^next-segment <s3>)
   (segment <s3> ^name segment3 ^trip trip2 ^next-segment <s7>)
   (segment <s7> ^load-shipment carpet -^load-shipment NIL ^name segment4
                 ^trip trip2 ^next-segment <s8>)
   (segment <s8> ^unload-shipment carpet -^unload-shipment NIL
                 ^name segment5 ^trip trip2 ^next-segment <s9>)
   (segment <s9> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                 ^trip trip2)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)
   (goal <g2> ^object NIL ^state <s6>)
   (state <s6> ^shipment <s5> <s4> ^license <l2> ^city <c2>)
   (shipment <s5> ^name newsprint ^weight 6000.0 ^volume 400.0
                  ^load-time 0.2)
   (shipment <s4> ^name carpet ^weight 500.0 ^volume 100.0 ^unload-time 0.25
                  ^load-time 0.2)
   (license <l2> ^truck-type small ^holder gray)
   (city <c2> ^name gary ^truck piper ^driver gray)
 -->
   (evaluation <e1> ^symbolic-value failure +))

(sp p946
   (goal <g2> ^desired <d4> ^problem-space <p1> ^state <d2>
              ^operator <o1> +)
   (problem-space <p1> -^two-level-attributes dummy-att* driver truck trip
                                              current-schedule
                       -^dont-copy dummy-att* driver truck trip
                                   current-schedule -^one-level-attributes
                       -^all-attributes-at-level two -^dont-copy-anything
                       -^default-operator-copy no -^default-state-copy no
                       ^name task)
   (state <d2> ^dummy-att* true ^driver <d3> ^truck <t4> ^trip <t3>
               ^current-schedule <l1>)
   (driver <d3> ^drive-time 12.5 ^name gray)
   (truck <t4> ^volume 400 ^weight-limit 5000 ^type small ^name piper)
   (trip <t3> ^name trip2 ^first-segment <s1>)
   (segment <s1> ^name segment1 ^trip trip2 ^source gary ^next-segment <s2>)
   (segment <s2> ^name segment2 ^trip trip2 ^next-segment <s3>)
   (segment <s3> ^name segment3 ^trip trip2 ^next-segment <s7>)
   (segment <s7> ^load-shipment carpet -^load-shipment NIL ^name segment4
                 ^trip trip2 ^next-segment <s8>)
   (segment <s8> ^unload-shipment carpet -^unload-shipment NIL
                 ^name segment5 ^trip trip2 ^next-segment <s9>)
   (segment <s9> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                 ^trip trip2)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <o1> -^dont-copy trip name truck driver ^name simulate-trip
                  ^trip <t5> ^truck <t1> ^driver <d1>)
   (trip <t5> ^name trip1)
   (truck <t1> ^volume 640 ^weight-limit 10000)
   (driver <d1> ^drive-time 11)
   (goal <g1> ^object NIL ^state <s6>)
   (state <s6> ^shipment <s5> <s4> ^license <l2> ^city <c2>)
   (shipment <s5> ^name newsprint ^weight 6000.0 ^volume 400.0
                  ^load-time 0.2)
   (shipment <s4> ^name carpet ^weight 500.0 ^volume 100.0 ^unload-time 0.25
                  ^load-time 0.2)
   (license <l2> ^truck-type small ^holder gray)
   (city <c2> ^name gary ^truck piper ^driver gray)
 -->
   (goal <g2> ^operator <o1> -))

(sp p1054
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> -^current-resources ^dummy-att* true)
   (operator <o1> ^name simulate-segment ^segment <s3> ^resources <l1>)
   (segment <s3> ^load-shipment typewriters -^load-shipment NIL
                 ^name segment1 ^trip trip1)
   (list <l1> ^car <r1>)
   (resources <r1> ^current-volume 400 ^current-time 12.5
                   ^current-weight 5000)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
                 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr nil + ^type* list +)
   (resources <r2> ^segment segment1 + segment1 & ^trip trip1 + trip1 &
                   ^current-weight 4000.0 + 4000.0 &
                   ^current-volume 200.0 + 200.0 &
                   ^current-time 12.3 + 12.3 &))


(sp p1098
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 200.0
                   ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p1099
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 200.0
                   ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2 ^trip trip1)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment2 + segment2 & ^trip trip1 + trip1 &
                   ^current-weight 4000.0 + 4000.0 &
                   ^current-volume 200.0 + 200.0 &
                   ^current-time 12.3 + 12.3 &))


(sp p1143
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 200.0
                   ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p1144
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 200.0
                   ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3 ^trip trip1)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment3 + segment3 & ^trip trip1 + trip1 &
                   ^current-weight 4000.0 + 4000.0 &
                   ^current-volume 200.0 + 200.0 &
                   ^current-time 12.3 + 12.3 &))

(sp p1194
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 200.0
```

```
                 ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment typewriters -^unload-shipment NIL
               ^name segment4)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
                ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p1195
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 200.0
                 ^current-time 12.3 ^current-weight 4000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment typewriters -^unload-shipment NIL
               ^name segment4 ^trip trip1)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name typewriters ^weight 1000.0 ^volume 200.0
                ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment4 + segment4 & ^trip trip1 + trip1 &
                 ^current-weight 5000.0 + 5000.0 &
                 ^current-volume 400.0 + 400.0 &
                 ^current-time 12.05 + 12.05 &))

(sp p1198
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 12.5)
   (truck <t1> ^volume 400 ^weight-limit 5000)
 -->
   (state <d2> ^current-schedule <l1> -))

(sp p1199
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^truck <t3> ^current-schedule <l1>)
   (truck <t3> ^volume 400 ^weight-limit 5000)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <c1> ^name simulate-trip ^truck <t3> ^driver <d1> ^trip <t1>)
   (driver <d1> ^drive-time 12.5)
 -->
   (state <d2> ^truck <t3> -))

(sp p1200
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^driver <d1> ^current-schedule <l1>)
   (driver <d1> ^drive-time 12.5)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (truck <t1> ^volume 400 ^weight-limit 5000)
 -->
   (state <d2> ^driver <d1> -))

(sp p1201
   (goal <g1> ^state <d2> ^operator <c1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1> ^trip <t3>)
   (list <l1> ^car <c2>)
   (operator <c2> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <c1> ^name simulate-trip ^trip <t3> ^driver <d1> ^truck <t1>)
   (driver <d1> ^drive-time 12.5)
   (truck <t1> ^volume 400 ^weight-limit 5000)
 -->
   (state <d2> ^trip <t3> -))

(sp p1202
   (goal <g1> ^state <d2> ^operator <c2>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip3)
   (operator <c2> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 12.5)
   (truck <t1> ^volume 400 ^weight-limit 5000)
 -->
   (state <d2> ^current-schedule <l2> &, <l2> +)
   (list <l2> ^type* list + ^cdr <l1> + ^car <c2> +))

(sp p1271
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> -^current-resources ^dummy-att* true)
   (operator <o1> ^name simulate-segment ^segment <s1> ^resources <l1>)
   (segment <s1> ^name segment1 ^trip trip2)
   (list <l1> ^car <r1>)
   (resources <r1> ^current-volume 640 ^current-time 11
                ^current-weight 10000)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr nil + ^type* list +)
   (resources <r2> ^segment segment1 + segment1 & ^trip trip2 + trip2 &
                ^current-weight 10000 + 10000 &
                ^current-volume 640 + 640 & ^current-time 11 + 11 &))

(sp p1315
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 640 ^current-time 11
                ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p1316
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment2 ^current-volume 640 ^current-time 11
                ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment2 ^trip trip2)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment2 + segment2 & ^trip trip2 + trip2 &
                ^current-weight 10000 + 10000 &
                ^current-volume 640 + 640 & ^current-time 11 + 11 &))

(sp p1360
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 640 ^current-time 11
                ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3)
 -->
   (state <s2> ^current-resources <l1> -))

(sp p1361
   (goal <g1> ^state <s2> ^operator <o1>)
   (state <s2> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment3 ^current-volume 640 ^current-time 11
                ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s1>)
   (segment <s1> ^name segment3 ^trip trip2)
 -->
   (state <s2> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment3 + segment3 & ^trip trip2 + trip2 &
                ^current-weight 10000 + 10000 &
                ^current-volume 640 + 640 & ^current-time 11 + 11 &))

(sp p1413
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 640 ^current-time 11
                ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment carpet -^load-shipment NIL ^name segment4)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
```

```
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p1414
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment4 ^current-volume 640 ^current-time 11
                   ^current-weight 10000)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment carpet -^load-shipment NIL ^name segment4
                 ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0 ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment4 + segment4 & ^trip trip2 + trip2 &
                   ^current-weight 9500.0 + 9500.0 &
                   ^current-volume 540.0 + 540.0 &
                   ^current-time 10.8 + 10.8 &))

(sp p1466
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment5 ^current-volume 540.0
                   ^current-time 10.8 ^current-weight 9500.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment carpet -^unload-shipment NIL
                 ^name segment5)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0
                  ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p1467
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment5 ^current-volume 540.0
                   ^current-time 10.8 ^current-weight 9500.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^unload-shipment carpet -^unload-shipment NIL
                 ^name segment5 ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name carpet ^weight 500.0 ^volume 100.0
                  ^unload-time 0.25)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment5 + segment5 & ^trip trip2 + trip2 &
                   ^current-weight 10000.0 + 10000.0 &
                   ^current-volume 640.0 + 640.0 &
                   ^current-time 10.55 + 10.55 &))

(sp p1517
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment6 ^current-volume 640.0
                   ^current-time 10.55 ^current-weight 10000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment newsprint -^load-shipment NIL
                 ^name segment6)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name newsprint ^weight 6000.0 ^volume 400.0
                  ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l1> -))

(sp p1518
   (goal <g2> ^state <s4> ^operator <o1>)
   (state <s4> ^dummy-att* true ^current-resources <l1>)
   (list <l1> ^car <r1>)
   (resources <r1> -^segment segment6 ^current-volume 640.0
                   ^current-time 10.55 ^current-weight 10000.0)
   (operator <o1> ^name simulate-segment ^resources <l1> ^segment <s3>)
   (segment <s3> ^load-shipment newsprint -^load-shipment NIL ^name segment6
                 ^trip trip2)
   (goal <g1> ^object NIL ^state <s2>)
   (state <s2> ^shipment <s1>)
   (shipment <s1> ^name newsprint ^weight 6000.0 ^volume 400.0
                  ^load-time 0.2)
 -->
   (state <s4> ^current-resources <l2> &, <l2> +)
   (list <l2> ^car <r2> + ^cdr <l1> + ^type* list +)
   (resources <r2> ^segment segment6 + segment6 & ^trip trip2 + trip2 &
                   ^current-weight 4000.0 + 4000.0 &
                   ^current-volume 240.0 + 240.0 &
                   ^current-time 10.35 + 10.35 &))

(sp p1521
   (goal <g1> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip1)
   (operator <o1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^current-schedule <l1> -))

(sp p1522
   (goal <g1> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^truck <t3> ^current-schedule <l1>)
   (truck <t3> ^volume 640 ^weight-limit 10000)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip1)
   (operator <o1> ^name simulate-trip ^truck <t3> ^driver <d1> ^trip <t1>)
   (driver <d1> ^drive-time 11)
 -->
   (state <d2> ^truck <t3> -))

(sp p1523
   (goal <g1> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^driver <d1> ^current-schedule <l1>)
   (driver <d1> ^drive-time 11)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip1)
   (operator <o1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^driver <d1> -))

(sp p1524
   (goal <g1> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1> ^trip <t3>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip1)
   (operator <o1> ^name simulate-trip ^trip <t3> ^driver <d1> ^truck <t1>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^trip <t3> -))

(sp p1525
   (goal <g1> ^state <d2> ^operator <o1>)
   (state <d2> ^dummy-att* true ^current-schedule <l1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t3>)
   (trip <t3> ^name trip1)
   (operator <o1> ^name simulate-trip ^driver <d1> ^truck <t1> ^trip <t2>)
   (driver <d1> ^drive-time 11)
   (truck <t1> ^volume 640 ^weight-limit 10000)
 -->
   (state <d2> ^current-schedule <l2> &, <l2> +)
   (list <l2> ^type* list + ^cdr <l1> + ^car <o1> +))

(sp p1534
   (goal <g3> ^problem-space <p1> ^state <s3> ^object <g2> ^operator <o1>)
   (problem-space <p1> ^name selection)
   (goal <g2> ^problem-space <p2>)
   (problem-space <p2> -^dont-copy-anything
                       -^dont-copy current-schedule dummy-att* trip truck
                                   driver
                       -^two-level-attributes current-schedule dummy-att*
                                              trip truck driver
                       -^all-attributes-at-level two -^one-level-attributes
                       ^name task -^name selection)
```

```
   (state <s3> ^evaluation <e1>)
   (evaluation <e1> ^object <o2>)
   (operator <o2> -^dont-copy trip name truck driver ^name simulate-trip
                ^trip <t3> ^truck <t1> ^driver <d1>)
   (trip <t3> ^name trip1)
   (truck <t1> ^volume 400 ^weight-limit 5000)
   (driver <d1> ^drive-time 12.5)
   (operator <o1> ^type evaluation ^attribute operator
                ^default-state-copy yes ^default-operator-copy yes
                ^object <o2> ^superproblem-space <p2> ^superstate <d3>
                ^desired <d4>)
   (state <d3> ^dummy-att* true ^driver <d2> ^truck <t4> ^trip <t5>
             ^current-schedule <l1>)
   (driver <d2> ^name brown ^drive-time 11)
   (truck <t4> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (trip <t5> ^name trip2 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)
   (goal <g1> ^object NIL ^state <s1>)
   (state <s1> ^license <l2> ^city <c2>)
   (license <l2> ^truck-type medium ^holder brown)
   (city <c2> ^name gary ^truck traveler ^driver brown)
 -->
   (evaluation <e1> ^symbolic-value success +))

(sp p1535
   (goal <g2> ^desired <d4> ^problem-space <p1> ^state <d2>
            ^operator <o1> +)
   (problem-space <p1> ^name task -^name selection
                    -^two-level-attributes driver truck trip dummy-att*
                                           current-schedule
                    -^dont-copy driver truck trip dummy-att*
                               current-schedule -^one-level-attributes
                    -^all-attributes-at-level two -^dont-copy-anything
                    -^default-operator-copy no -^default-state-copy no)
   (state <d2> ^dummy-att* true ^driver <d3> ^truck <t4> ^trip <t5>
             ^current-schedule <l1>)
   (driver <d3> ^name brown ^drive-time 11)
   (truck <t4> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (trip <t5> ^name trip2 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <o1> -^dont-copy trip name truck driver ^name simulate-trip
                ^trip <t3> ^truck <t1> ^driver <d1>)
   (trip <t3> ^name trip1)
   (truck <t1> ^volume 400 ^weight-limit 5000)
   (driver <d1> ^drive-time 12.5)
   (goal <g1> ^object NIL ^state <s1>)
   (state <s1> ^license <l2> ^city <c2>)
   (license <l2> ^truck-type medium ^holder brown)
   (city <c2> ^name gary ^truck traveler ^driver brown)
 -->
   (goal <g2> ^operator <o1> >))

(sp p1536
   (goal <g3> ^object <g2> ^problem-space <p1> ^desired <d4> ^state <d2>
            ^operator <o1> +)
   (goal <g2> ^state <s3> ^operator <o2>)
   (problem-space <p1> -^default-state-copy no -^default-operator-copy no
                    -^dont-copy-anything
                    -^dont-copy current-schedule dummy-att* trip truck
                               driver
                    -^two-level-attributes current-schedule dummy-att*
                                           trip truck driver
                    -^all-attributes-at-level two -^one-level-attributes
                    ^name task -^name selection)
   (state <d2> ^dummy-att* true ^driver <d3> ^truck <t4> ^trip <t5>
             ^current-schedule <l1>)
   (driver <d3> ^name brown ^drive-time 11)
   (truck <t4> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (trip <t5> ^name trip2 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (state <s3> ^evaluation <e1>)
   (list <l1> ^car <c1>)
   (operator <c1> ^trip <t2>)
   (trip <t2> ^name trip3)
   (operator <o2> ^name evaluate-object ^desired <d4> ^evaluation <e1>)
   (operator <o1> -^dont-copy driver truck name trip ^name simulate-trip
                ^trip <t3> ^truck <t1> ^driver <d1>)
   (trip <t3> ^name trip1)
   (truck <t1> ^volume 400 ^weight-limit 5000)
```

```
   (driver <d1> ^drive-time 12.5)
   (goal <g1> ^object NIL ^state <s1>)
   (state <s1> ^license <l2> ^city <c2>)
   (license <l2> ^truck-type medium ^holder brown)
   (city <c2> ^name gary ^truck traveler ^driver brown)
 -->
   (evaluation <e1> ^symbolic-value partial-success +))

(sp p1537
   (goal <g3> ^state <s5> ^object <g1> ^operator <o2>)
   (goal <g1> ^problem-space <p1>)
   (problem-space <p1> -^all-attributes-at-level two -^one-level-attributes
                    -^two-level-attributes current-schedule trip truck
                                           driver dummy-att*
                    -^dont-copy current-schedule trip truck driver
                               dummy-att* -^dont-copy-anything
                    -^default-state-copy no -^default-operator-copy no
                    ^name task -^name selection)
   (state <s5> ^evaluation <e1>)
   (operator <o2> ^name evaluate-object ^type evaluation ^attribute operator
                ^default-state-copy yes ^default-operator-copy yes
                ^evaluation <e1> ^superproblem-space <p1> ^superstate <s1>
                ^object <o1> ^desired <d6>)
   (state <s1> ^dummy-att* true ^trip <t6> <t4> ^truck <t5> <t2>
             ^driver <d5> <d4> ^current-schedule <l1>)
   (trip <t6> ^name trip2 ^first-segment <s4>)
   (segment <s4> ^source gary)
   (truck <t5> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (driver <d5> ^name brown ^drive-time 11)
   (driver <d4> ^drive-time 12.5 ^name gray)
   (truck <t2> ^volume 400 ^weight-limit 5000 ^type small ^name piper)
   (trip <t4> ^name trip1 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (operator <o1> -^dont-copy trip name truck driver ^name simulate-trip
                ^trip <t3> ^truck <t1> ^driver <d1>)
   (trip <t3> ^name trip3)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
   (driver <d1> ^drive-time 11)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (goal <g2> ^object NIL ^state <s3>)
   (state <s3> ^license <l3> <l2> ^city <c1>)
   (license <l3> ^truck-type medium ^holder brown)
   (city <c1> ^name gary ^truck traveler piper ^driver brown gray)
   (license <l2> ^truck-type small ^holder gray)
 -->
   (evaluation <e1> ^symbolic-value partial-success +))

(sp p1538
   (goal <g2> ^desired <d6> ^problem-space <p1> ^state <s1>
            ^operator <o1> +)
   (problem-space <p1> ^name task -^name selection
                    -^two-level-attributes dummy-att* driver truck trip
                                           current-schedule
                    -^dont-copy dummy-att* driver truck trip
                               current-schedule -^dont-copy-anything
                    -^one-level-attributes -^all-attributes-at-level two
                    -^default-operator-copy no -^default-state-copy no)
   (state <s1> ^dummy-att* true ^trip <t6> <t4> ^truck <t5> <t2>
             ^driver <d5> <d4> ^current-schedule <l1>)
   (trip <t6> ^name trip2 ^first-segment <s4>)
   (segment <s4> ^source gary)
   (truck <t5> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (driver <d5> ^name brown ^drive-time 11)
   (driver <d4> ^drive-time 12.5 ^name gray)
   (truck <t2> ^volume 400 ^weight-limit 5000 ^type small ^name piper)
   (trip <t4> ^name trip1 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (list <l1> ^car <d3>)
   (operator <d3> ^trip <d2>)
   (trip <d2> ^name dummy)
   (operator <o1> -^dont-copy trip name truck driver ^name simulate-trip
                ^trip <t3> ^truck <t1> ^driver <d1>)
   (trip <t3> ^name trip3)
   (truck <t1> ^volume 1280 ^weight-limit 32000)
   (driver <d1> ^drive-time 11)
   (goal <g1> ^object NIL ^state <s3>)
   (state <s3> ^license <l3> <l2> ^city <c1>)
   (license <l3> ^truck-type medium ^holder brown)
   (city <c1> ^name gary ^truck traveler piper ^driver brown gray)
   (license <l2> ^truck-type small ^holder gray)
 -->
   (goal <g2> ^operator <o1> >))
```

```
(sp p1548
   (goal <g1> ^object NIL ^state <s4> ^problem-space <p1> ^operator <o1>)
   (state <s4> ^dummy-att* true ^license <l3> <l2> <l1> ^city <c2> <c1>
               ^driver <d2> <d1> <d3> ^truck <t4> <t3> <t5>
               ^trip <t6> <t2> <t1>)
   (problem-space <p1> ^name top-space)
   (license <l3> ^truck-type small ^holder gray)
   (city <c2> ^name gary ^truck piper traveler ^driver gray brown)
   (license <l2> ^truck-type big ^holder green)
   (city <c1> ^name indy ^truck cannonball ^driver green)
   (license <l1> ^truck-type medium ^holder brown)
   (driver <d2> ^name gray ^drive-time 12.5)
   (truck <t4> ^type small ^name piper ^volume 400 ^weight-limit 5000)
   (trip <t6> ^name trip1 ^first-segment <s3>)
   (segment <s3> ^source gary)
   (driver <d1> ^drive-time 11 ^name green)
   (truck <t3> ^volume 1280 ^weight-limit 32000 ^type big ^name cannonball)
   (trip <t2> ^name trip3 ^first-segment <s2>)
   (segment <s2> ^source indy)
   (driver <d3> ^drive-time 11 ^name brown)
   (truck <t5> ^volume 640 ^weight-limit 10000 ^type medium ^name traveler)
   (trip <t1> ^name trip2 ^first-segment <s1>)
   (segment <s1> ^source gary)
   (operator <o1> ^name do-task ^control-stuff* <c3>)
   (control-stuff* <c3> ^edit-from-subgoal-enabled* true)
 -->
   (control-stuff* <c3> ^edit-from-subgoal-enabled* true -
                        ^adds* <a1> + <a1> &)
   (aug* <u1> ^complete-schedule <l7> &, <l7> + <l7> + <l7> &)
   (list <l7> ^type* list + ^cdr <l6> + ^car <o4> +)
   (control-stuff* <c6> ^edit-from-subgoal-enabled* true +
                        ^reconsider-disabled* true +)
   (operator <o3> ^name simulate-trip + ^driver <d2> + ^truck <t4> +
                  ^trip <t6> + ^control-stuff* <c5> =, <c5> +)
   (list <l5> ^type* list + ^cdr <l4> + ^car <o2> +)
   (control-stuff* <c4> ^edit-from-subgoal-enabled* true +
                        ^reconsider-disabled* true +)
   (operator <d5> ^name simulate-trip + ^trip <d4> +)
   (add* <a1> ^class state + ^id <s4> + ^aug <u1> &, <u1> +)
   (trip <d4> ^name dummy +)
   (list <l4> ^type* list + ^cdr nil + ^car <d5> +)
   (operator <o2> ^control-stuff* <c4> + <c4> = ^trip <t2> + ^truck <t3> +
                  ^driver <d1> + ^name simulate-trip +)
   (control-stuff* <c5> ^reconsider-disabled* true +
                        ^edit-from-subgoal-enabled* true +)
   (list <l6> ^car <o3> + ^cdr <l5> + ^type* list +)
   (operator <o4> ^control-stuff* <c6> + <c6> = ^trip <t1> + ^truck <t5> +
                  ^driver <d3> + ^name simulate-trip +))

(sp p1549
   (goal <g1> ^object NIL ^state <s4> ^problem-space <p1> ^operator <o1>)
   (state <s4> ^dummy-att* true ^license <l3> <l2> <l1> ^city <c2> <c1>
               ^driver <d3> <d2> <d1> ^truck <t5> <t3> <t1>
               ^trip <t6> <t4> <t2>)
   (problem-space <p1> ^name top-space)
   (license <l3> ^truck-type medium ^holder brown)
   (city <c2> ^name gary ^truck traveler piper ^driver brown gray)
   (license <l2> ^truck-type small ^holder gray)
   (driver <d3> ^name brown ^drive-time 11)
   (truck <t5> ^type medium ^name traveler ^volume 640 ^weight-limit 10000)
   (trip <t6> ^name trip2 ^first-segment <s3>)
   (segment <s3> ^source gary)
   (license <l1> ^truck-type big ^holder green)
   (city <c1> ^name indy ^truck cannonball ^driver green)
   (driver <d2> ^name gray ^drive-time 12.5)
   (truck <t3> ^type small ^name piper ^volume 400 ^weight-limit 5000)
   (trip <t4> ^name trip1 ^first-segment <s2>)
   (segment <s2> ^source gary)
   (driver <d1> ^name green ^drive-time 11)
   (truck <t1> ^type big ^name cannonball ^volume 1280 ^weight-limit 32000)
   (trip <t2> ^name trip3 ^first-segment <s1>)
   (segment <s1> ^source indy)
   (operator <o1> ^name do-task)
 -->
   (goal <g1> ^operator <o1> @))
```

References

Filman, R. E. (1988). The Big Giant Trucking Problem. Intellicorp, Inc. 1975 El Camino Real West, Mountain View, CA 94040. Unpublished.

Filman, R. E. (April 1988). Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environment. *Communications of the ACM*, *31*(4), 382-401.

Laird, J. E. (1984). *Universal Subgoaling*. Doctoral dissertation, Computer Science Department, Carnegie Mellon University.

Laird, J. E.; Congdon, C. B.; Altmann, E.; and Swedlow, K. (October 1990). *Soar User's Manual: Version 5.2* (Tech. Rep.). Electrical Engineering and Computer Science Department, The University of Michigan. Also available from The Soar Project, School of Computer Science, Carnegie Mellon University as CMU-CS-90-179.

Minton, S. (August 1985). Selectively Generalizing Plans for Problem-Solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. 596-599.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning*, *1*(1), 47-80.

Rosenbloom, P. S.; Laird, J. E.; and Newell, A. (August 1987). Knowledge Level Learning in Soar. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*. 499-504.

Tambe, M.; Newell, A.; and Rosenbloom, P. S. (August 1990). The Problem of Expensive Chunks and its Solution by Restricting Expressiveness. *Machine Learning*, *5*(3), 299-348.

Yost, G. R. (May 1992). *TAQL: A Problem Space Tool for Expert System Development*. Doctoral dissertation, Computer Science Department, Carnegie Mellon University. Available as CMU-CS-92-134.

Yost, G. R. and Altmann, E. (1991). TAQL 3.1.3: Soar Task Acquisition Language User Manual. School of Computer Science, Carnegie Mellon University, December, 1991. Unpublished.

Table of Contents

List of Figures