Soar-mode v5.0 User's Manual
for Soar release 5.2 and 6.0

Frank E. Ritter,* Michael Hucka,** Thomas F. McGinnis***

10 December 1992
CMU-CS-92-205

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891

Abstract

Soar-mode is a major mode within the GNU-Emacs editor. It provides an integrated, structured editor for editing, running, and debugging Soar models on the production level. Productions are treated as first class objects. With keystroke (or menu) commands productions can be directly loaded, examined, and queried about their current match status. Listings of the productions that have fired or are about to fire can be automatically displayed. Soar-mode includes and organizes, for the first time, complete on-line documentation on Soar and a simple browser to examine this information.

*Psychology Department
Carnegie Mellon University, Pittsburgh, PA 15213-3890

**Electrical Engineering and Computer Science Department
The University of Michigan, Ann Arbor, MI 48109-2122

***School of Computer Science
Carnegie Mellon University, Pittsburgh, PA 15213-3891

This work was sponsored in part by a training grant from the Air Force Office of Scientific Research, Bolling AFB, DC; and 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.

Keywords: Structured integrated editors, AI programming tools, Soar.

Overview

As a minimal description, soar-mode provides the following features:

- A structured editor for Soar productions and loading them directly into a running Soar interpreter.
- The ability to treat Soar problem spaces and operators as levels in an outline, performing the usual outline processing functions on them.
- Commands to test and examine productions bound to keys and mouse buttons that are smart enough to tell which productions they are in or over.
- Complete on-line documentation for Soar, soar-mode, the Soar default productions, and the Soar source code.
- Functions to generate and maintain informative source code file headers.
- Tags file support for Soar productions (i.e., find-production-source-code) to enable fast and easy retrieval of production's source code.
- Support for running one or more Soar processes in separate buffers, and commands for interacting with these subprocesses.
- Support for Common Lisp programming (this may disappear in later releases).

Obtaining later versions of this manual

Updated versions of Soar-mode and this manual are available via anonymous FTP from centro.soar.cs.cmu.edu [128.2.242.245]. The README file in /afs/cs/project/soar/public/Soar5 (or /afs/cs/project/soar/public/Soar6) will provide you with a listing of the latest versions, and which files to pull to get them. Note: CMU's machines do not allow you to access intermediate directories in this path.

Requests for clarifications or bug reports should be sent to soar-bugs@cs.cmu.edu.

DISCLAIMER

1. Design philosophy

This short section provides a conceptual overview of soar-mode and motivation for the current approach. Eager users can skip it. The following sections describe soar-mode in detail.

Soar programmers develop applications in a cycle that typically consists of the following phases:

1. Create an initial conceptual description of a task.
2. Map the conceptual description into the Soar computational model.
3. Create a set of Soar productions implementing the task. This actually can involve several substeps:
    a. create source text files containing subsets of the task productions,
    b. write the productions, following certain style and naming conventions, and
    c. intersperse descriptive comments throughout the source text files.
4. Run Soar and incrementally define the set of productions to the Soar interpreter, entering a cycle that ends when the task is running:
    a. read productions into Soar,
    b. collect error messages and examine Soar's behavior,
    c. edit productions and go to (a).

The first two steps require much intuition and soul-searching on the part of the Soar programmer; during these stages not much programming actually takes place. The 3rd and 4th steps are computer-intensive. They require at least two separate program components, a text editor and a Soar interpreter, and considerable time expenditure on the part of the programmer.

Until recently the 3rd and 4th phases were not well integrated. A number of attempts have been made at providing better, more integrated Soar programming environments embodying these phases for Unix systems: Blake Ward's 1987 electric-soar-mode, Milnes and Shevis's original soar-mode, Olin Shivers soar-mode scaffolding, and Frank Ritter's recent Hypersoar-mode. All were based on Emacs-style editors. A customizable editor such as Emacs provides a good substrate for integrating the phases together. By using a Lisp subsystem supported by Emacs one can create a structured editor that understands properties specific to Soar.

This new soar-mode integrates all of the features and ideas from Ward's, Shevis', Shivers', Ritter's and Hucka's modes, along with new facilities, on top of an extensive Lisp mode called ILISP by Chris McConnell, and parts of Franz Inc.'s mode for their Allegro Common Lisp. It is nothing if not extensive, but we have tried to make it accessible also.

2. GNU-Emacs typing conventions

If you are terribly familiar with GNU-Emacs, you could skip this section. If you are not, it will behoove you to at least quickly scan it, for it is quite necessary for understanding the keystroke notations used in later sections.

You should first note that GNU Emacs is case sensitive. For consistency, all Emacs commands in this mode are in lower case. When a command is called "taql-insert-construct", it can not be typed in as "TAQL-insert-construct". TAQL and Soar are only capitalized when they denote Soar or the Task AcQuisition Language themselves.

The standard GNU Emacs conventions for specifying control characters are used. For example, "C-c C-t" represents holding down the control key and typing "c" and then holding down the control key and typing "t". (Control characters are not case sensitive, that is, C-C is the same as C-c.) Spaces (and other whitespace characters like tab) are represented by their name in broken brackets, for example, <TAB>. Escape key sequences are denoted by "M-" for meta, because some keyboards will actually have a meta key that can be held down, shifting all keys to meta just like control does. Keyboards without this feature provide escape, a way to preface a single key as meta. For example, typing <escape> and then "x" is represented by M-x. Sometimes escape may also be denoted by "^[".

Typing C-g at any point in GNU Emacs will abort the current action. This is also true in the Soar and TAQL sub-modes where C-g will abort commands cleanly. For example, typing C-g while completing a template will leave the template in a state that further expansion will correctly fill it in.

Many of the soar- and taql-mode commands begin with the Soar Command Prefix (SCP) that can be set by the user. This prefix is used to avoid clashing with other key bindings and yet remain flexible. The default value is C-c.[1] (If you use C-c to bind other commands, you can specify another character, such as C-6 which is not otherwise used in Emacs.) Code to set the SCP to another character is shown in the defaults file described below.

Open and close delimiters are counted for you. These include all those delimiters in Soar used only as delimiters; that is, all those in this string: "{}[]()". We don't treat < and > as delimiters because they are also used in preferences and could thus be miscounted. When you type a close delimiter, the corresponding open delimiter will be flashed: the cursor will briefly move back to the open delimiter, or if it is not currently displayed, the line that it occurs on will be displayed in the message line.

The keybindings described below are only provided when the buffer is in soar-mode. At all other times, after the initialization file has been loaded and soar-mode has been called at least once, you can execute commands by typing "M-x command-name".

---

[1]C-6 was to be the default, but it appears that some keyboards can't generate it.

3. Commands for editing Soar files

Buffers containing files of Soar source code (and ending in .soar or .soar5 or .soar6) are automatically put into soar-mode by the commands in the default init file. When such a file is read into an Emacs buffer it is automatically put into soar-mode, which binds several keys to new, Soar specific functions. The most important of the initial bindings are listed below. Each command normally works on the production the cursor is in or just after. Keys can be rebound by users (for an example, see the default init-file). A complete and current listing is available on-line by typing C-h m, or individual keys can be tested by C-h k {key sequence}. Where possible (and feasible), commands that apply to productions also apply to TC's.

```
Operations on productions and regions:
  C-c e      Eval production or function (i.e., send to running Soar)
  ESC C-x        (alternative binding)
  C-c C-e    Eval production or function and goto the Soar buffer
  C-c f      Find production source code (using tags)
  ESC .      Find function source code (using tags)
              (standard version, not customized for productions)
  C-c r      Eval region
  C-c C-r    Eval region and goto the Soar buffer
  C-c n      Eval next s-expression (i.e. production)
  C-c C-n    Eval next production and goto the Soar buffer
  C-c c      Show production soar-pclass
  C-c p      Print object around or before point (using spr)
  C-c x      Excise production
  C-c w      Copy production (or TC) to kill buffer. C-y will insert a copy.

Tracing operations:
  C-c B      Pbreak production (with arg, e.g. C-u, unpbreak)
  C-c t      Ptrace production (with arg, e.g. C-u, unptrace)

Match information:
  C-c m      Smatches on production
  C-c M      Full-matches on production

Movement commands:
  ESC C-a    Go to beginning of production
  ESC C-e    Go to the end of production
  ESC C-b    Go backward one clause (or s-expression)
  ESC C-f    Go forward one clause (or s-expression)
  ESC C-k    Kill next clause (or s-expression)
  C-c C-z    Zwitch to the Soar buffer,
                  with arg (i.e., C-u) go to end of the Soar buffer
  C-c b      Same as C-c C-z.

Misc. source file editing commands:
  ESC ;      Start an in-line comment
  C-c ;      Comment-region.  will put prefix number of copies of
              ";" before the lines in region.  to
              uncomment a region, use a minus prefix (e.g., C-u - 3 C-c ;)
  C-c l      Load file into Soar
  C-c C-l    Run-soar, start up a Soar
```

Running commands

These commands all use the same default arg (soar-default-drm-arg), which can be set in your .emacs file. It is initially 1, that is: (d 1), (r 1) and (macrocycle 1). It is updated after each use. An arg of 0 sets the default to nil (i.e., (d), (r) and (macrocycle)). (ESC-x soar-run-task also uses this arg.)

```
C-c .      (d arg)
C-c ,      (r arg)
C-c SPC    (macrocycle arg)
```

Useful functions from Common Lisp mode:

```
C-c t      Trace a Lisp function
               (untrace when given argument, i.e., C-u)
]          Close all open parentheses (i.e., a "super-)")
              up to first [
C-c )      Find unbalanced Lisp parenthesis
M-j        Insert new comment line (if in a comment), or a plain newline
if not.
```

Other effective standard Emacs commands:

```
ESC /      Auto complete
C-/        Undo (can be repeated multiple times)
```

Reset and help functions:

```
C-c C-c    Interrupt the running Soar or Lisp
C-c z      Pop out of all break(s) to top level in the Soar or Lisp buffer
C-d        Pop out of single break in the Soar or Lisp buffer
M-x panic-lisp
           Panic reset for the inferior LISP or Soar
C-c C-m    Run the soar-mode command menu
C-h m      Describe the current mode, e.g. soar-mode
C-c d      Get Lisp documentation string for symbol under point
C-c D      Look up Common Lisp function under point in reference manual
C-c A      Apropos for Common Lisp functions
```

These last two commands provide a buffer of information on the selected item. In this buffer "a" is bound to apropos, "m" to manual lookup, "s" to search-forward-see-alsos, and "C-c C-c" to flush-doc.

Outline functions

Soar-mode allows the user to collapse blocks of productions into problem spaces and operators labeled headings, like working with an outline processor.

Blocks of text and productions beginning with "; @problem-space name" or "; @operator name" can be collapsed into a single line "; @problem-space name ...". Operator blocks collapse as children nodes of problem spaces as shown in this example:

```
; @problem-space  top-space
; @operator top-operator ...
; @operator near-top-operator ...
; @problem-space lower-space ...
```

Problem space and operator labels that are indented two spaces

collapse an additional layer down. Through this mechanism, problem spaces can be nested within other problem spaces and operators.

The functions to manipulate the parents and children are included on the soar-mode menu under outl/, and bound to keys. The default keybindings begin with C-z. This is a user setable variable, and is included in the soar-mode-defaults.el file. The most frequent used commands are likely to be hide-node (C-z C-c C-h) and show node (C-z C-c C-s). If users find these keybindings unwieldy, there are examples of how to rebind functions to keys in the soar-mode-defaults.el file

If the tag @problem-space (and @operator) is indented further than one space, they are treated as lower levels, one level for every two spaces of indention.

Extended commands:

In addition, a number of other commands are not bound to keys but are available as extended commands (that is, to execute them type ESC-x command-name):

run-soar *or* **soar**	**Start up a Soar5 process by calling soar-image-name. You can set this in your .emacs file. The default value is "Soar5". Beeps when things are set up.**
soar6	**Start up a Soar6 process by calling soar6-image-name.**
cms	**Creates a buffer with the productions that are currently matched. With just soar-mode, the buffer is updated after every elaboration cycle. With the SX display the rate is adjustable.**
make-header	**Insert file header into current buffer.**
make-revision	**Add a revision line to header.** **After soar-mode has been loaded, these two commands can also be called when in other modes. See below for more information.**
make-tags-table	**Make a tags table, prompting for a list of files.**
remake-tags-table	**Update a tags table, replacing entries only for files that have changed since the TAGS file was saved.**
find-tags-table	**Switch tags table files.**
soar-count-productions	**Count the number of productions in current buffer.**
soar-list-production-names	**Collect the names of all productions in buffer and output in other window.**
insert-date-string	**Insert the date after point, e.g., "5-27-91 -" if insert-date-with-month-name is not nil, month name is used and day/month order is shifted, e.g. "27-May-91".**
insert-time-string	**Insert the time in military format after point, e.g., "1457".**
insert-current-time-string	**Insert the full current-time-string after point, e.g., "Mon May 27 15:00:57 1991".**
inspect-lisp	**These two functions will call inspect or describe**

```
describe-lisp              on the current s-expression, or when called with
                           an argument will prompt for the object.

0-argument Soar commands   All argumentless Soar commands, such as pgs are
                           available as M-x commands.  For example,
                           "M-x excise-task".
0-argument DSI commands    The argumentless DSI commands load-taql and
                           sx are available as M-x commands.  For
                           example, "M-x sx" will start up the command
                           interpreter loop.
run-task                   User defined function.  The soar-mode version
                           passes a numeric argument, soar-default-drm-arg,
                           which is used and set in the same way as for the
                           soar-mode version of d, r and macrocycle (see
                           "Running Commands" above).
macroexpand-lisp           Macro expansion (also available on Lisp menu).
```

This package modifies the Emacs variable *auto-mode-alist* so that the major modes defined in this package are invoked when certain types of source files are read into Emacs. The list of filename extensions given by the value of "soar-file-types" causes soar-mode to be invoked whenever a file with one of these extensions is visited. The default extensions are .soar and .soar5, but more can be added. Any buffer may be put into soar-mode by calling the function *soar-mode* interactively (e.g., by doing "ESC-x soar-mode").

Delete now converts tabs to spaces as it moves back.

Mouse Support

When running a separate GNU window under X windows, soar-mode defines special functions for the mouse buttons. These button bindings will not work when using GNU with the -nw, no (separate) window, option. If you set mouse buttons in your window manager's init file, you may clobber these definitions because the window manager's settings will probably have priority.

```
BUTTON              FUNCTION
----------------------------------------------------------------------------
left                [ These are not set by soar-mode, but are  ]
middle              [ left for the user to set.  Typically they]
right               [ set point, paste, and select text.       ]

SHIFT-left            Run Soar "spr" on item under cursor
SHIFT-middle          Run Soar "full-matches" on item under cursor
SHIFT-right           Run find-tag on item under cursor

CONTROL-left          Load definition (Soar or Lisp) under cursor
CONTROL-middle        Run Soar "ptrace" on item under cursor
CONTROL-right         [ available for future expansion or user settable ]
```

File Headers in Soar

Soar-mode provides extensive support for file headers. A file header is a descriptive comment block placed at the beginning of a source code file to keep track of such information as the author, creation date, last-modified date, and so on.

To make a header for the file in the current buffer, execute the command "M-x make-header". This inserts a standard header at the top of the buffer, based on the current mode (it will work for non-soar buffers too) and the functions listed in the variable "make-header-hooks". The default configuration of "make-header" would create the following header for a file named "test.soar":

```
;;;; -*- Mode: Soar -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File            : test.soar
;;;; Author          : Joe User
;;;; Created On      : Thu Mar 29 14:12:46 1990
;;;; Last Modified By:
;;;; Last Modified On:
;;;; Update Count    : 0
;;;; Soar Version    : 5.1
;;;;
;;;; PURPOSE
;;;;    |>Description of module's purpose<|
;;;; TABLE OF CONTENTS
;;;;    |>Contents of this module<|
;;;;  <tab> header 1
;;;;  <tab> header 2
;;;;   (use tab so you don't have to remember number of spaces)
;;;;
;;;; (C) Copyright 1992, University of Fichigan, all rights reserved.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

And if you use the header-status and header-history lines (the default), this is tacked on at the end:

```
;;;; Status          : Unknown, Use with caution!
;;;; HISTORY
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

If you use RCS, the header ends like this:

```
;;;; $Locker$
;;;; $Log$
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

You must fill in the PURPOSE, TABLE OF CONTENTS, Status, and HISTORY entries yourself. When you save a file containing a header created by "make-header", and soar-mode has been loaded (or the header files have been loaded by you or some other mode), the "Last Modified By", "Last Modified On" and "Update Count" fields are automatically updated. We expect this will be the primary way file headers are updated.

The "$Locker$" and "Log" fields are used by RCS, a more advanced way of updating files. If you use RCS to maintain your source files, the RCS commands will automatically generate the appropriate information in those fields.

The fields that are used are taken from the header-elements-list variable, which you can set in your .emacs file. The default value is shown in the soar-mode-defaults.el file included with this distribution in the directory above this manual (and also in this manual, below). For example, you could remove the history and status line from that list.

Editing Comments

When in the middle of a block comment, "M-j" will break the current line at the cursor point, then indent to the same column, continuing the comment. If you are not in the middle of a comment "M-j" will add a new line and indent (as will CR).

Using the TAGS Features

The Emacs "tags" facility provides the means to quickly locate the source code for a symbol. Standard GNU Emacs supports "tags" files for C, Lisp and a few other languages. Once a tags file has been selected, the keystroke "M- ." (meta-dot) over a symbol will move the cursor to the source for that function.

Soar-mode adds support for tagging the SP form of Soar productions. A "tags table" is a list of tuples of <name, file, position>, describing how function and variable declarations of a multi-file program are separated into source files. For each name (the "tag"), the file in which the name is defined and the position in the file is recorded in the tags table. The file which stores the tags table is called a "tags table file" and its conventional name is "TAGS".

Using the tags feature of soar-mode, it's possible to create a tags table file listing all the productions in a task. The simplest way to generate this file is to use the function "make-tags-table". It prompts for the names of the source file and the name of the TAGS table to be created. Therefore, if you have a directory full of .soar files you would like to "tagify", you could do the following:

ESC-x make-tags-table <CR> /pathname/*.soar <CR> /pathname/TAGS <CR>

Once the tags file has been constructed, you can use it to quickly locate the source code of a production if you have its name. To look up the definition of a production (or Lisp function or variable or whatever), first position the cursor over the name of the production, and then type "C-c C-f" which invokes the find-tag function. (The almost-equivalent alternative, "ESC-." (so called "meta dot") is retained for compatibility with normal Emacs/Lisp key bindings.)

This feature works in both Soar subprocess buffers and Soar text buffers. Thus you can lookup the definition of a production whose name you see printed anywhere in the buffer (provide, of course, the appropriate tags table has been built.)

A suggestion: whenever you built a tags table for a set of Soar files, include in the list of filenames ~soar/src/default.soar. This often comes in handy when you see default productions firing and you're wondering what they are doing.

The function "remake-tags-table" can be used to update a TAGS file for a set of files.

Sometimes you will need to switch tags table files. The function "find-tags-table" will prompt you for the name of a new tags table to use.

The function "tags-apropos" will display a list of all tags matching a given regular expression in the current tags table.

4. Getting help

Help on the current buffer’s set of key bindings and state variables (the mode) is always available by typing "C-h m" in GNU Emacs. This presents help on what commands are generally available in the current buffer (the one the cursor is in), and often a brief statement about the major mode’s general orientation.

The manual you are reading now is available on-line in the soar-mode distribution directory in manuals/soar-mode-manual.doc (as text) and soar-mode-manual.ps (PostScript version). The command menu, available through C-c C-m, provides you with a list of manuals under the "Doc" menu item.

Help is also available for Soar commands. Soar commands are essentially Common Lisp functions, and as such, their descriptions are available under the standard get-description-string command in soar-mode. The default binding of this command in soar-mode is "C-c d".

If you encounter what you think is a bug, you can (and should) generate a report automatically by typing "M-x soar-bug". (This command is also available in the menu.) You should send this report by mail, using whatever mail program you normally use.

5. Loading and running soar-mode

To use soar-mode, load the file soar-mode-defaults.el. Its directory is dependent on your site. At CMU (and other AFS sites) you can just put the following statement in your .emacs file:

(load "/afs/cs.cmu.edu/project/soar/5.2/emacs/soar/{soar-mode-release}/soar-mode-defaults.el")

Then, when you want to edit a file of Soar productions (ending in .soar or .soar6), simply read the file normally into Emacs and the buffer automatically will be put into soar-mode. Similarly, if you edit a Lisp file (ending in .lisp), ILISP alone will get loaded.

If you don't like the default settings set by soar-mode-defaults, and there are enough that you should view this file when you start to use soar-mode more, put customizations in your .emacs file after the command to load the defaults, overwriting them. You could also insert the contents of the soar-mode-defaults file into your .emacs file, and make the appropriate changes there.

To run Soar as a subprocess, invoke the command "run-soar" or just "soar" by typing "M-x soar". The normal-appearing Soar process will start up in a window of Emacs. If you use several versions of Soar, such as the Soar5 and Soar5+sx, you can use the example code in soar-mode-defaults.el to choose between them at start up.

Once you're editing a file of Soar code or running Soar, type "C-h m" for help on Emacs soar-mode, or "C-c C-m" for the soar-mode menu.

Before you exit Emacs, you should quit the Soar process as you would normally. Not doing so is normally safe, but it relies on Emacs successfully killing the Soar job when Emacs exits, which does not always happen.

Soar and soar-mode are not small programs. Old Emacs versions (18.57 and older) can run out of address space, and are more likely to do so when you run these programs. The best answer is to use a newer Emacs. If you cannot do so, as a preventative measure you should exit and restart Emacs more often, particularly if you examine large files.

If Emacs does crash, don't panic. Your modified files should be backed up as #file-name# in their normal directory. You can edit them, and when you save them, they are written out without the surrounding pound-sign (#) characters.

Modifying soar-mode through hooks

A hook is a place to hang either a function, or a list of functions that get executed after a specific corresponding event occurs. They allow you to customize soar-mode to suit yourself, particularly if you want to do something to each file or Soar process. They are implemented as Lisp variables, so you can put functions or lambda expressions on them just as you would add to any list. The table below shows when hooks get called, and their calling order.

```
Keymaps and their relationships:

Keymap to change or changes            Hook(s) to use
===========================================================================
soar-mode-map                          soar-mode-load-hook
  This is used in soar-mode, on
  files of Soar productions.

isoar-mode-map                         soar-mode-load-hook -or- soar-hook
  This is used in the buffer
  running Soar.

Choosing a Soar program to run:
  - each time Soar is called           soar-hook
  - at start of session only           soar-mode-load-hook

ilisp-mode-map                         ilisp-load-hook
comint-mode-map                        lisp-mode-hook
lisp-mode-map
  These are the underlying keymaps
  soar-mode is built on.  Leave them
  alone unless you are running
  stand-alone Lisp too.

Hooks you may want to use (Advanced version):

Event                          Hooks called (in order)
===========================================================================
soar-mode loaded               ilisp-site-hook, ilisp-load-hook,
                               soar-mode-site-hook, soar-mode-load-hook

soar-mode entered              lisp-mode-hook, soar-mode-hook

inferior-soar mode started     ilisp-mode-hook,
  (Soar started up)            <dialect>-hook (e.g., allegro-hook),
                               soar-hook, comint-mode-hook

Executed after inferior Soar   ilisp-init-hook
  (or lisp) is initialized

lisp-mode entered              lisp-mode-hook, comint-mode-hook,
lisp started up                ilisp-mode-hook, clisp-hook,
                               <dialect>-hook (e.g., allegro-hook)
                               comint-mode-hook

                               soar-after-ilisp-hook
```

For example, consider the following code. In addition to changing a few small things, it queries you each time you start up about which image to run, a plain Soar image, or the Soar+sx image:

```
(setq soar-hook
    '( (lambda ()
          ;; this could be done on soar-mode-load-hook just as well and faster
          (setq soar-date-with-month-name t)
          (if (and (not (comint-check-proc "*soar*"))
                   (y-or-n-p "Use Soar5+sx(y) or plain Soar5 (n)? "))
              (setq ilisp-program
                    "/afs/cs/project/soar/5.2/src/sx/5.1.1/Soar5+sx.acli")
              (setq ilisp-program
                    "/afs/cs/project/soar/5.2/2/bin/pmax/mach/franz/Soar5"))
          (setq tab-width 4))
       ))
```

<u>Installing soar-mode at a remote site</u>

Updated versions of Soar-mode and this manual are available via anonymous FTP from centro.soar.cs.cmu.edu [128.2.242.245]. The README file in /afs/cs/project/soar/public/Soar5 (or /afs/cs/project/soar/public/Soar6) will provide you with a listing of the latest versions, and which files to pull to get them. Note: CMU's machines do not allow you to access intermediate directories in this path.

In those directories, the complete source for soar-mode is mostly likely still named "soar-mode.<soar-mode-version>.tar.Z". You can copy this file directly if you are at Michigan or ISI, or you can retrieve it via anonymous-FTP.

You should move it into the directory it will live in. We suggest putting it in the directory {soar-on-your-system}/soar-mode/{soar-mode-version}.

You must uncompress it (uncompress soar-mode.tar.Z) and untar it (tar xf soar-mode.tar). You then must change the following variables in the indicated files:

File	Variable	New Value
soar-mode-defaults.el	soar-mode-home-directory	New untared directory name
soar-site.el	soar-mode-home-directory	New untared directory name
	soar-image-name	What you call Soar (Soar5 is default)
	In file section IV, paths pointing to manuals & source	Local paths for: Soar source file Soar bibliography file default productions file
	header-copyright-notice	Your name or site name
defdialect-soar.el	"(defdialect soar" args	Lisp version that Soar is based on, e.g., Lucid. Allegro is the default.
	comint-prompt-regexp	A regular expression that matches the prompt when Soar is running. A good check to make is to evaluate **(string-match comint-prompt-regexp** *"your Soar prompt"***)** and verify that it returns 0 (If the call returns nil, you must edit the value of the variable to match your prompt).
ilisp/<version>/soar.lisp		Code you want to be loaded each time Soar starts up, such as as (init-soar). For format, see file.

Recompiling the emacs code. After you have set the above variables, you should recompile the Emacs files so soar-mode runs faster. In a separate X GNU-Emacs window (not a gnu -nw window), you must first load soar-mode-defaults.el (by calling ESC-x load-file soar-mode-defaults.el), then load soar-mode (ESC-x soar-mode), and then compile it (ESC-x soar-compile-soar-mode). This recompile may prompt you to confirm each file compilation and you should always answer "y". It should take less than 5 minutes to do, and subsequent loading and running will be considerably faster. Note to Epoch users: Before recompiling, you should rename ilisp/<version>/epoch-pop-el to ilisp/<version>/epoch-pop.el .

Recompiling the lisp code. Ilisp loads some common lisp files into Soar5 when it starts up. Users at remote sites should compile them on their own after the emacs code has been compiled. To start this procedd, bring up an inferior Soar process (ESC-x soar) and then typing "ESC-X ilisp-compile-inits". It uses an extension for binaries that makes them appropriate and unique to the machine and LISP implementation. So after you have started up Soar, you need to call M-x *ilisp-compile-inits* only once for each machine/lisp combination. This may be slightly screwy, and if it queries you, you may have to answer ("n" seems good so far). You can also compile the files by hand if you don’t figure it out. The distributed version includes source files to load into and set up: Allegro, Lucid, CMULisp, and KCL.

If you have problems getting soar-mode up because it appears to be in an endless loop initializing, the problem is probably that soar-mode can not recognize your local Soar prompt. Even if you don’t know how to read regular expressions, you can check to see that comint-prompt-regexp in the defdialect in defdialect-soar.el matches your prompt with string-match (see note in table above). Examples of some prompts that will be matched by the default value of comint-prompt-regexp are "<cl> ", "<soar:user> ", "<Soar 6>", and "[1c] <cl> ". Some prompts that would not match are "foo", "<soar ", and "% ". If you suspect this is part of a problem, please include your regular Soar prompt in all bug reports.

If you run on a Sun, we include a file in the distribution provided by Josef Nerb called sunfun-nj.el. It provides an attempt to provide a uniform mechanism for binding Sun function keys. We have not tested it, but Josef and Ronald Leenes report that it works. If all of your machines are Suns, you should put in the soar-site file a line that loads it (i.e., (load "sunfun-nj")) and insert into the soar-site file their suggestions for insertion in the .emacs. Otherwise you can point it out to users as something they can load in their .emacs files.

6. INCOMPATIBLE CHANGES FROM PREVIOUS VERSIONS

* Soar6 is defined as a dialect of Soar. To start it up, type M-x Soar6. To change its image name, modify the value of soar6-image-name in your .emacs file.

* inferior-soar-mode-map is now isoar-mode-map.

* The variable soar-load-hook has been replaced by the more appropriately named soar-mode-load-hook.

* C-z has been rebound to access the new outlining commands. You can change it by setting outline-prefix-char in your .emacs to another character, such as "\C-c\C-z".

* "Hooks" variables have been renamed to be "hook" variables. This is more consistent with Emacs coding conventions. For example, soar-mode-hooks is now soar-mode-hook.

* The variable soar-image-arguments has gone away. You must either define a shell alias or holler to soar-bugs for help.

* Several keys get rebound, including C-x o, C-c. You can rebind them, see the example defaults file, soar-mode-defaults.el.

6. Complete keybindings listing

There are a several keymaps around, but only two of them are likely to be interesting to you. The first, isoar-mode-map, is the keymap that is used by the *soar* buffer. (This buffer's mode is actually inferior-soar-mode. That is to say, it is running an inferior process, in this case Soar, and is designed to interact with that process.) The second keymap, soar-mode-map, is used by buffers simply in soar-mode. Soar-mode is used when editing buffers filled with Soar commands or productions. The two maps share a large number of keybindings and differ only when the purposes of each mode diverge.

A complete listing of the keybindings is presented below, along with the keybindings of lisp-mode and inferior-lisp-mode. If you use the underlying package, ILISP, to edit Lisp, you may be interested in these maps also.

You can customize your keybindings by placing code to rebind your favorite keys on the soar-mode-hook.

```
Soar Mode keybindings:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Leading:
  * = Different between soar-mode and lisp-mode,
  + = Additions to soar-mode beyond what lisp-mode offers
       (i.e., unbound by ilisp),
  i = Inferior modes only (i.e., those with a running Soar or lisp).

Otherwise soar-mode inherits from lisp-mode and isoar-mode (inferior
soar mode) inherits from ilisp-mode (inferior lisp mode).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Changes to the global mode map:

  C-x o            popper-other-window
                       (requires a C-u prefix to get to popper windows,
                       including buffer listings.)
  LFD              newline-and-indent-lisp
  C-]              close-and-send-lisp
  ]                close-all-lisp
* TAB              indent-line-ilisp ; lisp-mode
* TAB              dabbrev-expand    ; during expansion in taql-mode
  DEL              backward-delete-char-untabify
  C-c              Prefix Command
i C-a              bol-ilisp
i C-d              delete-char-or-pop-ilisp
i RET              return-ilisp
  RET              newline-and-indent-lisp
  C-x C-f          find-file-lisp
  C-c g            popper-grow-output  (overridden in soar- and lisp-modes)

Control-c map:      (order: punctuation, Caps, small, control)

  C-c !            default-directory-lisp
i C-c #            raw-keys-ilisp
  C-c )            find-unbalanced-lisp
  C-c ;            comment-region-lisp
  C-c ^            edit-callers-lisp
  C-c .            (d 1)
  C-c ,            (r 1)
  C-c 1            popper-bury-output
* C-c SPC          mark-change-lisp
```

```
* C-c SPC         macrocycle ;soar
  C-c TAB         send-invisible

  C-c *           Prefix Command
  C-c * 0         clear-changes-lisp
  C-c * c         compile-changes-lisp
  C-c * e         eval-changes-lisp
  C-c * l         list-changes-lisp

  C-c A           clman-apropos
  C-c B           soar-pbreak-production
  C-c D           clman
  C-c I           inspect-lisp
* C-c M           macroexpand-lisp ;lisp
* C-c M           full-matches     ;soar  (Soar 6 matches 1)
* C-c M-m         really-full-matches ;soar6 matches 2
  C-c P           set-package-lisp
i C-c R           comint-msearch-input-matching
  C-c S           select-ilisp
  C-c T           ptrace-soar

  C-c a           arglist-lisp
  C-c b           switch-to-lisp
* C-c c           compile-defun-lisp ;lisp
* C-c c           soar-pclass        ;soar
  C-c d           documentation-lisp
  C-c e           eval-defun-lisp
+ C-c f           soar-find-tag
  C-c g           abort-commands-lisp
  C-c i           describe-lisp
  C-c k           compile-file-lisp
* C-c l           load-file-lisp
* C-c l           load-file-soar
* C-c m           macroexpand-1-lisp ;lisp
* C-c m           smatches           ;soar (Soar 6 matches 0)
  C-c n           eval-next-sexp-lisp
* C-c p           package-lisp
* C-c p           spr
  C-c r           eval-region-lisp
  C-c s           status-lisp
  C-c t           trace-defun-lisp
  C-c v           popper-scroll-output
* C-c w           compile-region-lisp
* C-c w           soar-copy-sp
+ C-c x           soar-excise-production
  C-c y           call-defun-lisp
  C-c z           reset-ilisp

+ C-c C-a         taql-add-clause
  C-c C-c         interrupt-subjob-ilisp    ; ilisp-mode
  C-c C-b         soar-load-buffer
  C-c C-c         compile-defun-and-go-lisp ;  lisp-mode
  C-c C-d         insert-date-string
  C-c C-e         eval-defun-and-go-lisp
+ C-c C-f         taql-fixup-construct
  C-c M-f         find-production-in-other-window
+ C-c C-l         run-soar
```

```
+ C-c C-m         run-soar-menu
  C-c C-n         eval-next-sexp-and-go-lisp
i C-c C-o         comint-kill-output
  C-c C-r         eval-region-and-go-lisp
+ C-c C-t         taql-insert-construct
  C-c C-w         compile-region-and-go-lisp
i C-c C-u         comint-kill-input
+ C-c C-z         outline-commands

+ C-c C-z C-n     Move to next visible heading
+ C-c C-z C-p     Move to previous visible heading.
+ C-c C-z C-f     Forward same level.
+ C-c C-z C-b     Backward same level.
+ C-c C-z C-u     Up a heading.
+ C-c C-z C-a     Show all.
+ C-c C-z C-s     Show sub-tree.
+ C-c C-z C-i     Show children (takes arg with C-u).
+ C-c C-z C-e     Show entry.
+ C-c C-z C-x     Show leaves.
+ C-c C-z C-h     Hide subtree.
+ C-c C-z C-t     Hide body.
+ C-c C-z C-c     Hide entry.
+ C-c C-z C-l     Hide leaves.
```

<u>Escape map:</u>   (order: punctuation, Caps, small, control)

```
  ESC "           replace-lisp
  ESC ,           next-definition-lisp
* ESC .           edit-definitions-lisp  ;lisp
* ESC .           find-tag               ;soar
  ESC ?           search-lisp
  ESC ‘           next-caller-lisp
  ESC RET         close-and-send-lisp
  ESC TAB         complete-lisp
+ ESC e           taql-expand-construct

i ESC N           comint-psearch-input
i ESC P           comint-msearch-input

i ESC n           comint-next-input
i ESC p           comint-previous-input
  ESC q           reindent-lisp
i ESC s           comint-previous-similar-input

  ESC C-l         previous-buffer-lisp
  ESC C-a         beginning-of-defun-lisp
  ESC C-e         end-of-defun-lisp
  ESC C-q         indent-sexp-ilisp
  ESC C-r         reposition-window-lisp
  ESC C-x         eval-defun-lisp
```

7. Default startup file

```
;;;; -*- Mode: Emacs-Lisp -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File             : soar-mode-defaults.el
;;;; Author           : Frank Ritter
;;;; Created On       : Wed Jun 20
;;;; Last Modified By: Frank Ritter
;;;; Last Modified On: Wed Nov 11 17:54:48 1992
;;;; Update Count     : 98
;;;;
;;;;
;;;;                How to load GNU Emacs Soar mode
;;;;
;;;; This file contains details on a default set of commands to load and use
;;;; soar mode.  Novice users with vanilla tastes can just always load this,
;;;; more advanced users will want to cut and paste the commands out of this
;;;; into their .emacs files
;;;;
;;;; To use Soar6, merely set the image name you want to use as
;;;; soar6-image-name, and call the function Soar6 instead of Soar.
;;;;
;;;; Loading this file will cause the following major changes to Emacs to
;;;; take place:
;;;;
;;;; a) The file cl.el will be loaded if was not already loaded.
;;;; b) Many configuration variables will have been set.  These affect
;;;;    the behavior of Soar mode and the code on top of which it is built.
;;;; c) The following packages will be autoloaded when the listed functions
;;;;    are called:
;;;;         Package            Function
;;;;         -------            ---------
;;;;          ILISP             run-ilisp
;;;;          ILISP             allegro
;;;;          soar              soar
;;;;
;;;; d) The following mappings between buffer modes and file name patterns
;;;;    will be established, causing Emacs to put buffers that have these
;;;;    file extensions be put into the specified mode automatically:
;;;;       File name suffix     Mode
;;;;       ----------------     ----
;;;;          .soar             soar
;;;;          .lisp             lisp-mode (ilisp-version)
;;;;
;;;;
;;;; TABLE OF CONTENTS
;;;;    i.      Variables that must be set
;;;;    ii.     How to set keybindings and hooks
;;;;    iii.    Load associated code
;;;;    iv.     Grungy things you have to do
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; changed auto-mode-alist to ignore .soar-aliases 27-Jul-92 -FER

(require 'cl)                             ; Loaded often-used CL extensions.
                                          ; we will use things here like pushnew
```

```
;;;
;;;     i.      Variables that must be set
;;;
;;; If you insert this into your .emacs, this is a section of active code that
;;; you can not comment out or remove.

;; This says where soar-mode lives.  Do not end the pathname with '/'.
(setq soar-mode-home-directory
      "/afs/cs/project/soar/5.2/emacs/soar/new")
;(setq soar-image-name "/../stego/usr/doorenbs/soar6")
(setq soar-image-name "/afs/cs/project/soar/5.2/2/bin/pmax/mach/franz/Soar5")
(setq soar6-image-name "/afs/cs/project/soar/5.2/emacs/soar/new/soar6")


;; Relative pathname off of soar-mode for ilisp-mode
(setq soar-ilisp-subdirectory "ilisp/4.12")

;; tags should include taql code too...
(setq soar-default-tags-table "DEFAULT-TAGS")


;;;
;;;     ii.     Variables that can be set
;;;
;;;  These variables are shown with their default values.  If you want to
;;;  change their values, copy this file (or portions of this file) to your
;;;  directory or insert it into your .emacs file.

;; The variable 'soar-image-name' must contain the name of the Soar
;; executable or be an alias which runs Soar.  It must be something which is
;; in your normal command execution path, or Emacs will not be able to find
;; it.  If you are NOT using the site default (set in 'soar.el'), then you
;; must set this variable manually.

;; (setq soar-image-name "Soar5")

;; If t, soar commands print descriptions into soar-diversion-buffer (*glide*)
;; buffer.  If nil, dumps into *soar* buffer.
;; (setq soar-print-into-diversion-p t)

;; Set 'lisp-no-popper' to 't' if you want all Lisp loading output (as opposed
;; to that of Soar productions) to go to the inferior Soar buffer rather than
;; into a pop-up window.  You should probably also set 'comint-always-scroll'
;; to 't' as well so that output is always visible.

;; (setq lisp-no-popper nil)  ;default is nil
;; (setq comint-always-scroll nil) ;default is nil

;; Set the following to 't' to print out the month in 'insert-date-string' as
;; letters, and in 30-Oct-91 order, rather than as 10-30-91.
;; (setq insert-date-with-month-name nil)  ; default is nil

;; Set to 'nil' if you don't want 'C-x o' to skip pop-up buffers, such
;; as '*Buffer Menu*' or '*Help*'.  Default is 't'.
;; Can also be set to a list of pop-up buffers you want to skip.

(setq popper-buffers-to-skip nil)
```

```
;; Pop to the CMS (continuous match set) buffer if it is being written
;; to (default 'nil').

;; (setq pop-to-cms nil)

;; if this is C-^, it is also C-6 unshifted
;; (setq soar-command-prefix "\C-c")

;; The outline commands are bound on C-c then this prefix char.
;; (setq outline-prefix-char "\C-z")

;; If soar-erase-diversion-buffer-p is t (default), erase the diversion
;; buffer each time you use it.
;; (setq soar-erase-diversion-buffer-p t)

;; Name of the diversion buffer.
;; (setq soar-diversion-buffer-name "*glide*")

;; Popup the diversion buffer if t (the default) something gets put in there.
;; (setq soar-popup-diversion-buffer-p t)

;; soar-header-hooks contains a list of what to put on the header when
;; make-header is called.  The default is shown below.
;;
;; (setq soar-header-hooks
;;       '(;; a top line with mode  comes for free
;;         ;; a divisor line        comes for free
;;         header-blank
;;         header-file-name
;;         header-author
;;         header-creation-date
;;         header-modification-author
;;         header-modification-date
;;         header-update-count
;;         soar-version
;;         taql-version
;;         ;;Put PURPOSE and TOC near top
;;         header-blank
;;         header-purpose
;;         header-toc
;;         header-copyright
;;         ;;Generally want either RCS stuff or header-history
;;         ;;at CMU and elsewhere, fewer users and fewer non-hacky use RCS,
;;         header-divisor-line
;;         header-status
;;         header-history
;;         ;;header-rcs-locker
;;         ;;;;header-rcs-header
;;         ;;header-rcs-log
;;         ;; divisor line        comes for free
;;         ))

;; Soar will beep when initialized if t.
;; (setq soar-beep-after-setup-p t)


;;;
```

```
;;;     ii.     How to set keybindings and hooks
;;;
;;; If you wish to change the keybindings or add to them for buffers in
;;; soar-mode, put the changes on the ‘soar-hook’ in your .emacs file
;;; with code comparable to the code below.  Similar code could be put on
;;; the ‘inferior-soar-mode-hook’ for buffers running Soar.
;;;
;;; Here is an example of a soar-hook function which defines ‘C-c C-t’
;;; to run function ‘favorite-cmd’ in both Soar mode buffers and Soar process
;;; buffers; and redefines a mouse button.
;;; Further information on how to reset the mouse keys are available in the
;;; soar-mouse-x.el file.
;;;

;; Example set up for stuff to do when putting a buffer into soar-mode
;(setq soar-mode-hook
;     ’((lambda ()
;         (visit-tags-table soar-default-tags-table)
;         ;; works for code buffers
;         (define-key soar-mode-map "\C-z" ’favorite-cmd2)
;         ;; this command rebinds the control middle mouse to what it was
;         ;; originally.
;         (define-key mouse-map x-button-c-middle-up ’x-cut-and-wipe-text)
;        )))

;;; works for running Soar buffers
;(setq soar-mode-hook
;     ’((lambda ()
;         (visit-tags-table soar-default-tags-table)
;         (define-key isoar-mode-map "\C-z" ’favorite-cmd)
;         ;; this command rebinds the control middle mouse to what it was
;         ;; originally.
;         (define-key mouse-map x-button-c-middle-up ’x-cut-and-wipe-text)
;        )))


;;; Example of how to set soar-hook, and what you can put on it, which gets
;;; gets called when an inferior (running) Soar starts up.
;;;

;(if (not (boundp ’soar-hook)) (setq soar-hook nil))
;;; soar-hook gets called after ilisp inits, but before soar gets called
;(setq soar-hook
;   (append soar-hook
;      ’((lambda ()
;         ;; use C-6 instead of C-c as command prefix
;         (setq soar-command-prefix "\C-6")
;         ;; start by visiting a tags table
;         (visit-tags-table "/afs/cs/project/soar/5.2/2/lib/TAGS")
;         ;; ask which soar I want iff don’t have a live one
;         (if (and (not (comint-check-proc "*soar*"))
;                (y-or-n-p
;                 "Use (perhaps) local copy of Soar5+sx(y) or Soar5 (n)? "))
;             (setq ilisp-program
;                   "/afs/cs/project/soar/5.2/src/sx/new/Soar5+sx.acli"))
;
;        ;; or just always use a single version
```

```
;         (setq ilisp-program
;            "/afs/cs.cmu.edu/project/soar/5.2/2/bin/pmax/mach/franz/Soar5")))
;            ;; set my header notice
;         (setq header-copyright-notice "Copyright 1991, Frank Ritter.")
;      ))))


;;;
;;;      iii.    Load associated code
;;;
;;; 'utilities/taql-indent-line.el' provides code to indent TC's correctly
;;; when using soar- or taql-mode (but is not automatically loaded by
;;; soar-mode).
;;;
;;; (e.g., Set up ilisp so you can find it w/o soar-mode.)
;;; We believe that if you don't use ilisp alone, that you could comment or
;;; cut this out to save space. (But I wouldn't do that if I were you.)
;;;

;; Establish path to the home directory of Soar mode and ILISP mode.
;; The latter is set here so that you can use ILISP mode independently
;; of Soar mode:

(pushnew soar-mode-home-directory load-path)
(pushnew (concat soar-mode-home-directory "/" soar-ilisp-subdirectory)
         load-path)

;; To load 'utilities/taql-indent-line' iff it has not been loaded.
;; Cut this into your soar-mode-hook, if you wish.
;; (require 'taql-indent-line)

;; Some handy things for working on just lisp that you are carrying around
;; too, let's not let them go to waste:

(autoload 'run-ilisp "ilisp" "Select a new inferior LISP." t)
(autoload 'allegro   "ilisp" "Inferior Allegro Common LISP." t)

;; Additional useful functions.

(autoload 'comint-mem "comint"
  "Test to see if ITEM is equal to an item in LIST.
Option comparison function ELT= defaults to equal." t)

(autoload 'add-hook "comint-ipc"
          "Add a function to a hook if not already present." t)

(autoload 'soar-manual "utilities/soar-manual"
          "Read an online manual, such as for Soar or soar-mode." t)

(autoload 'clman "allegro/allegro-mode-init"
          "Read about a specific topic in the online CL manual." t)

(autoload 'clman-apropos "allegro/allegro-mode-init"
          "List topics matching a subexpression in the online CL manual." t)

;; set the ilisp-command-prefix in case you are headed into ilisp first
(setq ilisp-prefix "\C-c")
```

```
;;; Emacs would normally put .lisp files in it's default simple lisp-mode.
;;; This makes reading a lisp file load in ilisp.
(add-hook 'lisp-mode-hook
          (function
           (lambda ()
            (require 'ilisp))))

;;;
;;;     iv.     Grungy things you have to do
;;;
;;; This is live code that you have to have, but that you don't have to
;;; understand if you leave it alone.
;;;

;; Put files that end in .soar into soar-mode
;; remove comment to do it for .Soar files too

(defvar soar-file-types
   '("\\.soar$"
     ;; "\\.Soar$"
     "\\.soar5$"
     "\\.soar6$"))

(dolist (suffix soar-file-types)
  (if (not (assoc suffix auto-mode-alist))
      (push (cons suffix 'soar-mode) auto-mode-alist)))

;;; make calling  {run-soar, soar, soar-mode} load the mode code.
(mapcar
 (function (lambda (x) (autoload (car x) "soar" (car (cdr x)) t)))
 '((run-soar  "Starting up an inferior (buffer) soar process.")
   (soar  "Another way to start up an inferior (buffer) soar process.")
   (soar6  "Another way to start up an inferior (buffer) soar process.")
   (soar-mode "When editing a file of Soar productions or lisp code.")
   ))

(defun Soar ()
  (interactive)
  (soar))

;;;; This should be covered by the new  use-soar-mode-if-available  variable.
;;; If taql-mode is around, put .taql files into soar-mode as the major
;;; mode when they start up, with taql-mode behind
;;; This works if soar-mode is loaded second.
;(if (assoc "\\.taql" auto-mode-alist)
;    (set-default 'auto-mode-alist
;             (append
;                (mapcar '(lambda (x)
;                           (cons x 'soar-and-taql-mode))
;                          '("\\.taql$"))
;                auto-mode-alist)
;    ))
```

Table of Contents