Bill J. Ellis
February 3, 2005

Tweaking the SPADE Proof Checker

Contents

*Hums are short notes intended for distribution between those involved with EPSRC grant GR/T12289/01. Hums describe ϵ-baked ideas, where $1 \geq \epsilon \geq 0$. (Hum refers to both the Praxis Humming bird and Winnie-the-Pooh's indirect approach to writing: "Poetry and Hums aren't things which you get, they're things which get you.").

1 Introduction

In Hum 2 changes to the SPADE Proof Checker were proposed. Here these changes are implemented and documented. The result of completing these changes will be to enhance the SPADE Proof Checker with a new, more controlled, mode of operation: "tamed". Where the SPADE Proof Checker is being used in this mode it will be referred to as SPADE Proof Checker-Tame. Note that the desired changes involve getting the SPADE Proof Checker not to do things it does already and to do things it does not do already. The SPADE Proof Checker is being curtailed in some areas and extended in others.

2 Rapid design

2.1 The original SPADE Proof Checker-Tame

A version of the SPADE Proof Checker-Tame was implemented as part of the NuSPADE project. Initially this SPADE Proof Checker-Tame refined the behaviour of the SPADE Proof Checker by directly modifying its existing commands. However, it was later found to be much more straight forward to gain the desired functionality by implementing new commands. This approach will be repeated for this new version of SPADE Proof Checker-Tame.

The original version of the SPADE Proof Checker-Tame did not maintain backwards compatibility with the SPADE Proof Checker. The code was directly modified, replacing undesired existing behaviour as it was encountered. Thus, the original version of the SPADE Proof Checker-Tame is helpful in pinpointing the areas where changes need to be made to the SPADE Proof Checker. However, the actual implementation of these changes needs to be refined to achieve a true, backwards compatible, extension of the SPADE Proof Checker.

2.2 Configuration options

The SPADE Proof Checker has the following existing qualifiers (extract from the SPADE Proof Checker user manual):

> ...It is possible, however, to specify the filename, followed by other optional qualifiers, on the operating system command-line. The syntax which should be used is:
>
> `CHECKER [ filename [ /qual1 ] [ /qual2 ] ... [ /qualn ] ]`
>
> for the VAX/VMS or Windows NT/2000 platforms, or
>
> `checker [ filename [ -qual1 ] [ -qual2 ] ... [ -qualn ] ]`
>
> on the Sun platform. Each qualifier may be one of the following forms:
>
> ```
> command_log=filename
> execute=filename
> load=(lib1,...,libn)
> proof_log=filename
> resume
> ```

Note that the filename must come first. Further note that additional qualifiers are only processed if placed after a filename. The SPADE Proof Checker also consults an initialisation file, configuring its behaviour accordingly. The commands seen in the initialisation file, called `checker.ini`, control many aspects of the SPADE Proof Checker, including its reasoning abilities and behaviour during interactive proof. The commands are:

> `auto_done`, `auto_newvc`, `command_logging`, `display_subgoals_max`, `display_var_free_only`, `indentation_increment`, `inverse_video`, `newline_after_prompts`, `normal_video`, `print_command`, `prooflog_width`, `record_consults`, `show_vc_changes`, `simplify_during_load`, `simplify_in_infer`, `typechecking_during_load`, `use_subst_rules_for_equality`

2.3 Extending configuration options for tame mode

SPADEase[1], in particular the SPADEase-Verifier, will want to invoke the SPADE Proof Checker fully automatically. It would not seem appropriate to modify the static `checker.ini` file to configure the SPADE Proof Checker for these specialised sessions. Thus the SPADE Proof Checker should accept additional qualifiers on the command line.

Although the changes made to the SPADE Proof Checker will add additional (and safe) commands, these are unlikely to be valuable to the regular user of the SPADE Proof Checker. The intention is to only make available the extra commands when the SPADE Proof Checker is operating as SPADE Proof Checker-Tame.

2.4 Qualifier to activate the SPADE Proof Checker-Tame

One qualifier is considered:

```
checker -tame INPUT_FILE VC_NUMBER CONC_NUMBER CMD_FILE RESULT_FILE
```

This starts the SPADE Proof Checker-Tame on the selected problem (`INPUT_FILE`, `VC_NUMBER`, `CONC_NUMBER`) and automatically executes the provided proof script (`CMD_FILE`). In normal behaviour the proof script should contain a command that causes the SPADE Proof Checker-Tame to store the result in the result file (`RESULT_FILE`) and terminate. Omitting this command will result in control being passed to the user in an interactive session, ideal for debugging proof plans. Note that this qualifier will only be accepted exactly as above. Further, note that this qualifier can not to be mixed with the existing qualifiers supported by the SPADE Proof Checker.

2.5 Maintain semantics of existing settings

The SPADE Proof Checker supports a collection of settings that refine its behaviour. There are cases where a pedant, concerned with having total control over the SPADE Proof Checker, might argue that these settings are not always strictly implemented. However, the SPADE Proof Checker behaviour is designed for user interaction and it is the case that, at least, the spirit of these settings are followed. It is thought that a pedantic interpretation and implementation of the existing SPADE Proof Checker settings would upset many more users that it would please[2]. Thus, the SPADE Proof Checker will be refined by introducing new settings (internally) rather than messing around with the existing ones.

2.6 Avoid reliance on existing settings

There will be cases where the existing SPADE Proof Checker settings could be exploited to achieve aspects of the desired behaviour of the SPADE Proof Checker-Tame. An alternative is to have a single master setting and amend the code accordingly. For example, given the code snippet:

```
layout(MS, L, F) :-
    (
       simplify_in_infer(on),
       simplify(F,FF)
```

Should the activation of SPADE Proof Checker-Tame ensure that `simplify_in_infer(on)` is not set or should the code be modified to:

```
layout(MS, L, F) :-
    (
       \+ tame_active(yes),
       simplify_in_infer(on),
       simplify(F,FF)
```

And ensure that the master setting `tame_active(X)` is set accordingly?

It is thought that cases will exist where the existing SPADE Proof Checker settings provide constraints that are not required by the SPADE Proof Checker-Tame. Further, by having a master setting for the SPADE Proof Checker-Tame, it will be straight forward to identify the areas of code affected by the tame mode. Finally, having

[1] Andrew suggested this alternative name for the new system. Previously Andrew coined the inspired 'NuSPADE', maintaining a reference to the SPADE system, acknowledging previous projects that extended and improved upon existing systems (e.g. NUPRL), and (possibly accidentally) has the delightful anagram: 'Nude Spa'. Recognising such marketing savvy the suggestion is taken, thus the SPADE Helper becomes SPADEase, SPADE Helper Parser becomes SPADEase-Parser, and SPADE Helper Verifier becomes SPADEase-Verifier.

[2] (U-1) to 1, where U is the number of users, and is certainly greater than 1.

the reliability of SPADE Proof Checker-Tame depend on the behaviour of other settings (which are arguably semantically variable) is not ideal.

Note that it would be straight forward to remove such a master setting and rely upon the existing commands, by scanning the source for occurrences of the master setting, deleting lines accordingly, and ensuring that on entering the tame mode the appropriate SPADE Proof Checker settings are set.

2.7 Supersede checker.ini file

The original version of the SPADE Proof Checker-Tame relied upon the following `checker.ini` for correct behaviour:

```
set prooflog_width to 0.
set display_var_free_only to on.
set simplify_in_infer to off.
set simplify_during_load to off.
set auto_done to off.
```

Thus all of these settings give a strong indication as to the points in the SPADE Proof Checker that should be tweaked in order to have a version of the SPADE Proof Checker-Tame that behaves correctly irrespective to the contents of the `checker.ini` file.

2.8 Do not (unnecessarily) pollute the code

The additions to the SPADE Proof Checker are likely to number a couple of hundred lines of Prolog[3]. Placing this code in its natural location (based on functionality) would result in many tiny code fragments distributed throughout the code base. It is felt that grouping together the new code makes for a less natural but more practical extension. Note that, in a few cases, direct changes to the code will be required. However the vast majority of the new code can be isolated from the core code base.

2.9 Have no side effects

The SPADE Proof Checker amends files as part of its normal behaviour. The execution of the SPADE Proof Checker-Tame should not effect the status of the SPADE Proof Checker and thus should not modify these files.

3 Describing SPADE Proof Checker-Tame

The following qualifier will place the SPADE Proof Checker in a "tame" mode. This mode forces the SPADE Proof Checker to behave in a restricted manner, making it more straight forward for external tools to automatically control the SPADE Proof Checker.

```
CHECKER -tame INPUT_FILE VC_NUMBER CONC_NUMBER CMD_FILE RESULT_FILE
```

The parameters are as follows:

- **INPUT_FILE** - The file containing the input verification conditions.
- **VC_NUMBER** - The number of the verification condition being proved.
- **CONC_NUMBER** - The number of the conclusion of the verification condition being proved.
- **CMD_FILE** - The command file that will prove the selected conclusion of the selected verification condition.
- **RESULT_FILE** - The file where the result will be stored. (Note, this is currently not implemented. It is the case that some means for returning a result will be required. However the most useful form for this depends on the behaviour of the controlling system, the SPADEase-Verifier, the exact nature of which remains undecided.)

The SPADE Proof Checker-Tame has a few additional commands:

- **tame_subgoal_on_term TERM** - Will always subgoal on the provided `TERM`.

[3]This is a not a lot of code. However, this is much larger than the meagre changes proposed to the SPADE Simplifier in Hum 3, prompting the need for some reasonable coding policy.

- **tame_subgoal_on_conc NUMBER** - Will always subgoal on conclusion `NUMBER`.
- **tame_induction** - Perform induction in exactly the same manner as the SPADE Proof Checker. The only difference is that this version of the command also asks the user to provide the name of the generated induction variable.
- **tame_simp_and** - Decompose conjoined terms into separate hypotheses. (Note this is seen as a temporary measure. This command is required because there is currently not an ideal version of the `infer` command. Once the format of proof plans are better understood a `tame_infer` command should be created, allowing this command to be deleted.)
- **tame_done** - Try to prove a conclusion. This does not automatically close a subgoal if all of its conclusions are proved.
- **tame_all_done** - If all of the conclusions are proved in a subgoal, this will close the subgoal.
- **tame_finish** - Finish the proof attempt, exit the SPADE Proof Checker, registering the success of the proof accordingly.
- **tame_echo TEXT** - Does nothing, without failing. Note that the SPADE Proof Checker echos every command to the screen, so this command need not do anything to display `TEXT`.
- **tame_show_depth** - Show the current proof depth.
- **tame_compare_depth NUMBER** - Compare the current proof depth with `NUMBER`. If the two are not the same the command will fail.
- **tame_help** - List the additional commands available when running the SPADE Proof Checker-Tame.

4 Brief description of changes

- `checker.pl` - **Consult tame** -
  Add a line to consult the new code held in `tame.pl`.
- `checker.pl` - **Modify startup sequence** -
  Modify the startup sequence to look for the tame qualifier and initialise accordingly where found.
- `done3.pl` - **Fix done command** -
  Change `command_arg(expression, CONCS)` to `command_arg(to_do, CONCS)`, so that it correctly picks up the asserted command argument. This fixes the ability to invoke done on selected conclusions, which is useful for SPADE Proof Checker-Tame.
- `done3.pl` - **Divide done and all done** -
  Change `done` so that is does not backtrack and call `all_done` while in tame mode.
- `done3.pl` - **Prevent extra call to done** -
  For reasons unclear, an extra call to done is made within done, even where `auto_done(on)` is not set. These extra calls are blocked when in tame mode.
- `done3.pl` - **Do not automatically close VCs** -
  Modify `done` so that `auto_done(on)` features are not considered while in tame mode.
- `getdcldat.pl` - **New qualifiers** -
  Add new qualifiers to activate the tame mode.
- `inferenc2.pl` - **Do not simplify at infer** -
  Modify `infer(A)` so that zero simplifications are performed when in tame mode.
- `loadvc5.pl` - **Do not simplify during layout phase of load** -
  Modify `layout(A, B, C)` so that zero simplifications are performed when in tame mode.
- `loadvc5.pl` - **Do not simplify during store phase of load** -
  Modify `store_vc(A,B,C,D)` so that the VC is unchanged when in tame mode.
- `loadvc5.pl` - **Do not simplify during formulae processing of load** -
  Modify `process_formula(A, B)` so that zero simplifications are performed when in tame mode.

- `tame.pl` - **Tame specific code -**
  The vast majority of the new SPADE Proof Checker-Tame code is placed in this file. The main loop is here and all of the new commands are here.

- `toplevel.pl` - **Fix execute command -**
  Change `command_arg(FILE)` to `command_arg(filename, FILE)` so that it correctly picks up the asserted command argument.

- `toplevel.pl` - **Register new commands -**
  Add entries for `match_command(A, B)` to register the new commands.

- `toplevel.pl` - **Handle arguments of new commands -**
  Add entries for `parse_command_arguments(A, B)` to handle arguments of the new commands.

- `toplevel.pl` - **Loud failure -**
  Add a very loud failure message for SPADE Proof Checker-Tame. This will be issued where a legal command fails.

- `utilities.pl` - **Register trivial -**
  Mark which of the new commands do not affect the status of the VC via `trivial_command(A)`.

- `utilities.pl` - **Always add hypotheses -**
  Modify `add_new_hyp(A, B)` so that, while in tame mode, new hypotheses are inserted even if they match existing hypotheses (except for 'true', which is never added).

5 Executive summary

Changes to the SPADE Proof Checker are described so that it can be placed in a new "tame" mode. This mode forces the SPADE Proof Checker to behave in a restricted manner, making it more straight forward for external tools to automatically control the SPADE Proof Checker. The tame mode is activated by providing a qualifier on the command line. Once in the tame mode, the behaviour of some SPADE Proof Checker commands will be slightly refined. Further, additional commands will be available.

Note that this implementation of the SPADE Proof Checker-Tame should be viewed as a first draft. It extends the SPADE Proof Checker in a coherent manner with desired additional functionality. However, this behaviour will inevitably be fine tweaked, following the integration between SPADE Proof Checker-Tame and the emerging SPADEase tools.

A All SPADE Proof Checker changes

A.1 aritheval.pl

A.2 aritheval.pl

A.3 cases2.pl

A.4 checker.pl

```
118a119,120
> :- bonsult('tame.pl').
>
179a182,189
>     fail.
>
> startup_sequence :-
>     tame_active(yes),
>     !,
>     startup_sequence_tame.
>
> startup_sequence :-
201a212,245
> startup_sequence_tame:-
>     %load_vc recieves the file via the asserted predicate: cmd_line_filename(FILENAME)
>     tame_input_file(INPUT_FILE),
>     assert(cmd_line_filename(INPUT_FILE)),
>     load_vc,
>
>     %Display checker tame message.
>     nl,
>     write('Welcome to the SPADE Proof Checker *Tame*'),
>     nl,
>
>     %Start checker tame.
>     dopop(' procedure(); chain(prolog_invoke(% "start" %)) endprocedure -> interrupt; '),
>     !,
>     start_checker_tame.
>
> startup_sequence :-
>     load_buffered_libs,
>     write_log,
>     fail.
> startup_sequence :-
>     do_do_newvc,
>     see_correct_input_stream,
>     execute_command(newvc),
>     write_log,
>     fail.
> startup_sequence :-
>     dopop(' procedure(); chain(prolog_invoke(% "start" %)) endprocedure -> interrupt; '),
>     !,
>     start.
>
>
>
>
```

A.5 cipher.pl

A.6 contra.pl

A.7 datecheck.pl

A.8 declar.pl

A.9 decrypt.pl

A.10 deduce.pl

A.11 deduction.pl

A.12 done3.pl

```
56c56
<          command_arg(expression, CONCS),
---
>         command_arg(to_do, CONCS),
65a66
>          \+ tame_active(yes),
102a104
>        \+ tame_active(yes),
167a170
>                 \+ tame_active(yes),
171a175
>                 \+ tame_active(yes),
172a177,178
>              ;
>                 true
```

A.13 fwdch2.pl

A.14 getdcldat.pl

```
76,77c76
<         !
<     .
---
>         !.
81a81,96
>
> %Search for checker tame qualifiers before regular qualifiers.
> %Accept perfect match only.
>
> process_dcl_args(LIST):-
>     %This processes the input, transfering it from some an obsqure
>     %poplog type to the expected prolog types.
>     split_qualifiers(LIST, LISTOUT),
>     process_tame_qualifiers(LISTOUT),
>     !.
>
> %If the above fails, the checker is not in tame mode.
> process_dcl_args(LIST) :-
>     assert(tame_active(no)),
>     fail.
>
89a105,148
> process_tame_qualifiers([QUALIFIER,
>                          INPUT_FILE,
>                          VC_NUMBER_ATOM,
>                          CONC_NUMBER_ATOM,
>                          CMD_FILE,
>                          RESULT_FILE]):-
>     %Check and process qualifiers.
>     qualifier_match(QUALIFIER, 'tame'),
>     convert_atom_to_integer(VC_NUMBER_ATOM, VC_NUMBER_INT),
>     convert_atom_to_integer(CONC_NUMBER_ATOM, CONC_NUMBER_INT),
>
>     %Store these.
>     assert(tame_input_file(INPUT_FILE)),
>     assert(tame_vc_number(VC_NUMBER_INT)),
>     assert(tame_conc_number(CONC_NUMBER_INT)),
>     assert(tame_cmd_file(CMD_FILE)),
>     assert(tame_result_file(RESULT_FILE)),
>     assert(tame_status(batch)),
>     assert(tame_active(yes)),
>     !.
>
> qualifier_match(QUALIFIER, MATCH_ATOM):-
>     write(QUALIFIER),nl,
>     qualifier_prefix(QP),
>     write(QP),nl,
>     %Check QUALIFIER is off form: <prefix>characters
>     %and extract the characters as CHARS.
>     name(QUALIFIER, [QP|CHARS]),
>     write(CHARS),nl,
>     name(ATOM, CHARS),
>     write(ATOM),nl,
>     MATCH_ATOM=ATOM,
>     !.
>
> convert_atom_to_integer(VC_NUMBER_ATOM, VC_NUMBER_INT):-
>     name(VC_NUMBER_ATOM, CHARS),
>     name(VC_NUMBER_INT, CHARS),
>     integer(VC_NUMBER_INT),
>     !.
>
> convert_atom_to_integer(VC_NUMBER_ATOM, VC_NUMBER_INT):-
>     write('ERROR: Expected integer and found: '),nl,
>     write(VC_NUMBER_ATOM), nl,
>     halt.
```

A.15 getlicence.pl

A.16 getlicence_vax.pl

A.17 help.pl

A.18 induction.pl

A.19 infer2.pl

A.20 inferenc2.pl

```
45a46
>           \+ tame_active(yes),
```

A.21 initialise.pl

A.22 initvals.pl

A.23 interrupt.pl

A.24 listthm.pl

A.25 loadlib.pl

A.26 loadvc5.pl

```
1201a1202
>           \+ tame_active(yes),
1737a1739
>     \+ tame_active(yes),
1748a1751,1754
> layout(MS, L, F) :-
>     store_vc(MS, L, 1, F),
>     !.
>
1784a1791
>     \+ tame_active(yes),
```

A.27 lvc.pl

A.28 make_vax.pl

A.29 myspy.pl

A.30 newrules.pl

A.31 newvc.pl

A.32 prooflogs.pl

A.33 quantif.pl

A.34 records2.pl

A.35 repall.pl

A.36 replace2.pl

A.37 rulefiles.pl

A.38 save.pl

A.39 semistan.pl

A.40 setflags.pl

A.41 simp.pl

A.42 simplify.pl

A.43 spycheck.pl

A.44 standard.pl

A.45 subgoal.pl

A.46 tame.pl

```
0a1,478
> %Extensions for SPADE Proof Checker Tame.
>
> %------------------------------------------------------------
>
> %Interactive mode. The user controls checker tame.
> start_checker_tame:-
>     tame_status(interactive),
>     begin_selected_vc,
>     check_conclusion_exists,
>     main_checker_tame.
>
> %Bath mode. The cmd file controls checker tame.
> start_checker_tame:-
>     tame_status(batch),
>     begin_selected_vc,
>     check_conclusion_exists,
>     begin_cmd,
>     main_checker_tame.
>
> %------------------------------------------------------------
>
> %Select the specified VC.
> begin_selected_vc:-
>     tame_vc_number(VC_NUMBER),
>
>     %Make it appear the user selected this VC number on the command
>     %line.
>     retractall(command_arg(_,_)),
>     assert(command_arg(vc_number, VC_NUMBER)),
>     execute_command(newvc),
>     !.
>
> %Error in selecting this VC.
> begin_selected_vc:-
>     tame_vc_number(VC_NUMBER),
>     write('ERROR: In selecting specified VC: '),
>     write(VC_NUMBER), nl,
>     halt.
>
> %------------------------------------------------------------
>
> check_conclusion_exists:-
>     tame_conc_number(CONC_NUMBER_INT),
>     conc(CONC_NUMBER_INT, _FORMULA),
>     !.
>
> check_conclusion_exists:-
>     tame_vc_number(VC_NUMBER),
>     tame_conc_number(CONC_NUMBER_INT),
>     write('ERROR: Specified VC: '),
>     write(VC_NUMBER), nl,
>     write('Does not contain specified conclusion: '),
>     write(CONC_NUMBER_INT), nl,
>     halt.
>
> %------------------------------------------------------------
>
> %This does not actually execute the cmd file. Rather it connects the
> %cmd file to the input stream.
> begin_cmd:-
>     tame_cmd_file(CMD_FILE),
>
>     %Make it appear the user selected this cmd file on the command
>     %line.
>     retractall(command_arg(_,_)),
>     assert(command_arg(filename, CMD_FILE)),
>     execute_command(execute),
>     !.
>
```

```
> begin_cmd:-
>     !.
>
> %------------------------------------------------------------
>
> %The main loop for checker tame.
> %Note that the user may still use the regular spade commands.  In
> %particular, the user may exit via 'exit', bypassing 'the checking and
> %reporting seen in 'tame_finish'.
> main_checker_tame:-
>     restore_temp_del_hyps,
>     repeat,
>     see_correct_input_stream,
>     nl,
>     write_check_prompt,
>     retractall(command_arg(_,_)),
>     read_user_command(COMMAND,ARGUMENTS),
>     check_command_arguments(COMMAND,ARGUMENTS),
>     execute_command(COMMAND),
>     (verified_exit_command(COMMAND) ; COMMAND=tame_finish),
>     dopop('1 -> pop_exit_ok'),
>     halt.
>
> %------------------------------------------------------------
>
> ensure_checker_tame:-
>     tame_active(yes),
>     !.
>
> ensure_checker_tame:-
>     write('ERROR: The tame commands are only available when using'),nl,
>     write('the SPADE Proof Checker in Checker Tame mode.'),nl,
>     fail.
>
> %------------------------------------------------------------
>
> tame_help:-
>     ensure_checker_tame,
>     do_tame_help,
>     !.
>
> do_tame_help:-
>     nl,
>     write('SPADE Proof Checker Tame commands: '),nl,
>     nl,
>     write('Commands that affect the goal:'),nl,
>     write('tame_subgoal_on_term, tame_subgoal_on_conc, tame_induction,'),nl,
>     write('tame_simp_and, tame_done, tame_all_done, tame_finish'),nl,
>     nl,
>     write('Commands that do not affect the goal:'),nl,
>     write('tame_echo, tame_show_depth, tame_compare_depth'), nl,
>     write('tame_help'), nl,
>     nl,
>     !.
>
> %------------------------------------------------------------
>
> tame_echo:-
>     ensure_checker_tame,
>     do_tame_echo,
>     !.
>
> %Do nothing, without raising an error.
> do_tame_echo:-
>     !.
>
> %------------------------------------------------------------
>
> tame_show_depth:-
>     ensure_checker_tame,
>     do_tame_show_depth,
>     !.
>
> do_tame_show_depth:-
>     case_pointer(CP),
>     write('SPADE proof depth is: '),
>     write(CP),
>     write('    '),
>     !.
>
> %------------------------------------------------------------
>
> tame_compare_depth:-
>     ensure_checker_tame,
>     do_tame_compare_depth,
>     !.
>
> do_tame_compare_depth:-
>     case_pointer(CP),
>     command_arg(tame_compare_depth__expected_depth, EXPECTEDDEPTH),
>     do_tame_compare_depth_x(CP, EXPECTEDDEPTH),
>     !.
>
> %Are the same.
> do_tame_compare_depth_x(EXPECTEDDEPTH, EXPECTEDDEPTH):-
>     write('SPADE depth ('),
>     write(EXPECTEDDEPTH),
>     write(') equals expected depth ('),
>     write(EXPECTEDDEPTH),
>     write(')'),
>     !.
>
> %Are different.
> do_tame_compare_depth_x(CP, EXPECTEDDEPTH):-
>     write('ERROR: SPADE depth ('),
>     write(CP),
>     write(') does not equal expected depth ('),
>     write(EXPECTEDDEPTH),
>     write(')'),nl,
>     !,
>     fail.
>
> %------------------------------------------------------------
>
> tame_done:-
>     ensure_checker_tame,
>     do_tame_done,
>     !.
>
> %Just call done.
> do_tame_done:-
>     done,
>     !.
>
> %------------------------------------------------------------
>
> tame_all_done:-
>     ensure_checker_tame,
>     do_tame_all_done,
>     !.
>
> %Just call all_done.
> do_tame_all_done:-
>     all_done,
>     !.
>
> %------------------------------------------------------------
>
> tame_subgoal_on_term:-
>     ensure_checker_tame,
>     do_tame_subgoal_on_term,
>     !.
>
> %This is a bit like using:
> %infer GOAL using inference(1).
> %
> %However, infer tries to be clever and prove the GOAL immediately,
> %leading to unpredictable behaviour. This always subgoals.
> do_tame_subgoal_on_term:-
>     %Retrieve conclusion.
>     command_arg(tame_subgoal_on_term__term, PROPERTY),
>
>     %Check no variables.
>     nonvar(PROPERTY),
>
>     %Do a type check.
>     checktype(PROPERTY, boolean),
>
>     %start_subgoal(FORMULA,CONDITIONS,HYPOTHESES,METHOD)
>     %Subgoal as if subgoaling from inference(1).
>     start_subgoal(PROPERTY,[PROPERTY],true,'SUBGOALING'),
>     !.
>
> %------------------------------------------------------------
>
> tame_subgoal_on_conc:-
>     ensure_checker_tame,
>     do_tame_subgoal_on_conc,
>     !.
>
> %This is a bit like using:
> %infer c#N using inference(1).
> %
> %However, infer tries to be clever and prove the term at this
> %conclusion number immediately, leading to unpredictable behaviour.
> %This always subgoals.
> do_tame_subgoal_on_conc:-
>     %Retrieve conclusion.
>     command_arg(tame_subgoal_on_conc__conc, CONCNUMBER),
>
>     %Check no vairables.
>     nonvar(CONCNUMBER),
>     %Check integer.
>     integer(CONCNUMBER),
>
>     %Retrieve conclusion.
>     conc(CONCNUMBER,CONCLUSION),
>
>     %start_subgoal(FORMULA,CONDITIONS,HYPOTHESES,METHOD)
>     %Subgoal as if subgoaling from inference(1).
>     start_subgoal(CONCLUSION,[CONCLUSION],true,'SUBGOALING'),
>     !.
>
> %------------------------------------------------------------
>
> tame_induction:-
>     ensure_checker_tame,
>     do_tame_induction,
>     !.
>
> %Very similar to spade induction, however forces the user to provide
> %the names of the generated variable. (Soundness is ensured by
> %ensuring that these names are unique)
> %
> %This is done because the spade auto-names are difficult to emulate at
> %planning -- the spade names are based on the global number of
> %inductions in order of execution. The planner only knows the global
> %number of inductions once a concrete plan is found (which is too
> %late, as names are required immediately).
> do_tame_induction:-
>     write('INDUCTION -- on conclusion number? '),
>     rread(CONCNO),
>     write('INDUCTION -- on what? '),
>     rread(INDVAR),
>     write('INDUCTION -- Base case? '),
>     rread(BASECASE),
>     write('INDUCTION -- Induction variable name? '),
>     rread(INDVARNAME),
>     write('INDUCTION -- Quantified variable name? '),
>     rread(QUANTVARNAME),
>
>     addVariableToSpade(uvar, integer, INDVARNAME),
>     addVariableToSpade(uvar, integer, QUANTVARNAME),
>
>     novars(INDVAR>=BASECASE),
>     infer(INDVAR>=BASECASE),
>     induction(c#CONCNO, INDVAR, BASECASE, INDVARNAME, QUANTVARNAME),
>     !.
>
> addVariableToSpade(VARCLASS, TYPE, NEWVAR):-
>     nondeclared(NEWVAR),
>     find_core_type(TYPE, CORE_TYPE),
>     assertz(var_const(NEWVAR, CORE_TYPE, p)),
>     CLASSDEC=..[VARCLASS,NEWVAR],
```

```
>     assertz(CLASSDEC),
>     !.
>
>
> /*** INDUCTION(FORMULA,VAR,BASECASE) -- proof by induction ***/
> induction(F,VAR,BASE, INDVAR, QUANTVAR) :-
>         (
>            F=c#N,
>            integer(N),
>            conc(N,FORMULA)
>         ;
>            checktype(F,boolean),
>            FORMULA=F
>         ),
>         subst_vbl(VAR,BASE,FORMULA,BASECASE),
>         !,
>         subst_vbl(VAR,QUANTVAR,FORMULA,ASSUME),
>         subst_vbl(VAR,INDVAR+1,FORMULA,PROVE),
>         write('ENTERING PROOF BY INDUCTION ATTEMPT'),
>         INDCASE=(INDVAR>=BASE ->
>                   (for_all(QUANTVAR:integer,
>                     (BASE<=QUANTVAR and QUANTVAR<=INDVAR) ->
>                       ASSUME) -> PROVE)),
>         nl,
>         assertz(logfact(induction, [FORMULA, VAR, BASE])),
>         start_subgoal(FORMULA,[BASECASE,INDCASE],true,'INDUCTION'),
>         !.
>
> %------------------------------------------------------------
>
> tame_simp_and:-
>     ensure_checker_tame,
>     do_tame_simp_and,
>     !.
>
> %Specialised simplification, reducing conjunctions of hypotheses
> %to several individual hypotheses. This is a subset of the more
> %powerful spade simplify command.
> do_tame_simp_and:-
>     split_hyps,
>     !.
>
> %This is really, really, really messy.
> %The spade infer command tries to infer a term using a rule and
> %existing hypotheses. However, infer searches for the relevant
> %hypotheses, rather than being explicitly told which hypotheses to
> %use.  This search brakes the fundamental idea of checker tame being
> %under total control. Worse, this search can break plans, as infer can
> %get hopelessly lost in this search.
> %
> %***This really needs addressed once the desired behaviour of infer is
> %better understood.***
> %
> %***Note this simplification is only needed because of the poor
> %interface to infer. Addressing this issues should involve deleting
> %all of this code***
> %
> %As a temporary fix, the hypotheses were simply reordered. This
> %reordering tends to make infer find newer hypotheses first, and this
> %tends to work due to the nature of the plans. But, of course, this is
> %very unstable.
> do_tame_simp_and:-
>     order_hyps_first_to_last,
>     !.
>
> do_tame_simp_and:-
>     !.
>
> %If a simplification is being performed then it is likely that the
> %interesting hyps have recently been introduced, so they are placed at
> %the top.  The key parts of this code are taken from the original
> %simplify, which performed this reordering as a side effect of
> %movenots.
> order_hyps_first_to_last:-
>     retractall(hn(_)),
>     assertz(hn(1)),
>     repeat,
>     hn(N),
>     movenots_x(N),
>     !.
>
> %Found hyp.
> movenots_x(N):-
>     gethyp(N,H),
>     assertz(hyp(N,H)),
>     M is N+1,
>     retract(hn(N)),
>     assertz(hn(M)),
>     fail.
>
> %Not found hyp.
> movenots_x(N):-
>     retract(hn(N)),
>     !.
>
> %Do: callpro. writeouthyps.
> %To see what order spade will look at the hyps.
> writeouthyps:-
>     hyp(N, _),
>     write(N),
>     write(' '),
>     fail.
>
> writeouthyps:-
>     !.
>
> %------------------------------------------------------------
>
> tame_finish:-
>     ensure_checker_tame,
>     do_tame_finish,
>     !.
>
> do_tame_finish:-
>
>     %Check correct vc is being considered.
>     check_vc_match,
>
>     %Check that the proof depth is zero. This ensures that the original
>     %conclusions are being observed.
>     check_zero_proof_depth,
>
>     %Check that the conclusion under consideration no longer exists.
>     check_conclusion_absent,
>
>     write('IS PROVED'),nl,
>     !.
>
> do_tame_finish:-
>     write('IS **NOT** PROVED'),nl,
>     !.
>
> check_vc_match:-
>     tame_vc_number(VC_NUMBER_INT),
>     current_vc(_NAME, VC_NUMBER_INT),
>     !.
>
> check_vc_match:-
>     tame_vc_number(VC_NUMBER_INT),
>     current_vc(_NAME, CURRENT_VC_NUMBER_INT),
>     write('VC to prove: '),
>     write(VC_NUMBER_INT), nl,
>     write('Does not match with VC currently loaded: '),
>     write(CURRENT_VC_NUMBER_INT), nl,
>     !,
>     fail.
>
> check_zero_proof_depth:-
>     case_pointer(0),
>     !.
>
> check_zero_proof_depth:-
>     case_pointer(CP),
>     write('At finish, proof depth should be: 0'), nl,
>     write('(Currently the proof depth is: '),
>     write(CP), nl,
>     !,
>     fail.
>
> check_conclusion_absent:-
>     tame_conc_number(CONC_NUMBER_INT),
>     \+ conc(CONC_NUMBER_INT, _FORMULA),
>     !.
>
> check_conclusion_absent:-
>     tame_conc_number(CONC_NUMBER_INT),
>     write('Conclusion to prove: '),
>     write(CONC_NUMBER_INT), nl,
>     write('Still remains unproven!'), nl,
>     !,
>     fail.
>
> %------------------------------------------------------------
>
>
```

A.47 toplevel.pl

```
71c71,76
<
---
> write_check_prompt :-
>     tame_active(yes),
>     seeing(user),
>     write('CHECK-TAME'),
>     output_newline_if_necessary,
>     !.
209a215,227
> match_command([t,a,m,e,_,s,u,b,g,o,a,l,_,o,n,_,t,e,r,m], tame_subgoal_on_term) :- !.
> match_command([t,a,m,e,_,s,u,b,g,o,a,l,_,o,n,_,c,o,n,c], tame_subgoal_on_conc) :- !.
> match_command([t,a,m,e,_,i,n,d,u,c,t,i,o,n], tame_induction) :- !.
> match_command([t,a,m,e,_,s,i,m,p,_,a,n,d], tame_simp_and) :- !.
> match_command([t,a,m,e,_,d,o,n,e], tame_done):- !.
> match_command([t,a,m,e,_,a,l,l,_,d,o,n,e], tame_all_done):- !.
> match_command([t,a,m,e,_,f,i,n,i,s,h], tame_finish):- !.
>
> match_command([t,a,m,e,_,e,c,h,o], tame_echo) :- !.
> match_command([t,a,m,e,_,s,h,o,w,_,d,e,p,t,h], tame_show_depth) :- !.
> match_command([t,a,m,e,_,c,o,m,p,a,r,e,_,d,e,p,t,h], tame_compare_depth) :- !.
> match_command([t,a,m,e,_,h,e,l,p], tame_help):- !.
>
677a696,749
>
> %Retrieve the property to subgoal from the command line.
> parse_command_arguments(tame_subgoal_on_term, TERM):-
>         nonvar(TERM),
>         assertz(command_arg(tame_subgoal_on_term__term, TERM)),
>         !.
>
> %Retrieve the conclusion number to subgoal from the command line.
> parse_command_arguments(tame_subgoal_on_conc, CONCNUMBER):-
>         nonvar(CONCNUMBER),
>         integer(CONCNUMBER),
>         assertz(command_arg(tame_subgoal_on_conc__conc,CONCNUMBER)),
>         !.
>
> %All induction arguments are requested through interaction.
> parse_command_arguments(tame_induction, _):-
>     !.
>
> %No argumants for simplify hyp conjuncts.
> parse_command_arguments(tame_simp_and, _):-
>     !.
>
> %The tame version of done.
> parse_command_arguments(tame_done, _):-
>         !.
>
> %Close a subgoal or VC proof.
> %(Previously 'done' also did this.)
> parse_command_arguments(tame_all_done, _):-
>         !.
>
```

```
> %Ignore any arguments to tame finish.
> parse_command_arguments(tame_finish,_):-
>         !.
>
> %Ignore any arguments to echo.
> parse_command_arguments(tame_echo,_):-
>         !.
>
> %Ignore any arguments to tame_show_depth.
> parse_command_arguments(tame_show_depth,_):-
>     !.
>
> %Retrieve the expected depth from the command line.
> parse_command_arguments(tame_compare_depth, EXPECTEDDEPTH):-
>         nonvar(EXPECTEDDEPTH),
>         integer(EXPECTEDDEPTH),
>         assertz(command_arg(tame_compare_depth__expected_depth,EXPECTEDDEPTH)),
>         !.
>
> %Ignore any arguments to tame_help.
> parse_command_arguments(tame_help,_):-
>     !.
>
876a949,962
>
> %Make a really loud failure message for checker tame.
> execute_command(C) :-
>         tame_active(yes),
>         nl, nl, nl, nl,
>         write('     **************************************************     '), nl,
>         write('     *                                                *     '), nl,
>         write('     * FAIL! - Proof Command Failure in Checker Tame! *     '), nl,
>         write('     *                                                *     '), nl,
>         write('     **************************************************     '), nl,
>         nl, nl, nl, nl,
>         retractall(logfact(_,_)),
>         !,
>         fail.
884d969
<
1201c1286
<             command_arg(FILE)                                        /* CFR017 */
---
>             command_arg(filename, FILE)
```

A.48 traverse.pl

A.49 typecheck5.pl

A.50 utilities.pl

```
175c175
< add_new_hyp(H,1) :- hyp(_,H), !.
---
> %add_new_hyp(H,1) :- \+ tame_active(yes), hyp(_,H), !.
473a474,478
> trivial_command(tame_echo).
> trivial_command(tame_show_depth).
> trivial_command(tame_compare_depth).
> trivial_command(tame_help).
>
```