CS-Note* 18

Bill J. Ellis
September 11, 2002

Documenting SIMPLIFIER Changes (For FRiENDLY)

Contents

1 Introduction

This note serves to document the changes made to the SIMPLIFIER for the NuSPADE project, in particulr for FRiENDLY (See CS-note 17). In the long term these changes should be integrated into the distribution SIMPLIFIER. The changes have been undertaken following the postcondition guidelines outlined in CS-note 16 for making changes to the SPARK tools. The need for these changes, and how the precondition guidelines were met, is discussed in CS-note 17.

2 Describing Flag `/friendly`

2.1 Human Readable Log (`NAME.slg`)

As default the SIMPLIFIER stores a human readable log (`NAME.slg`) to record its actions. Internally this is supported by asserting a number of predicates of the form:

```
log_fact(ACTION_ATOM, ARGUMENTS_ITEM)
```

These predicates are removed before considering each VC, thus internally it is implicitly known which VC the `log_fact/2` predicates are describing. Further, the order in which these predicates are asserted is important. The predicates should be described in the `NAME.slg` log in the order they were asserted. This ordering is achieved by using the underlying Prolog search, which considers the oldest asserted predicate first.

2.2 Machine Readable Log (`NAME.slg`)

The machine readable log (`NAME.slg`) will contain a number of Prolog quoted predicates all of the form:

```
mlg_log(CURRENT_VC_INTEGER, ORDER_INTEGER, ACTION_ATOM, ARGUMENTS_ITEM)
```

- `CURRENT_VC_INTEGER -` Explicit number of VC being tackled. These may be disjoint and need not be in any particular order. Duplicate VC numbers are not expected, however may potentially occur.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

- `ORDER_INTEGER` - Explicit ordering of the log information. This is reset to 1 for the first log of each VC, increasing by 1 for each subsequent log.
- `ACTION_ATOM` - The kind of action taken on this VC. These are taken from the first argument of `log_fact/2`.
- `ARGUMENTS_ITEM` - A structure describing in detail the action taken. These are taken from the second argument of `log_fact/2`.

2.3 Relevant Flags

- **/friendly -** Where the `/friendly` flag is specified the `NAME.mlg` log will be stored. The value of `NAME` will be same as would used for the `NAME.slg` log.
- **Multiple /friendly -** Like other flags in the SIMPLIFIER, the `/friendly` flag can only be specified once. If it is specified more than once, the program should terminate and display a message describing the error.
- **/nolog -** It is possible to switch off the `NAME.slg` log using `/nolog`. Even where this is the case the `/friendly` flag may still be provided to store the `NAME.mlg` log. Note that this is possible as the `log_fact/2` predicates are always asserted, regardless of whether or not they will eventually be used.
- **/log=NEW_NAME -** It is possible to change the name used for the `NAME.slg` log with `/log=NEW_NAME`. In this case the name for the `NAME.mlg` log will also change to `NEW_NAME.slg`.

2.4 Documentation

If the `/friendly` flag is ever documented in the SIMPLIFIER manual the following might be used.

2.5 Describing the Qualifier

The "/friendly" qualifier may be used to tell the simplifier to generate a NAME.MLG machine readable log. The default is that this log should not be generated. The NAME used will the same as that for the NAME.SLG human readable log.

2.6 Describing the Log

The NAME.MLG machine readable log describes the same information in the NAME.SLG human readable log, only in a form intended for automated analysis. The log contains a number of entries of the form:

```
mlg_log(CURRENT_VC_INTEGER, ORDER_INTEGER, ACTION_ATOM, ARGUMENTS_ITEM)
```

Where `CURRENT_VC_INTEGER` is the number of a VC, `ORDER_INTEGER` indicates the order (incremented from 1 for each VC) in which simplification actions where taken and together `ACTION_ATOM` and `ARGUMENTS_ITEM` describe a single simplification action.

3 Implementation

The following describes the implementation of the `/friendly` flag and how to add it to version 2p04 of the SIMPLIFIER.

3.1 Accept the `/friendly` Flag

Accept the flag `/friendly`. If this flag has already been set it writes an error before exiting the program.

Add this code to file `getdcldat.pl`, just before the predicate `process_dcl_qualifier(Q)` on line 342.

```
%##############################################################
%process_dcl_qualifier/1
%
%Extra flag "/friendly" to switch on features that assist
%in integrating the Simplifier with the NuSpade project.
%
%This flag switches on the following extra behaviour:
%
%+Alongside the simplifier human readable log file (slg),
% store a machine readable log file (mlg).

process_dcl_qualifier([47, 102, 114, 105, 101, 110, 100, 108, 121]):-
        (
           %Check if flag has already been specified.
           nuspade_friendly,
           !,
           scream('/friendly qualifier appears more than once on command-line')
        ;
           true
        ),
        !,
        %Set nuspade friendly mode.
        assert(nuspade_friendly),
        !.
%##############################################################
```

3.2 Call Extra Code Where /friendly Flag is Set

At suitable points the /friendly flag is checked and suitable predicates are called.

3.2.1 Writing to the NAME.mlg Log

Where the /friendly flag is set, write out the next batch of entries to the NAME.mlg log.

Add this code to file makelog.pl*, just after the* :- *of the first occurrence of predicate* write_log_facts *on line 131.*

```
        (
              nuspade_friendly,
              write_vc_to_mlg
            ;
              true
        ),
```

3.2.2 Close the NAME.mlg Log

Where the /friendly flag is set, close the NAME.mlg log.

Add this code to file makelog.pl*, just after the* :- *of the first occurrence of predicate* close_log_file *on line 854.*

```
        (
              nuspade_friendly,
              close_mlg_log_file
            ;
              true
        ),
```

3.3 Manage the NAME.mlg Log

Perform the opening of, writing to and closing of the NAME.mlg log.
Add this code to the end of the file makelog.pl.

```
%#############################################################
%write_vc_to_mlg/0
%
%Write the next VC details to the mlg file.

write_vc_to_mlg:-
    %Keep the current output stream for recovery later.
    telling(CURRENT_OUTPUT_STREAM_ATOM),

    %Switch to the mlg output stream, creating it if necessary.
    get_mlg_name_dot_ext(MLG_NAME_DOT_EXT_CODE_LIST),
    tell(MLG_NAME_DOT_EXT_CODE_LIST),

    %Determine the number of the current VC. This makes heavy
    %use of existing Simplifier code.
    %VC_NAME_ATOM is of the form: procedure_partition_section_1.
    vc_name(VC_NAME_ATOM),
    name(VC_NAME_ATOM, VC_NAME_CODE_LIST),
    triple_append(_, "_", DIGITS_AND_DOT_CODE_LIST, VC_NAME_CODE_LIST),
    %Force the finding of the last '_'.
    is_digits_and_dot(DIGITS_AND_DOT_CODE_LIST),
    gen_append(DIGITS_CODE_LIST, ".", DIGITS_AND_DOT_CODE_LIST),
    name(DIGITS_ATOM, DIGITS_CODE_LIST),

    %Reset the order number for each VC.
    reset_order_number,

    %Write out all of the log_fact predicates for this VC.
    write_facts_to_mlg(DIGITS_ATOM),

    %Return to the stream on entry.
    tell(CURRENT_OUTPUT_STREAM_ATOM),
    !.


%resetOrderNumber/0
%
%Set the order number to zero. This must be called prior
%to next_order_number/1.

reset_order_number:-
    %Use retractall/1 rather than retract/1 as it is always
    %successful. Thus this predicate can also be used to
    %initialise the first order number and reset future
    %order numbers.
    retractall(order_number(_)),
    assert(order_number(0)),
    !.


%next_order_number/1
%
%Return the next order number.
```

```
next_order_number(NEXT_ORDER_INTEGER):-
    retract(order_number(NOW_INTEGER)),
    NEXT_ORDER_INTEGER is NOW_INTEGER+1,
    assert(order_number(NEXT_ORDER_INTEGER)),
    !.


%write_facts_to_mlg/1
%
%Write out the mlg facts to the current output stream,
%in a Prolog-quoted form that can be directly read back
%in by Prolog.

write_facts_to_mlg(CURRENT_VC_INTEGER):-
    %Use a failure-driven loop to iterate over all of the
    %log_fact/2 predicates. Note that these must not be
    %retracted here as they are required later.
    log_fact(ACTION_ATOM, ARGUMENTS_ITEM),

    %Get the order number.
    next_order_number(NEXT_ORDER_INTEGER),

    writeq(mlg_log(CURRENT_VC_INTEGER, NEXT_ORDER_INTEGER,
                   ACTION_ATOM, ARGUMENTS_ITEM)),
    %Add the terminating full stop. Note that this must not
    %use writeq, otherwise it is quoted, resulting in an
    %unparseable: predicate(...)' .'
    write(' .'),
    nl,
    fail.

write_facts_to_mlg(_):-
    nl,
    nl,
    true.


%close_mlg_log_file/0
%
%Close the mlg log file.

close_mlg_log_file:-
        telling(CURRENT_OUTPUT_STREAM_ATOM),

        %Switch to the mlg output stream, creating it if necessary.
        get_mlg_name_dot_ext(MLG_NAME_DOT_EXT_CODE_LIST),
        tell(MLG_NAME_DOT_EXT_CODE_LIST),

        %Close the current output stream.
        told,

        %Return to the stream on entry.
        tell(CURRENT_OUTPUT_STREAM_ATOM),
        !.


%get_mlg_name_dot_ext/1
%
%Return the name for the machine readable log.
```

```
%This is the same as for the NAME.lsg log changing
%the extension from lsg to mlg to give NAME.mlg.

get_mlg_name_dot_ext(MLG_NAME_DOT_EXT_ATOM):-
    %Find slg log as list.
    logfile_name(LSG_NAME_DOT_EXT_ATOM),
    name(LSG_NAME_DOT_EXT_ATOM, LSG_NAME_DOT_EXT_CODE_LIST),

    %Strip off the slg extension.
    append(LSG_NAME_DOT_CODE_LIST, "slg", LSG_NAME_DOT_EXT_CODE_LIST),

    %Put on the mlg extension.
    append(LSG_NAME_DOT_CODE_LIST, "mlg", MLG_NAME_DOT_EXT_CODE_LIST),

    %Return result as an atom.
    name(MLG_NAME_DOT_EXT_ATOM, MLG_NAME_DOT_EXT_CODE_LIST),
    !.
%############################################################
```