Online Appendix to: Simplify: A Theorem Prover for Program Checking

GREG NELSON AND JAMES B. SAXE

Hewlett-Packard

This short document describes two automatic theorem-prover test suites: the "frontend test suite" and the "small test suite". These tests were used to study performance of the automatic theorem prover Simplify [Detlefs et al. 2005, Sec. 9]. The files in the test suites are in the input format of the Simplify theorem prover [Detlefs et al. 2005]. This document aims to describe the syntax and semantics of the test suite files at a sufficient level of detail to allow readers to translate them into forms suitable for input to other provers.

The frontend test suite consists of 207 files containing 2331 valid verification conditions generated by applying the Extended Static Checker for Java (ESC/Java) [Flanagan et al. 2002] to its own front end.

The small test suite consists of 18 files, each containing a single valid conjecture. Five of these are verification conditions generated by ESC/Modula-3 [Detlefs et al. 1998] (addh1.sx, cat.sx, fastclose.sx, frd-seek.sx, and simplex.sx). Two are artificial tests (domino6x4x.sx and domino6x6x.sx) that reduce the well-known problem of tiling a mutilated checkerboard with 1-by-2 dominos into test cases that exercise case splitting and reasoning about equality. The remaining eleven are verification conditions taken from the frontend test suite.

This document is about the test suites, not about Simplify. We intentionally omit descriptions of several features of Simplify's input language that are not used in the test suites. For the features of Simplify that we do describe, we generally ignore nuances that are not relevant for the test suites. When we mention that some expression is logically equivalent to some other expression, we do not address the pragmatic impact on Simplify's efficiency or the efficacy of using one expression in place of the other.

Each file contains a sequence of S-expressions. Each expression is either a formula to be proved or a directive that modifies the context in which subsequent proofs are carried out.

S-Expressions

An S-expression is either leaf symbol or a parenthesized list of S-expressions.

A leaf symbol is an identifier or a literal decimal integer constant (optionally including a sign).

There are three forms of identifiers. The first form is a sequence of letters, digits, and underscores, beginning with a letter. The second form is a sequence of characters from the following set:

```
! # $ % & * + - . / : < = > ? @ [ ] ^ _ { } ~
```

The third form is a sequence of ordinary characters or escape sequences, surrounded by vertical bars. An ordinary character is any printing character (including the space character) except "|" or "\". We don't describe the permitted escape sequences here; none of them occur in the test suites. The vertical bars in the third form of identifier delimit the identifier, but are not part of it. Thus the identifiers "abc" and "|abc|" are considered identical.

A semicolon, other than one occurring in an identifier delimited by vertical bars, begins a comment which continues until the end of the line. Comments are equivalent to white space.

Formulas

Simplify draws a strict distinction between formulas and terms. The identifiers TRUE and FALSE are formulas. The formula

(FORALL (id_1 ... id_n) (PATS ...) F)

is true if the formula F is true for any values of the identifiers id_1 ... id_n. The subexpression "(PATS ...)" is an optional clause that is of heuristic significance only. We omit the description of its internal syntax.

The formula

(EXISTS (id_1 ... id_n) (PATS ...) F)

is true if the formula F is true for some values of the identifiers id_1 ... id_n. The subexpression "(PATS ...)" is an optional clause that is of heuristic significance only. We omit the description of its internal syntax.

The built-in boolean connectives

AND OR NOT IMPLIES IFF EXPLIES

take formulas as arguments and produce formulas. The formula

(EXPLIES Q P)

is equivalent to (IMPLIES P Q). AND and OR may take any number of arguments.

The built-in relations

EQ NEQ < <= > >= DISTINCT

take terms as arguments and produce formulas. The formula

(DISTINCT t_1 ... t_n)

where the t's are terms, asserts that the t's are distinct. Arguments to arithmetic relations are implicitly integral, regardless of whether the relation holds. For example, Simplify considers the conjecture (OR (<= x 3) (>= x 4)) to be valid.

Formulas of the forms

(LBLPOS L F)
(LBLNEG L F)
(LBL L F),

where L is an identifier and F is a formula, are logically equivalent to F.

If F and G are binary quasi-relations (description follows), then the formula

(ORDER F G)

asserts that F and G are the irreflexive and reflexive versions of some partial order, respectively. There are no occurences of ORDER in the frontend test suite.

Terms

Integer literals are terms. The built-in functions

$$+ \quad - \quad * \quad \text{select store}$$

take terms as arguments and produce terms. The functions "select" and "store" are predefined to satisfy the following axioms:

```
(FORALL (a  i  x)
        (EQ  (select (store a i x) i)
             x)

(FORALL (a i j x)
        (OR  (EQ i j)
             (EQ (select  (store a i x ) j)
                 (select a j) ) ) ).
```

If an identifier that is not predefined is used as a term, it denotes an arbitrary value. If an identifier that is not predefined is used as a formula, then it denotes a unknown Boolean value. If an identifier that is not predefined is usen as a function symbol in a term, it denotes an uninterpreted function. There is no way for a user to introduce new relation symbols, but the directives DEFPRED and DEFPREDMAP, described in the following, offer a similar capability.

Pushing and Popping

The directive

```
(BG_PUSH F_1 ... F_n),
```

where the F's are formulas, adds the F's to the background predicate where they are implicit assumptions to be used in future proofs.

The directive

```
(BG_POP)
```

restores the background predicate to its state prior to the matching BG_PUSH.

DEFPRED and DEFPREDMAP

Simplify makes a strict distinction between formulas and terms. Because this is sometimes inconvenient, Simplify allows the directive

```
(DEFPRED (R x_1 ... x_n)),
```

where R and the x's are identifiers which declares R to be a "quasi-relation" that takes n arguments. This means that R is a function symbol but that a term of the form

```
(R t_1 ... t_n)
```

is allowed to appear in a position where a formula is expected, in which case, it will be desugared to

```
(EQ |@true| (R t_1 ··· t_n)).
```

That is, the constant "|@true|" is used to model the propositional value TRUE in the term space.

The DEFPRED directive has a second version:

```
(DEFPRED (R x_1 ··· x_n) F),
```

where F is a formula. In addition to declaring R to be a quasi-relation with n arguments, this version of the directive adds the assumption

```
(FORALL (x_1 ··· x_n)
        (IFF (EQ  (R x_1 ··· x_n |@true | )
             F))
```

to the background predicate

The directive

```
(DEFPREDMAP (R x_1 . . . x_n) i)
```

where R, the x's, and i are identifiers, allows a term of the form

```
(select (R t_1 . . . t_n) ind)
```

to appear where a formula is expected, in which case, it will be desugared to

```
(EQ (select (R t_1 . . . t_n) ind) |@true|).
```

The DEFPREDMAP directive has a second version:

```
(DEFPREDMAP (R x_1 . . . x_n) i F),
```

where F is a formula. This has the additional effect of adding the assumption

```
(FORALL (x_l ... x_n i)
        (IFF (EQ (select (R x_l . . . x_n) i) |@true|)
             F))
```

to the background predicate.

There are no occurrences of the DEFPREDMAP directrive in the frontend test suite.

REFERENCES

DAVID, L., DETLEFS, D. L., LEINO, K. R. M., NELSON, G., AND SAXE, J. B. 1998. Extended static checking. Compaq Systems Research Center, Research Rep. 159. Available at `http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-159.html` or `http://new-az.hpl.hp.com/techpubs/Compaq-DEC/SRC-RR-159.html`.

DETLEFS, D., NELSON, G., AND SAXE, J. B. 2005. Simplify: A theorem prover for program checking. *JACM* 52, 3, 000–000.

FLANAGAN, C., LEINO, K. R. M., LILLIBRIDGE, M., NELSON, G., SAXE, J. B., AND STATA, R. 2002. Extended static checking for Java. *Conference on Programming Language Design and Implementation*. 234–245.