Sibyl 2.3 User Guide

Marta Cialdea Mayer

April 2017

1 Calling the prover

Sybil can be called from the command line with the syntax

`sibyl [options] <file-name>`

The available options are the following (and are listed when calling Sibyl with the option `-help`):

`-v <int>` (verbosity level): if the value is 0, then only the final output and the elapsed time are printed. If the value is 1 (default), also information on the number of expanded branches and rule applications are given in output. The value 2 causes the output of a subset of the nodes of the complete and open branch found by the prover, if any: nodes labelled by either blockable formulae (possibly annotated by their blocking status), or relational formulae or formulae of the form $a{:}\,p$ or $a{:}\,\neg p$ for $p \in \mathsf{PROP}$ are printed out. Such nodes can be used to compute a (possibly infinite) model represented by the branch. Values 3 and 4 give more information on tableau construction, printing messages every time a node is added to the explored branch, or a branch closes. Finally, if a complete and open branch is found, it is printed out. The assertions in the initial tableau, however, are not shown.

`-t <int>` (timeout): sets the timeout to the given value (in seconds). The default is 300 secs. If the value is 0, then sybil runs with no timeout.

`-bjoff` turns backjumping off.

`-sboff` turns semantic branching off.

`-tex` : with this option, a LaTeX file is generated, with the same name of the input file and extension `.tex`, containing a sequence of all the explored branches (either closed or open and complete), what changes when the equality rule is applied and a description of the offspring relation in each branch. In tableau branches built by Sibyl, nodes are not necessarily numbered consecutively, but there may be gaps between a node number and the next one. In the LaTeX file produced with this option, however, nodes are renumbered consecutively.

`-texn` : a LaTeX file is generated like with the option `-tex`, but nodes are not renumbered. This option may be useful when one wishes to compare the LaTeX file with the step-by-step construction of branches that can be monitored with verbosity levels greater than 2.

The expansion strategy can be specified in a file, called either `sibylrc` or `.sibylrc` in the same directory of the input file. Expansion rules are identified by the following keywords: SUB (the equality rule), AND (the $\wedge$ rule), OR (the $\vee$ rule), AT (the @ rule), BINDER (the $\downarrow$ rule), BOX (the $\Box$ rule), DIAMOND (the $\Diamond$ rule), A (the A rule), E (the E rule), TRANS (the Trans rule) and LINK (the Link rule). Moreover, the keywords PROP, NOT_PROP and NOT_NOM identify the closure rules considering nodes of the form $a : p$, $a : \neg p$ (for $p \in \mathsf{PROP}$) and $a : \neg b$ (for $b \in \mathsf{NOM}$), respectively.

The rules written in the n-th line of the file `sibylrc` (or `.sibylrc`) are associated the priority value n (the lower a rule's value, the higher its priority). The default rule priorities (used in the absence of both `sibylrc` and `.sibylrc`, are the following (the rules are listed in increasing priority values):

```
SUB NOT_NOM
PROP NOT_PROP
AT
A
AND
DOWN
BOX TRANS LINK
DIAMOND E
OR
```

Remark. In the tableau calculus underlying sibyl, nominal equalities are treated by substitution: the expansion of $a{:}\, b$, with $a \neq b$, replaces every formula F occurring in the branch with $F[b/a]$ (i.e. b replaces a in every node label). However, Sibyl treats nominal equalities so that nominals occurring in the input file are possibly preserved: if a occurs in the input formulae and b does not, then b is replaced by a, and not vice versa.

2 Syntax of the Input File

The input file contains:

1. The declaration of transitive relations (optional).
2. The declaration of relation inclusion assertions (optional).
3. The formula section, where the formulae to be tested for satisfiability are specified.

Comments can be included either between the characters `\*` and `*/` or between the symbol `#` and the end of the line.

2.1 Transitivity and Inclusion Assertions

A *relation name* begins with an alpha-numeric character, possibly followed by a sequence of alphanumeric characters and the underscore symbol.

The declaration of transitive relations is introduced by the keyord `Transitive:`, followed by a sequence of relation names (separated by spaces, tabs or new lines), and concludes with a period:

```
Transitive:  <relation-name>+.
```

Relation names are identifiers beginning with any alpha-numeric character (excluding `A` and `E`, which are reserved for the global modalities), possibly followed by other alpha-numeric characters and the underscore symbol. For instance:

```
Transitive: R3 father.
```

The declaration of relation inclusion assertions is introduced by the keyord `Inclusions:`, followed by a sequence of *inclusion assertions* (separated by spaces, tabs or new lines), and concludes with a period:

```
Inclusions:  <inclusion-assertion>+ .
```

Every inclusion assertion is a pair constituted by a *relation* and a relation name, enclosed in brackets and separated by a comma:

```
    (<relation> , <relation-name>)
```

A relation is either a relation name or a relation name preceded or followed by a dash (indicating the converse relation). For instance:

```
Inclusions: (R10,R6) (R9-,R9) (-father,child) (-sibling,sibling).
```

2.2 The Formula Section

The list of formulae in the formula section is enclosed between the keywords `begin` and `end`. Formulae are separated by either a comma or a semicolon:

```
begin
    <formula> ; <formula> ; ... ; <formula>
end
```

Identifiers for nominals, variables, relation names and propositional letters are sequences of alpha-numeric characters and the underscore symbol, subject to the following restrictions: a propositional letter begins with an uppercase letter; a nominal begins with either a lowercase letter or a numeric character; variables and relation names begin with any alpha-numeric character. Obviously, no identifier must contain any keyword (i.e. `begin, end, Transitive, Inclusions, true, false, binder, All, Some`).

Non atomic formulae are written as shown in Table 1 (keywords are case sensitive). Brackets are used in the usual way: unary operators have a higher priority than binary ones, and the priority order of binary operators is the following: $\wedge$, $\vee$, $\rightarrow$, $\equiv$. They are all left associative.

2.3 Examples

The following text is the content of a simple input file, with an unsatisfiable set of assertions and formulae.

$\top$	`true`
$\bot$	`false`
$\neg F$	`- F`
$F \wedge G$	`F & G`
$F \vee G$	`F | G`
$F \rightarrow G$	`F -> G`
$F \equiv G$	`F <-> G`
$t{:}\,F$	`t:F`
$\downarrow x.F$	`binder x.F`
$\Box_R F$	`[R] F`
$\Diamond_R F$	`<R> F`
$\Box_{R^-} F$	either `[-R] F` or `[R-]F`
$\Diamond_{R^-} F$	either `<-R> F` or `<R->F`
$\Box^n_R F$	`[R,n] F`
$\Diamond^n_R F$	`<R,n> F`
$\Box^n_{R^-} F$	either `[-R,n] F` or `[R-,n]F`
$\Diamond^n_{R^-} F$	either `<-R,n> F` or `<R-,n>F`
$\mathsf{A}F$	`All F`
$\mathsf{E}F$	`Some F`

Table 1: Syntax of non atomic formulae, where R is any relation name and n a non-negative integer

```
Transitive: ancestor.
Inclusions: (child-,father) (father-,child) (sibling-,sibling)
(father,ancestor).

begin
# tony is bill's father and mary is one of tony's children
tony:(<father> bill & mary:<child>tony);
# bill is a sibling of mary's
mary:<sibling> bill;
# everybody has a father
All binder x. Some <child-> x ;
/* either mary, tony and bill have no ancestor in common
   or mary is not a sibling of bill's */
- Some (<ancestor> mary & <ancestor>tony & <ancestor>bill)
     | bill:-<sibling>mary
end
```

It is worth pointing out that the proof procedure on which Sibyl is based is proved to be terminating and complete provided that the negation normal form F of each formula in the input file respects the following conditions:

1. F contains no universal operator scoping over a binder, that in turn has scope over a universal modality.

2. no subformula $\Box^n_R G$ of F occurs in the scope of any universal modality;

3. no subformula $\Diamond_R^n G$ of F is both in the scope and contains in its scope a universal modality.

If this is not the case, Sibyl warns that the proof procedure may not terminate or give an incorrect result. In principle, in fact, it might be the case that the input problem is unsatisfiable, and yet Sibyl does not find a closed tableau and answers "satisfiable".

A simple example of an input file "outside the decidable fragment" is presented below. The negation normal form of the last formula is in fact:

$$\mathsf{A}(\Box_{ancestor}\neg mary \vee \downarrow x.\Box_{ancestor}\neg x)$$

```
Transitive: ancestor.
Inclusions: (child-, father) (father-,child)
(father,ancestor).

begin
# everybody has a father
All binder x. Some <child-> x ;
# ancestor is reflexive
All binder x. <ancestor> x;
/* nobody is both an ancestor of mary's and one of
   his own ancestors */
- (Some (<ancestor> mary & binder x.<ancestor>x))
end
```

3 Known bugs

1. The LaTeX functionality is not particularly refined, and the LaTeX file is far from being perfect.

   There is, for instance, a repetition of the description of the offspring relation when the equality rule is applied.

   Since internally generated nominals are here printed out with names of the form N_i, one should avoid using predicate names that can be confused with such nominals in the input file, such as `N_1, N_2`,..., that, in LaTeX become $N_1, N_2, \ldots$. In general, every underscore character used in an indentifiers becomes the LaTeX command for subscripts.

2. It may happen that the complete and open branch printed out with verbosity levels 3 and 4 (as well as the branches output in the LaTeX file when Sibyl is called with the option `-tex`) contains two nodes obtained by application of the same rule to the same node(s). However, such nodes are not actually both added to the branch, as can be checked by monitoring the step by step construction of the branch.

And many others will surely show up!