SLDNF-Draw user's manual
Chapter 1: static trees

Marco Gavanelli

December 26, 2016

1 Running and consulting

Run ECL^{i}PS^e and consult file `sldnf.pl`:

```
>eclipse
ECLiPSe Constraint Logic Programming System [kernel]
Copyright Imperial College London and ICL
Certain libraries copyright Parc Technologies Ltd
GMP library copyright Free Software Foundation
Version 5.7 #35, Thu Mar  4 00:15 2004
[eclipse 1]: [sldnf].
sldnf.pl   compiled traceable 12864 bytes in 0.00 seconds

Yes (0.00s cpu)
[eclipse 2]:
```

To draw the SLDNF tree for a goal, use predicate `draw_goal`/2. The first argument is a goal, the second is a file name. SLDNF-Draw will write the LaTeX commands painting the SLDNF tree to the file.

```
[eclipse 2]: draw_goal(X=Y+1,"tree.tex").

X = X
Y = Y
Yes (0.00s cpu)
[eclipse 3]:
```

As you can see in the example, the first parameter is a goal, and the second is the name of a file; in our case, tree.tex. Now, the file tree.tex contains exactly the LaTeX commands to draw the SLDNF tree for the goal X=Y+1.

Note that you cannot run LaTeX directly on the generated file, but you can copy and paste its content in your LaTeX documents, or you can include it in your LaTeX document by writing:

```
\input{tree}
```

As the commands in the generated file tree.tex contain commands of the packages ecltree and epic, you have to put

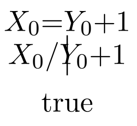
```
\usepackage{epic}
\usepackage{ecltree}
```

in the preamble of your LaTeX file.

The minimal file to produce the tree is (in the following, we will assume this file is called `envelope.tex`):

```
\documentclass{article}
\usepackage{epic}
\usepackage{ecltree}

\begin{document}

\input{tree}

\end{document}
```

Running pdflatex (or latex) on the previous file creates a pdf file (resp. a dvi file) with the following tree:



2 Adding a knowledge base

In order to generate SLDNF trees of a given program, you have to consult the program with all the predicates defined as dynamic predicates. In fact, SLDNF Draw is a meta-interpreter, and ECL^{i}PS^e requires you to declare the predicates as dynamic predicates to access the clauses by means of the clause/2 predicate.

Please, do not use the underscore symbol in the clause database, as it is a special symbol for LaTeX.

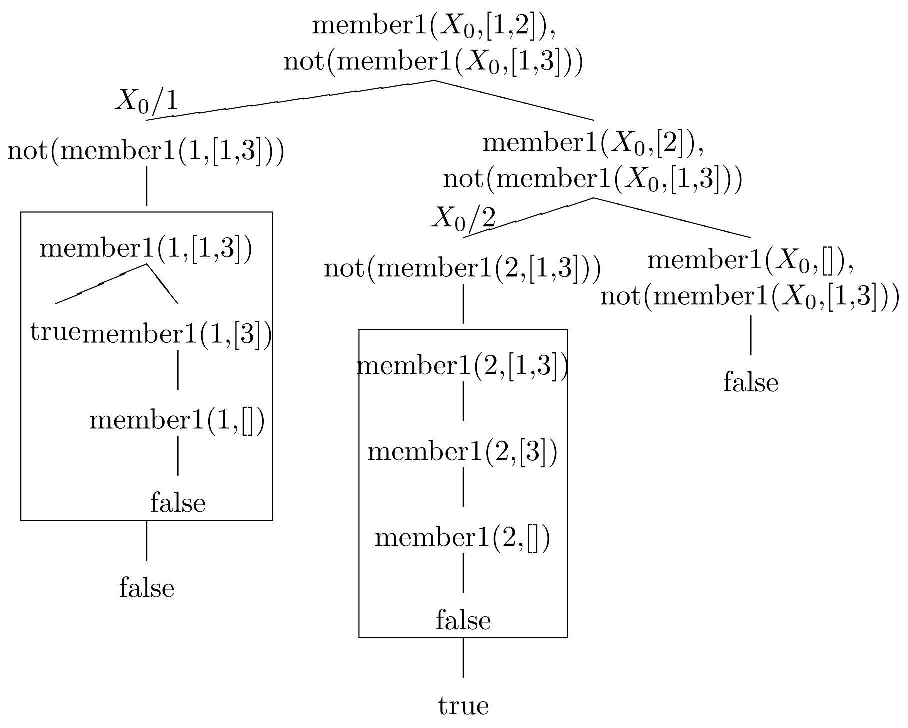
As an example, you could define a predicate member1/2 (member/2 is a predefined predicate in ECL^{i}PS^e, so it would complain if you tried to re-define it) as follows:

```
:- dynamic member1/2.

member1(X,[X|T]).
member1(X,[H|T]):-
    member1(X,T).
```

Suppose you wrote the definition in the file prog.pl, you can now consult it:

```
[eclipse 1]: [sldnf].
sldnf.pl   compiled traceable 12864 bytes in 0.00 seconds

Yes (0.00s cpu)
[eclipse 2]: [prog].
*** Warning: Singleton variables in clause 1 of member1/2: T
*** Warning: Singleton variables in clause 2 of member1/2: H
prog.pl    compiled traceable 184 bytes in 0.00 seconds

Yes (0.00s cpu)
[eclipse 3]: draw_goal(
              (member1(X,[1,2]),not member1(X,[1,3])),
             "tree.tex").

X = X
Yes (0.00s cpu)
[eclipse 4]:
```

and, by running LaTeX on the previous file `envelope.tex`, you get this tree:



3 Occur-check

SLDNF Draw can deal with occur-check as ECL^{i}PS^e does. You have to switch the occur-check on (see the ECL^{i}PS^e documentation) before loading the program.

4 Built-in predicates

SLDNF Draw deals also with built-in predicates. This is useful, e.g., for the `is/2` predicate. But beware! ECL^{i}PS^e compiles `is/2` into a predicate in the module `sepia_kernel`, that, unluckily, contains an underscore. So, in this case, I suggest to use the alternative syntax, for a clause:

```
my_clause(len([],0),[]).
my_clause(len([H|T],N),[len(T,N1),N is N1+1]).
```

Now, you can do the following:

```
[eclipse 1]: [sldnf].
sldnf.pl   compiled traceable 12864 bytes in 0.00 seconds

Yes (0.00s cpu)
[eclipse 2]: [prog].
*** Warning: Singleton variables in clause 2 of my_clause/2: H
```

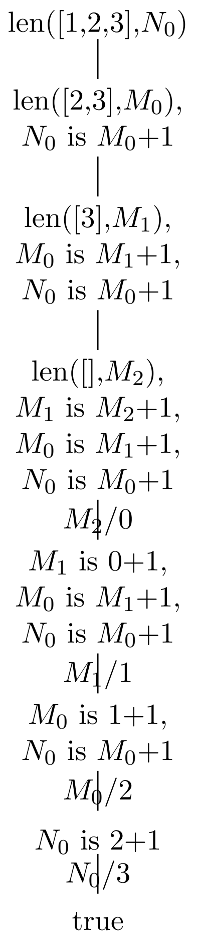

len([1,2,3],N_0)
len([2,3],M_0), N_0 is M_0+1
len([3],M_1), M_0 is M_1+1, N_0 is M_0+1
len([],M_2), M_1 is M_2+1, M_0 is M_1+1, N_0 is M_0+1
M_2/0
M_1 is 0+1, M_0 is M_1+1, N_0 is M_0+1
M_1/1
M_0 is 1+1, N_0 is M_0+1
M_0/2
N_0 is 2+1
N_0/3
true

Figure 1: Predicate `len`

```
prog.pl      compiled traceable 3424 bytes in 0.01 seconds

Yes (0.01s cpu)
[eclipse 3]: draw_goal(len([1,2,3],N),"tree.tex").

N = N
Yes (0.00s cpu)
```

And you get the tree in Figure 1.

Concerning other built-in predicates, the program works (if the predicate does not do too strange things ...) but it will consider the built-in predicate as a "flat" predicate. For example, the query:

```
draw_goal(member(X,[1,2,3]),"tree.tex").
```

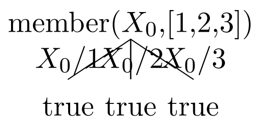


Figure 2: Tree generated using the built-in `member`/2 predicate

produces the tree in Figure 2 (this tree is not very readable, but you will learn how to customize the spacing in Section 5).

5 Customization of the output in the LaTeX code

The visualization of the tree can be customized in several ways; some are provided as Prolog predicates, others can be given as LaTeX commands; we provide here the customizations available in the LaTeX code, while in Section 6 we provide those available as Prolog predicates.

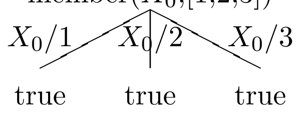
SLDNF-Draw relies on the LaTeX package `ecltree`, available in the standard LaTeX distributions. `ecltree` has various options to customize the rendering of the tree, as is detailed in its manual. In particular, the following parameters can be adjusted in the LaTeX code:

- `\GapDepth`: minimum height of gaps between adjacent nodes (the current value is 15.0pt)
- `\GapWidth`: minimum width of gaps between adjacent nodes (currently 4.0pt)
- `\EdgeLabelSep`: height of an edge label from the lower node of the edge (currently 7.0pt)

For example, the spacing of the tree in Figure 2 can be increased by setting

```
\setlength{\GapDepth}{2em}
\setlength{\GapWidth}{2em}
\setlength{\EdgeLabelSep}{1em}
\input{tree}
```

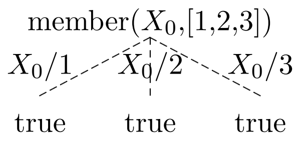
and the result is the following tree:



Also, the line style can be changed with all the styles available in `epic.sty`; for example

```
\drawwith{\dashline[65]{3}}
```

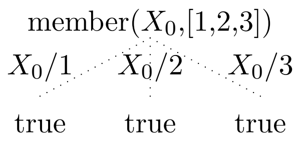
will produce trees with dashed lines, as follows



while

```
\drawwith{\dottedline{3}}
```

will produce:



Also, the commands `\thinlines` and `\thicklines` can be used to have, respectfully, the tree drawn with thin or thick lines.

See the `ecltree` manual for further details.

6 Customizations available as Prolog commands

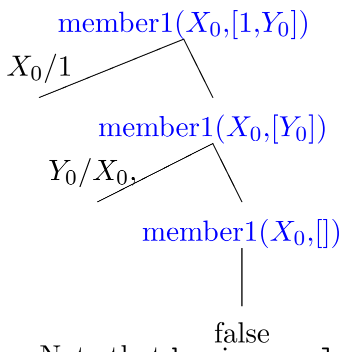
6.1 Customizing the resolvent

You can change they way the resolvent is visualized by defining two predicates: `begin_resolvent/1` and `end_resolvent/1`. The only parameter should be an atom or a string containing the LaTeX code that you want to be included before and after the resolvent. Note that the backslash character needs to be escaped.

For example, if you want the resolvent to be printed in blue color you can define:

```
begin_resolvent('{\\color{blue} ').
end_resolvent('}').
```

A sample output follows:



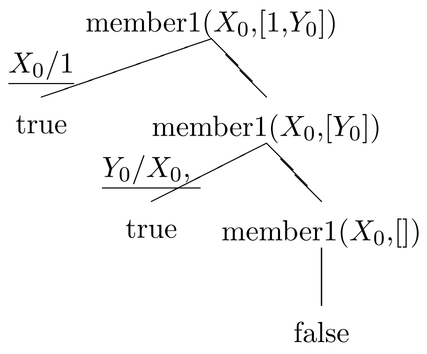
Note that `begin_resolvent/1` and `end_resolvent/1` are defined as dynamic predicates, so they can be defined from the Prolog command-line with `assert`.

6.2 Customizing the binding

In the same way, you can change they way the binding is visualized by defining two predicates: `begin_binding/1` and `end_binding/1`. The only parameter should be an atom or a string containing the LaTeX code that you want to be included before and after the binding. Again, that the backslash needs to be escaped.

For example, if you want the binding to be underlined you can define:

```
begin_binding('\\underline{').
end_binding('}').
```



6.3 Width of the resolvent

If the resolvent is long, it will be split on several lines, in order to avoid having trees too wide, that might not fit into a page.

The predicate `max_resolvent_length`/1 defines the maximum number of characters that should fit in a line of the resolvent. In case the length of the resolvent exceeds the value defined in `max_resolvent_length`/1, SLDNF Draw will try to split it into different lines. Note, however, that this is done only if the resolvent is a conjunction of several atoms: a single atom is never split.

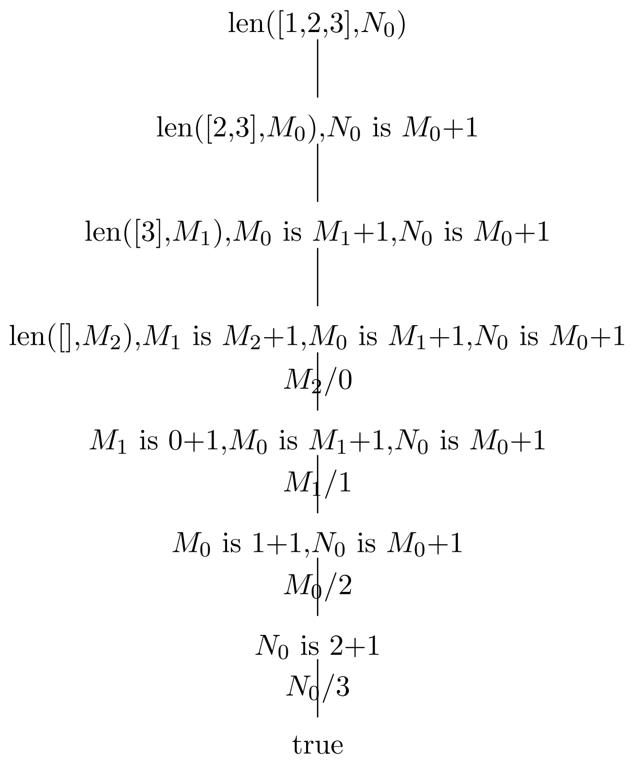
By default, the predicate `max_resolvent_length/1` is defined as

```
:- dynamic max_resolvent_length/1.
max_resolvent_length(25).
```

however the default value can be changed by redefining the predicate with `assert` and `retract`, for example:

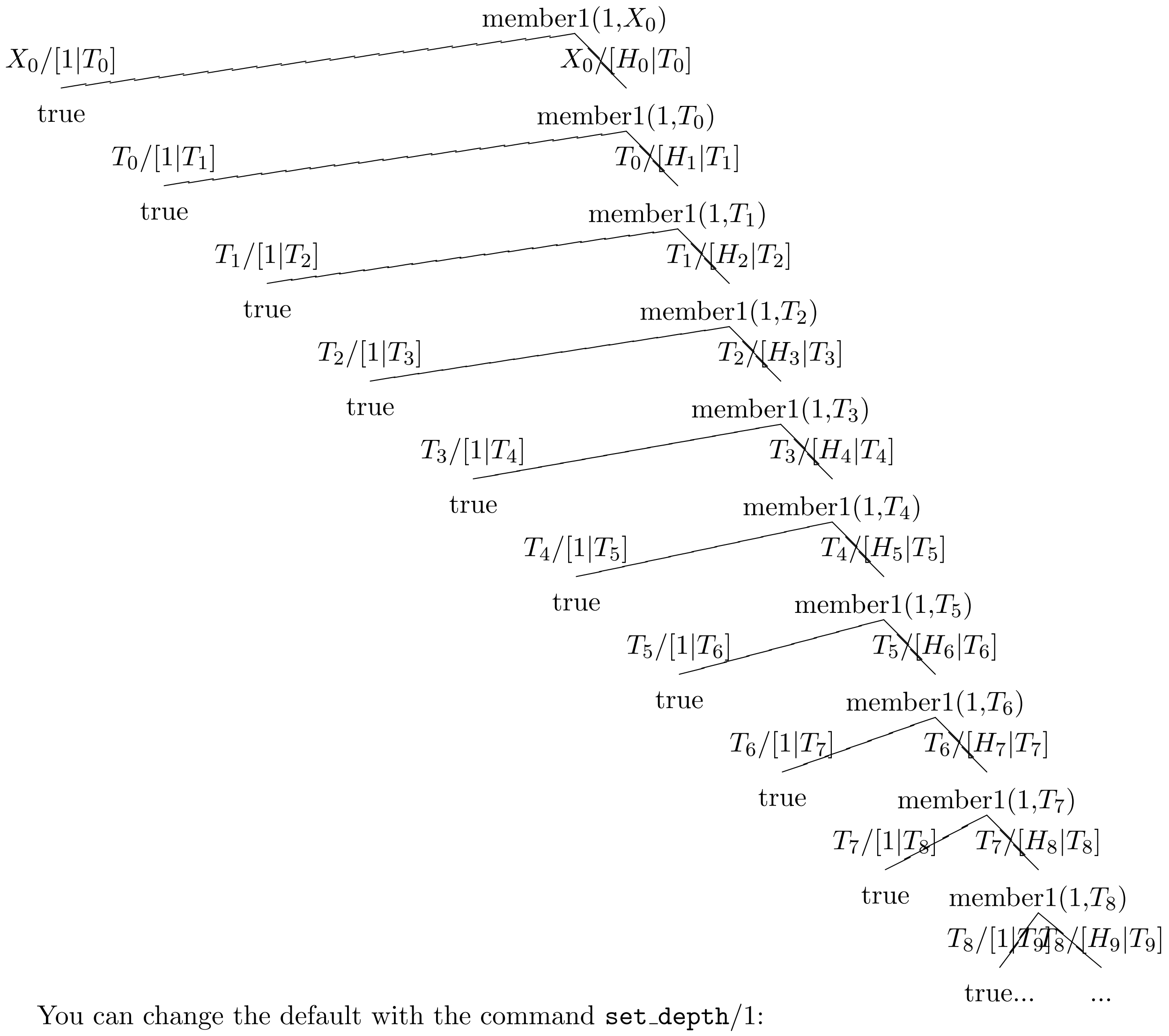
```
[eclipse 1]: retract(max_resolvent_length(_)).

Yes (0.00s cpu)
[eclipse 2]: assert(max_resolvent_length(200)).

Yes (0.00s cpu)
[eclipse 3]: draw_goal(len([1,2,3],N),"tree.tex").

N = N
Yes (0.00s cpu)
```

will produce the following tree, that is the same tree of Figure 1 without resolvent splitting:


len([1,2,3],N_0)
len([2,3],M_0),N_0 is M_0+1
len([3],M_1),M_0 is M_1+1,N_0 is M_0+1
len([],M_2),M_1 is M_2+1,M_0 is M_1+1,N_0 is M_0+1
M_2/0
M_1 is 0+1,M_0 is M_1+1,N_0 is M_0+1
M_1/1
M_0 is 1+1,N_0 is M_0+1
M_0/2
N_0 is 2+1
N_0/3
true

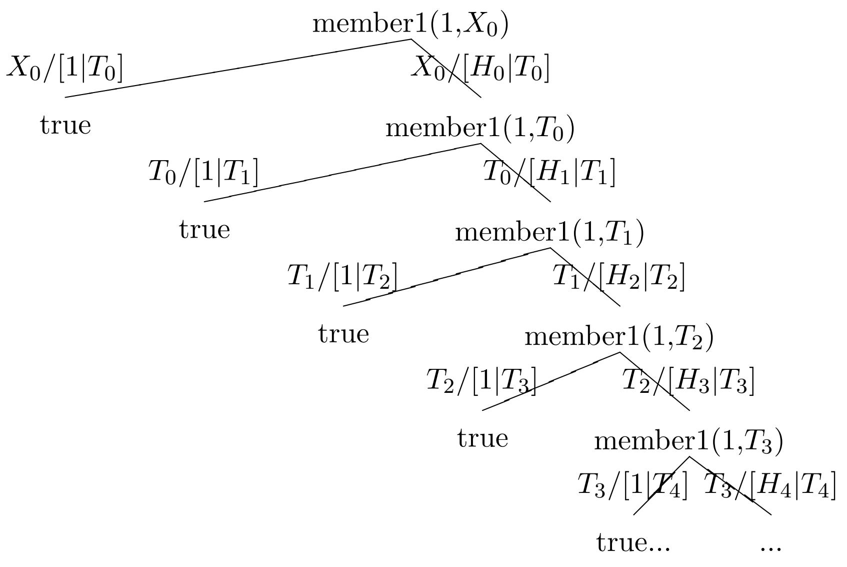
6.4 Maximum Depth

In order to avoid looping in infinite trees, SLDNF-Draw stops printing a tree after a predefined depth, by default 20.



You can change the default with the command `set_depth`/1:

```
[eclipse 1]: set_depth(5).

Yes (0.00s cpu)
[eclipse 2]: draw_goal(member1(1,X),"tree.tex").

X = X
Yes (0.00s cpu)
```



6.5 Cut, success and failure symbols

The following symbols are used by default:

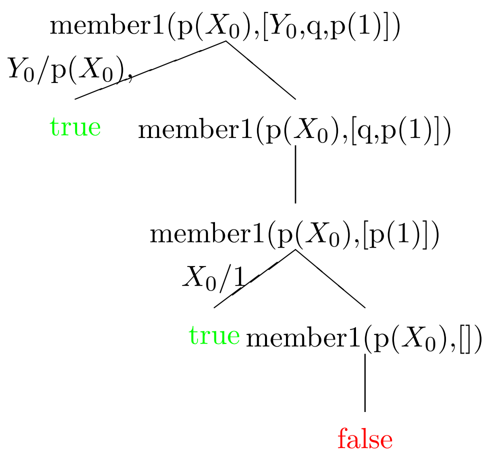
- ”true” indicates a success node (end of a derivation)
- ”false” indicates a failure node
- ”(cut)” indicates a cut node (a node which is not explored due to the execution of a cut (`!`))

These defaults can be changed by redefining the predicates `success_symbol`/1, `fail_symbol`/1 and `cut_symbol`/1. In all cases, the argument is a string (i.e., a text between double quotes). All these predicates are defined as dynamic predicates, so that they can be redefined at the Prolog interpreter through `assert` and `retract` commands.

For example, with

```
[eclipse 1]: retract(fail_symbol(_)),
             assert(fail_symbol("{\\color{red} false}")).

Yes (0.00s cpu)
[eclipse 2]: retract(success_symbol(_)),
              assert(success_symbol("{\\color{green} true}")).

Yes (0.00s cpu)
[eclipse 3]: draw_goal(member1(p(X),[Y,q,p(1)]),"tree.tex").
Yes (0.00s cpu)
```

one obtains a tree in which success nodes are in green and failure nodes are in red:



7 One predicate does all (for Linux with Gnome)

If your objective is not to generate a tree that you will include into your favorite document, but you just want to visualize the SLDNF tree of a goal, you can invoke predicate $ds(Goal)$.

Such predicate takes care of invoking latex for you, convert the output in pdf, and open the pdf viewer. It requires that files drawTree.sh and animateTree.sh are in the same folder. You can edit them to use you favorite pdf viewer.

Predicate $ds/1$ executes drawTree.sh in case animations are switched off, and animateTree.sh if animations are switched on (see "SLDNF Draw Users manual - Chapter 2: Animations").