Reseda version 1.2

User's manual

Author: Antoine Rauzy
Reference: IML/LP/Reseda/TN01-1
Version: 1 (10/07/2001)

Table of contents

I. Presentation of Reseda

Reseda is a network compiler. It scans network descriptions and it compiles them into Boolean equations (at the Aralia format) that encode either the cuts or the paths of the network.
The version 1.2 implements two compilation algorithms: the main algorithm is derived from the method proposed in [MCFB94] (see also [DRS96]). It works for all networks, including networks that embed loops and logical gates. The treatment of logical gates is a new functionality. The second algorithm works by backward induction. It is restricted to networks without loop (and a fortiori birectional edges). However, it produces a much simpler set of equations. Conversely to the set of equations produced by the first algorithm, that involves an universal quantifier, this latter set can be handled by any classical fault tree assessment tool.
Reseda proposes also several variable ordering heuristics. Variable ordering is a key issue in the assessment of Boolean formulae by means of Binary Decision Diagrams. Classical variable ordering heuristics do not work very well on sets of equations produced by Reseda because they fail to take into account the topology of these equations (the compiled network). Heuristics proposed by Reseda are simple and give satisfying results.

Reseda is basically a batch command for unix systems. For instance,

```
reseda file1 file2 file3
```

scans the network descriptions contained in file `file1', `file2' and `file3' and compiles them, according to compilation commands that are also given in these files. The results are displayed on the standard output. The version 1.2 proposes also an interactive mode (when invocated without arguments).

II. Network descriptions

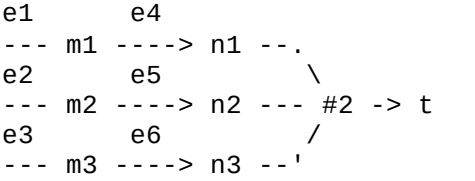
II.1. Syntax

The Reseda syntax for network is as follows (recall that the notation <objet> { <sep> <object> } stands for a non-empty list of <object> separated with <sep>).

```
<network> ::=
      network <identifier> ;
      nodes <nodes> ;
      labels <labels> ;
      edges <edges> ;
      source <node> ;
      target <node> ;
      reliable <components> ;
      aralia { <aralia-commands> }
      ranking <ranks> ;
      ;

<nodes>     ::= <node> { , <node> }
<node>  ::= <identifier>

<labels> ::= <label> { , <label> }
<label>  ::= <identifier>

<edges>     ::= <edge> { , <edge>}
<edge>      ::= <node> |- <label> -> <node>
      ::= <node> <- <label> -> <node>
      ::= <nodes> -# <integer> -> <node>

<components> ::= <component> { , <component> }
<component> ::= <node>
            ::= <label>

<ranks>     ::= <rank> { , <rank> }
<rank>      ::= <node> : <integer>
      ::= <label> : <integer>

<identifier> ::= [a-zA-Z][a-zA-Z0-9_-.]*
<integer> ::= [0-9]+
```

It is easier to understand that syntax on an example. Let us consider the following network, so-called "grid".

grid:



This network is made of the following components.
- 8 nodes: s, m1, m2, m3, n1, n2, n3 and t. s is the source of the network. t isthe target of the network.
- 3 bidirectional edges e1, e2 and e3 and 3 unidirectional edges e4, e5 and e6.
- A logical 2-out-of-3 gates (t is reachable from s if at least 2 out of the 3 nodes n1, n2 and n3 are reachable from s).

The Reseda description for this network is as follows.

```
network grid;
  nodes
    s, m1, m2, m3, n1, n2, n3, t;
  labels
    e1, e2, e3, e4, e5, e6;
  edges
     s <- e1 -> m1,
     s <- e2 -> m2,
     s <- e3 -> m3,
     m1 |- e4 -> n1,
     m2 |- e5 -> n2,
     m3 |- e6 -> n3,
     n1, n2, n3 -# 2 -> t;
  source
    s;
  target
    t;
  ;
```

Some precisions:
- Nodes and labels must declared (with commands "nodes" and "labels") before their use in edges.
- It is allowed to use the same label for two distinct edges. However, this should be avoided (we shall see later how to handle common cause failures).
- It is allowed to use a command twice. None of the command is mandatory.
- Take care that the description ends with a semi-colon `;'.
- The syntax for node, label and network identifiers is a letter followed by any sequence of letters, digits or characters `-', `.' and `_'.
- Comments can be inserted in Reseda description. A comment may spread over several lines. It starts with a `/*' and ends with a `*/'. Comments can be nested.

II.2. Principle of the compilation process

The Reseda description for networks includes several other commands that indicate what is to be compiled.

The principle of the compilation process is to associate a Boolean variable to each component of the network that is subject to failures, i.e. to each node and to each label (edge failures are thus represented as label failures). This variable represents the (potential) failure of the corresponding component, i.e. it is true when the component is failed and false otherwise. It is named as the component. Then, a set of Boolean equations is written that encodes a formula with a single output and the above variables as inputs. The compiler produces a formula that corresponds to operational paths and another one that corresponds to cuts. Prime implicants of these formulae encode exactly the minimal configuration in which such a path (resp. cut) exists.
the node s to the node t of the bridge network are the following.

```
{-s, -m1, -e1, -m2, -e2, -n1, -e4, -n2, -e5, -t}
{-s, -m1, -e1, -m3, -e3, -n1, -e4, -n3, -e6, -t}
{-s, -m2, -e2, -m3, -e3, -n2, -e5, -n3, -e6, -t}
```

Note that they contain only negative literals, for they describe what is working, i.e. what is not failed.

II.3. Commands to describe what is to be compiled

It is often the case that some of the elements of the network are perfectly reliable. It is thus useless to introduce variables to compile them (for these variables would be stuck at 0). This is indicated by means of the command `reliable' followed with a list of nodes and labels.

As explained above, the compiler builds a Boolean formula. The variables of this formula are the components (nodes and labels) of the network. It is possible to define these variables by means of an equation (in the Aralia format) and to associate probability laws with the the terminal variables (still in the Aralia format). This is done by means of the command `aralia' followed with Aralia commands (surrounded with `{' and `}').

Assume, for instance, that in the above network:
- Nodes are perfectly reliable.
- Edges e1 and e2 have a common cause failure dcc. They have also independant cause failures (which are respectively f1 and f2).
- Probability laws are associated with terminal variables.

The (extended) description for the network is as follows.

```
network grid;
  nodes
    s, m1, m2, m3, n1, n2, n3, t;
  labels
    e1, e2, e3, e4, e5, e6;
  edges
     s <- e1 -> m1,
     s <- e2 -> m2,
     s <- e3 -> m3,
     m1 |- e4 -> n1,
     m2 |- e5 -> n2,
     m3 |- e6 -> n3,
     n1, n2, n3 -# 2 -> t;
  source
    s;
  target
    t;
  reliable
    s, m1, m2, m3, n1, n2, n3, t;
  aralia {
    e1 := (dcc | f1);
    e2 := (dcc | f2);
    law dcc, f1, f2, e3, e4, e5, e6 exponential lambda;
    set parameter lambda lognormal 0.001 3;
    }
  ;
```

III. Reseda commands

There is several kinds of commands:
- Commands that print networks and Boolean formulae.
- Commands that compile networks into Boolean formulae.
- Commands that determine variable ordering.
- ...

The basic sequence of commands to compile the network grid is as follows.

```
load "grid.net";                /* loads the file in which grid is described */
compile grid;                   /* compiles the network grid */
ranking BFS grid;               /* determines a variable ordering */
print store grid > "grid.dag";  /* saves the result in the file "grid.dag" */
```

In other words, a Reseda file consists basically in a network description followed by several commands.

The commands that are available in the current version of Reseda are the following.

III.1. Commands to manage networks

load <filename> ;
This command loads the given file. File names must be surrounded with quotes: "grid.net", "Reseda/tests/grid.net", ...

clear <network> ;
This command removes the given network from the data base.

III.2. Commands to compile networks

compile MCFB <network> ;
This command compiles the given network with the main compilation algorithm. The resulting set of equations is created in a store associated with the network (it is not printed).

compile backward-induction <network> ;
This command compiles the given network with the backward-induction algorithm. The resulting set of equations is created in a store associated with the network (it is not printed). If the network contains loops, an error message is displayed.

compile <network> ;
This command tries first to apply the backward-induction compiler. If the network contains loops, it calls the MCFB compiler.

III.3. Commands to print networks and boolean formulae

print network <network> [<redirection>] ;
This command prints the given network in the Reseda format. The network is printed on the default output. The redirection option makes it possible to print it on a different file. The syntax for the redirection option is `> <filename>' (rewrite mode) or `>> <filename>' (append mode). E.g.
print grid > "grid.net";

print store <network> [<redirection>] ;
This command prints all of Boolean formulae, probability laws and probability law parameters that are associated with the given network.

print components <networks> ;

This command prints the strongly connected components (with more than one node) of the given network.

III.4. Commands for variable ordering

ranking user-defined <network> ;
Boolean variables are associated with network components (nodes and labels) either by the directive `aralia' or during the compilation process. This command defines BDD indices for input variables according to the ranks of the components. The rank of each component (i.e. each node and each label) of the network are assumed to be given by the user through the directive `ranking'. Ranks are integer.Two distinct components are assumed to have distinct ranks.

ranking BFS <network> ;
This command works in two steps. First, it associates a rank to each component (node and label) of the network by means of a breadth-first traversal starting from the source node. Then, it defines BDD indices according to these ranks (see explanation above). This heuristics produces in general very good results.

ranking DFS <network> ;
This command works in two steps. First, it associates a rank to each component (node and label) of the network by means of a depth-first traversal starting from the source node. Then, it defines BDD indices according to these ranks (see explanation above). This heuristics produces in general rather poor results.

References

[MCF94] J.-C. Madre, O. Coudert, H. Fraisse and M. Bouissou. Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis, in Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94, Annaheim, California, pp 118-123, 1994.

[DRS96] Y. Dutuit, A. Rauzy and J.-P. Signoret. Reseda: a Reliability Network Analyser, in Proceedings of European Safety and Reliability Association Conference, ESREL'96, pp 1947-1952, vol. 3, C. Cacciabue and I.A. Papazoglou eds., Springer Verlag, ISBN 3-540-76051-2, 1996