ReLTS 1.0

User Manual

Mihir Mehta

Shibashis Guha

November 29, 2014

Contents

1 Introduction

ReLTS is an Ocaml tool that checks the existence of a family of relations parameterised by the number of alternations(n) and the number of rounds(k). Following Stirling's approach, we use two player games to implement our relation checking. This approach lends itself very naturally to the generation of distinguishing formulae which we report when the challenger wins.

2 Installation

ReLTS runs on GNU/Linux. The installation follows the familiar

```
$ ./configure
$ make
$ sudo make install
```

installation procedure, which installs the `relts` executable in `/usr/local/bin`

Installation requires Ocaml 3.12 or newer, the `findlib` tool and the `Ocamlgraph` library. It may work with previous versions of Ocaml but this has not been tested.

In addition, an executable `random_lts` is also installed. This is a C++ tool which allows generation of random LTSs with a specified number of states. Compiling `random_lts` requires GCC 4.6 or newer. In Ubuntu, the corresponding packages that need to be installed are ocaml-inetrp, ocaml-findlib and libocamlgraph-dev.

3 Usage

ReLTS accepts LTSs as digraphs in the dot format. Each vertex should be a unique integer and each edge should be labelled by an action which is an integer.

Example. `lo5.dot`

```
digraph {
  23 -> 23 ["action" = 0]
  23 -> 24 ["action" = 0]
}
```

Example. `lo6.dot`

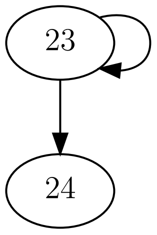

Figure 1: l05.dot

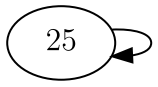

Figure 2: l06.dot

```
digraph {
  25 -> 25 ["action" = 0]
}
```

As it can be observed in these two examples, when running ReLTS on two LTSs, their vertices should be disjoint.

ReLTS requires the specification of the options `--lts1`, `--lts2`, `-p` and `-q` on the command line, as in this example.

Example. `$ relts --lts1 l05.dot --lts2 l06.dot -p 23 -q 25`

```
The relation does not hold.
n = 1, k = 2, f = <0>[0]ff
```

In this example, the challenger starts its move from state p in $lts1$ and the defender starts from state q in $lts2$. In the output, `n` and `k` denote the number of alternations and the number of rounds the challenger needs respectively in order to win the game. By default, the number of alternations and the number of rounds allowed in the game are assumed to be infinite, i.e. there is no restriction either on the number of alternations or on the number of rounds. They can be specified using the switches `-n` and `-k` respectively. Note that these switches are different from the `n` and `k` that are part of the output.

Example. Setting the number of alternations allowed in the game to 0

```
$ relts --lts1 l05.dot --lts2 l06.dot -p 23 -q 25 -n 0
The relation holds.
```

Example. Similarly, setting the number of rounds allowed in the game to 1,

```
$ relts --lts1 l05.dot --lts2 l06.dot -p 23 -q 25 -n 0
The relation holds.
```

We can also evaluate the relation where the challenger has a choice to start from the p state in $lts1$ or the q state in $lts2$ with the `--equivalence` switch.

Example. `$ relts --lts1 l06.dot --lts2 l05.dot -p 25 -q 23`

```
The relation does not hold.
n = 2, k = 3, f = <0>([0]<0>tt  && <0>tt)
```

`f` is the distinguishing formula. For an action 0, $< 0 >$ and [0] have their usual meanings. `tt` and `ff` stand for true and false respectively and the infix operators && and || represent conjunction and disjunction respectively.

Example. `$ relts --lts1 l06.dot --lts2 l05.dot -p 25 -q 23 --equivalence`

```
The relation does not hold.
n = 1, k = 2, f = <0>[0]ff
```

The last two examples show that the challenger may have a better strategy if it starts from state q instead of state p. Primarily, the switch `--equivalence` evaluates the relation for the case where once the game is played such that the challenger chooses p in the first round and next the game is played such that the challenger chooses q in the first round. The challenger wins if it has a winning strategy by starting from either p or from q. They also demonstrate that the elimination of non-optimal strategies, which is the key feature of ReLTS.

In case we only require only the `n, k` pairs for optimal strategies but not the corresponding distinguishing formulae, the switch `--pairs` may be used.

Example. `$ relts --lts1 l06.dot --lts2 l05.dot -p 25 -q 23 --pairs`

```
The relation does not hold.
n = 2, k = 3
```

If we only need to know whether the challenger has a winning strategy, we can use the `--relation` switch.

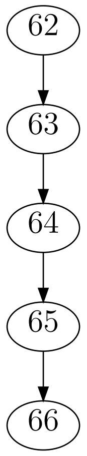

Figure 3: l18.dot

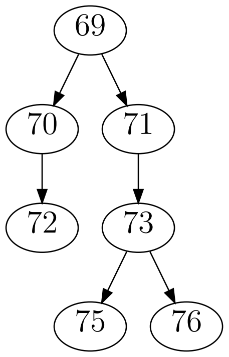

Figure 4: l22.dot

Example.
```
$ relts --lts1 l06.dot --lts2 l05.dot -p 25 -q 23 --relation
The relation does not hold.
```

ReLTS produces multiple distinguishing formulas which is demonstrated in the following example:

Example. `l18.dot`
```
digraph {
  62 -> 63 ["action" = 0]
  63 -> 64 ["action" = 0]
  64 -> 65 ["action" = 1]
  65 -> 66 ["action" = 2]
}
```

```
l22.dot
digraph {
  69 -> 70 ["action" = 0]
  69 -> 71 ["action" = 0]
  70 -> 72 ["action" = 0]
  71 -> 73 ["action" = 0]
  73 -> 75 ["action" = 1]
  73 -> 76 ["action" = 3]
}
    $ relts --lts1 l18.dot --lts2 l22.dot -p 62 -q 69
The relation does not hold.
n = 0, k = 4, f = <0>(<0><1>tt && <0><1><2>tt)
n = 1, k = 3, f = <0>(<0><1>tt && <0>[3]ff)
```

There are two distinguishing formulae corresponding to two strategies of the challenger which are not comparable.

3.1 Test Suite

A number of test cases has been provided along with the ReLTS download. The folder `test` contains these test cases. The user can `cd` to this directory and run each of the examples provided in this manual.

3.2 Generating Random LTSs

`random_lts` is a simple C++ tool which generates random LTSs with a given number of states. By default, the LTS produced is a rooted binary tree. The edges are directed and all of them are labelled with action 0.

`-n` is the only mandatory option. It specifies the number of states.

Example. `$ random_lts -n 8`

```
digraph {
1 -> 2 ["action" = 0]
1 -> 3 ["action" = 0]
3 -> 4 ["action" = 0]
3 -> 6 ["action" = 0]
4 -> 5 ["action" = 0]
6 -> 7 ["action" = 0]
7 -> 8 ["action" = 0]
}
```

A back edge can be introduced with the use of the `-c` switch producing a cycle in the LTS. In the following example, the back edge is `8 -> 1 ["action" = 0]`.

Example. `$ random_lts -n 8 -c`

```
digraph {
```

```
1 -> 2 ["action" = 0]
1 -> 6 ["action" = 0]
2 -> 3 ["action" = 0]
2 -> 4 ["action" = 0]
4 -> 5 ["action" = 0]
6 -> 7 ["action" = 0]
7 -> 8 ["action" = 0]
8 -> 1 ["action" = 0]
}
```

Example. `$ random_lts -n 16 -b 0 > l01.dot; random_lts -n 16 -b 16 > l02.dot; relts -`

```
The relation does not hold.
n = 1, k = 4, f = <0>(<0>(<0>[0]ff && <0>tt) && <0>[0]ff)
```

Example. `$ random_lts -n 16 -b 0 > l03.dot; relts --lts1 l03.dot --lts2 l03.dot -p 1`

```
The relation holds.
```

The -b option advances all the vertex numbers by its argument, so, for instance,

Example. `$ random_lts -n 16 -b 0 > l01.dot; random_lts -n 16 -b 16 > l02.dot`

```
$ relts --lts1 l01.dot --lts2 l02.dot -p 1 -q 17
```

will generate l02.dot with vertices labelled 17 through 32.

Note how the -b option is used to ensure the vertex labels of l01.dot and l02.dot are disjoint in the first example

The above example shows the application of random_lts towards generating test cases of a given size and profiling running time and memory utilisation.

3.3 Wrapper for time abstracted relations

`ta_wrapper.sh` is a simple shell script which provides a wrapper over ReLTS and reltool to allow parameterised time abstracted relations on timed automata to be determined through zone graphs. It assumes that ReLTS and reltool are already installed, as well as the standard command-line utilities, head and tail.

Example. `$ bash ta_wrapper.sh example1.txt example2.txt`

```
The relation does not hold.
n = 0, k = 3, f = <0><d>(<1>tt && <1>tt && <1>tt)
```

This example uses the timed automata described by files example1.txt and example2.txt which are distributed with reltool (in the top level directory). Here, `0` and `1` are actions in these timed automata, while `d` is a delay action, which is usually represented by ϵ in zone graphs.