ReLTS 1.0

User Manual

Mihir Mehta Shibashis Guha

April 27, 2014

Contents

1 Introduction **1**

2 Installation **1**

3 Usage **1**

3.1 Generating Random LTSs 4

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 the generation of random LTSs with a specified number of states. Compiling `random_lts` requires GCC 4.6 or newer.

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.

Examples Following is an LTS saved as `lo5.dot`.

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

The following LTS is saved as `lo6.dot`

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

As you can see 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.

```
$ relts --lts1 l05.dot --lts2 l06.dot -p 23 -q 25
The relation does not hold.
n = 1, k = 2, f = <0>[0]ff SHOULD WE CHANGE THE OUTPUT?
```

In this example, the challenger starts its move from the p state in $lts1$ and the defender starts from the q state in $lts2$. By default, n and k 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.

Setting n to 0

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

Similarly, setting k 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.

```
$ 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.

```
$ 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 switches side before its first move. 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.

```
$ 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.

```
$ 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:

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

```
$ 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]`.

```
$ 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]
}
```

```
$ random_lts -n 16 -b 0 > l01.dot; random_lts -n 16 -b 16 > l02.dot; relts --lts1 l01.
The relation does not hold.
n = 1, k = 4, f = <0>(<0>(<0>[0]ff && <0>tt) && <0>[0]ff)
```

```
$ random_lts -n 16 -b 0 > l03.dot; relts --lts1 l03.dot --lts2 l03.dot -p 1 -q 1
The relation holds.
```