ReLTS: A Tool for Checking Relations over LTSs

Mihir Mehta [1] Shibashis Guha [2] S. Arun-Kumar [2]

[1]Samsung Research Institute, Noida
`mihir.mehta@samsung.com`

[2]Indian Institute of Technology Delhi
{`shibashis,sak`}`@cse.iitd.ac.in`

Abstract. We present ReLTS, a tool that implements a generalized form of Stirling's bisimulation game. Given two labelled transition systems (LTSs), ReLTS checks the existence of a family of relations parameterised by number of rounds and number of alternations. These relations provide a quantitative measure of the similarity between two LTSs which otherwise cannot be captured by simulation or bisimulation relations. Simulation preorder, bisimulation, simulation equivalence are special cases of this family. If the challenger has a winning strategy, ReLTS also reports the number of alternations and rounds involved in the best possible strategies for the challenger and generates distinguishing formulae for these strategies.

1 Introduction

ReLTS is a tool that checks the existence of a family of relations parameterised by the number of alternations(n) and the number of rounds(k). This family includes the well known relations simulation preorder, simulation and bisimulation equivalence as special cases. We generalize Stirling's approach of modelling bisimulation checking as a game[6] between a challenger and a defender, by specifying the maximum number of alternations and rounds allowed in the game. Although a theoretical framework for these generalized relations has been developed in [1], we are not aware of the existence of a tool which decides these relations. The generation of the distinguishing formulae while playing a game is another important feature of ReLTS. Removing the restrictions on the number of alternations and the number of rounds gives us the bisimulation game. Similarly, removing the restriction on the number of rounds and disallowing alternations give us the game corresponding to simulation preorder.

ReLTS is written in Ocaml and it uses the Ocamlgraph library [2] for representing LTS. For each optimal winning strategy A of the challenger, the tool reports the Hennessy-Milner logic(HML) [4] based distinguishing formula f_A, the number of alternations n_A and the number of rounds k_A that the challenger needs to win the game. Together, n_A and k_A measure the 'ease' with which the challenger wins. The distinguishing formula helps one to understand the difference between the two LTSs. Recent versions of the Edinburgh Concurrency Workbench [5] also provide distinguishing formulae generated through the game playing approach, but parameterised relations are not included.

2 Architecture of ReLTS

ReLTS uses the dot format for specifying the input LTSs. If the challenger has several strategies to win, only the optimal strategies are reported. Strategy A represented by the tuple (n_A, k_A, f_A) is considered *at least as good as* strategy B represented by the tuple (n_B, k_B, f_B) iff $(n_A \leq n_B)$ and $(k_A \leq k_B)$. This notion of goodness evidently forms a preorder, therefore multiple optimal strategies may exist. The core functionality of the tool is provided by the function

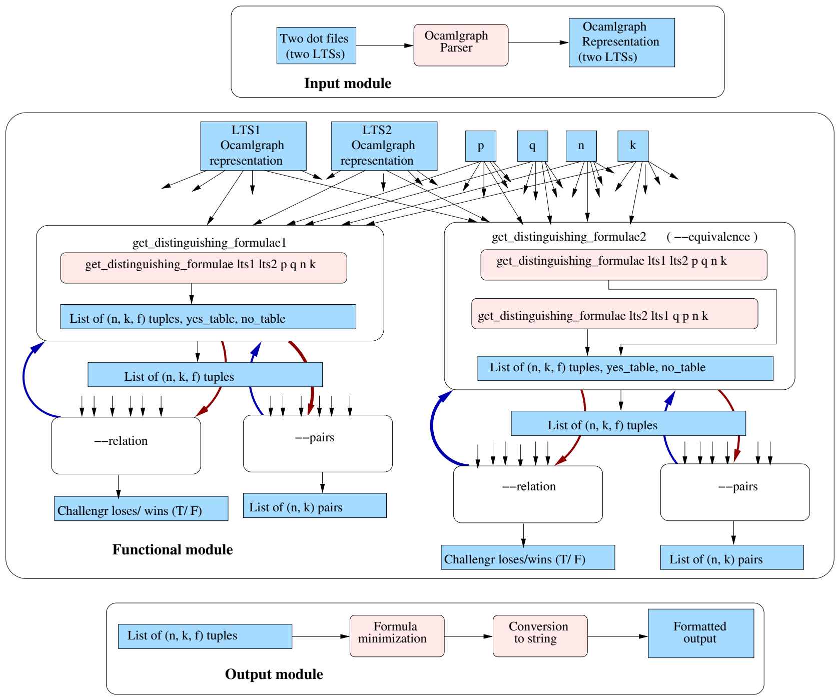

Two dot files (two LTSs)
Ocamlgraph Parser
Ocamlgraph Representation (two LTSs)
Input module
LTS1 Ocamlgraph representation
LTS2 Ocamlgraph representation
p
q
n
k
get_distinguishing_formulae1
get_distinguishing_formulae lts1 lts2 p q n k
List of (n, k, f) tuples, yes_table, no_table
get_distinguishing_formulae2 (--equivalence)
get_distinguishing_formulae lts2 lts1 q p n k
List of (n, k, f) tuples
--relation
--pairs
Challengr loses/ wins (T/ F)
List of (n, k) pairs
Functional module
Formula minimization
Conversion to string
Formatted output
Output module

Fig. 1. Components of ReLTS

`get_distinguishing_formulae`. When the challenger and the defender make their first move from state p in $lts1$ and state q in $lts2$ respectively, the call `get_distinguishing_formulae lts1 lts2 p q n k yes_table no_table` returns a list of optimal winning strategies for the challenger in which it uses at most n alternations and at most k rounds. It also returns the updated values of *yes_table* and *no_table* which are the data structures used for memoisation. The list is empty iff the challenger has no winning strategies. A negative value for n denotes no restriction on the number of alternations and likewise for k.

The command line interface provides a number of options for customising the output from `get_distinguishing_formulae`. Primarily, the switch `--equivalence` evaluates the relation for the case where once the game is played such that the

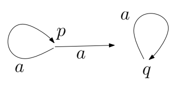
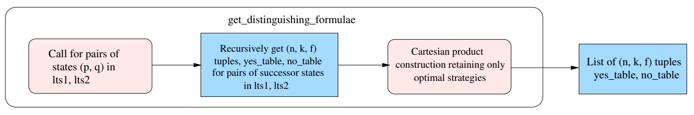


Fig. 2. A simple example



get_distinguishing_formulae
Call for pairs of states (p, q) in lts1, lts2
Recursively get (n, k, f) tuples, yes_table, no_table for pairs of successor states in lts1, lts2
Cartesian product construction retaining only optimal strategies
List of (n, k, f) tuples yes_table, no_table

Fig. 3. Internals of `get_distinguishing_formula`

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. Further, the option `--pairs` filters out the distinguishing formulae and displays only (n, k) pairs for optimal strategies, and the option `--relation` further reduces the output to a single line indicating whether or not the relation holds.

2.1 Example

We refer to Figure 2. Starting from p in $lts1$, the challenger has a winning strategy with 1 alternation and 2 rounds while starting from q, it wins with 2 alternations and 3 rounds. Thus `get_distinguishing_formulae2 lts1 lts2 p q (-1) (-1)` in turn calls `get_distinguishing_formulae lts1 lts2 p q n k yes_table no_table` that returns a list of strategies as part of the return value such that a strategy A being (n_A, k_A, f_A), with $n_A = 1$ and $k_A = 2$ as mentioned above. The subsequent call to `get_distinguishing_formulae lts2 lts1 q p n k yes_table no_table` returns a list of strategies as part of the return value such that a strategy B being (n_B, k_B, f_B), with $n_B = 2$ and $k_B = 3$. The two lists of strategies are merged; and B is discarded in the process since it is clearly worse than A. The distinguishing formula is $\langle a\rangle[a]\mathtt{ff}$.

3 Algorithm

Consider the call
`get_distinguishing_formulae lts1 lts2 p q n k yes_table no_table`.
Consider a transition $p \xrightarrow{a} p'$. We can obtain a strategy for this transition for the following two cases: 1. $\nexists q_i$ such that $q \xrightarrow{a} q_i$ (Base case). The strategy is simply $(0, 1, \langle a\rangle\mathtt{tt})$. 2. $\forall q_i$, such that $q \xrightarrow{a} q_i$, there exists a strategy (n_i, k_i, f_i) (Induction case). For the pair (p', q_i), the list of strategies L_{p',q_i} such that the challenger starts from p', is merged with the list of strategies $L_{q_i,p'}$, where the challenger starts from q_i, in the following way. $merge(L_{p',q_i}, L_{q_i,p'}) = L_{p',q_i} \cup \{(n_A + 1, k_A, \neg f_A) \mid (n_A, k_A, f_A) \in L_{q_i,p'}\}$. The non-optimal strategies are then removed from the merged list to obtain a set of strategies M_i. The final strategies for (p, q) such that the challenger starts from p is defined as the set of optimal strategies in $\prod_i (n_{Ai}, k_{Ai} + 1, f_{Ai})$, where $(n_{Ai}, k_{Ai}, f_{Ai}) \in M_i$. The product of two strategies $(n_{Ai}, k_{Ai} + 1, f_{Ai}) \in M_i$ and $(n_{Aj}, k_{Aj} + 1, f_{Aj}) \in M_j$ is defined as $(\max(n_{Ai}, n_{Aj}), \max(k_{Ai} + 1, k_{Aj} + 1), f_{Ai} \wedge f_{Aj})$. p' satisfies both f_{A_i} and

States	Acyclic LTS, no alt	Cyclic LTS, no alt	Acyclic LTS, alt
63	0.041	0.1	.5
127	0.459	0.5	11.7
255	5.1	8.2	606.8
511	79.3	123.4	—

Table 1. All durations in seconds

f_{A_j}, whereas none of q_i or q_j satisfies both f_{A_i} and f_{A_j}. Hence we have the distinguishing formula $f_{A_i} \wedge f_{A_j}$.

A set M_{ij} consisting of the product of the strategies from M_i and M_j is built incrementally such that a strategy A (n_A, k_A, f_A) is included in the set M_{ij} iff none of the strategies already included is as good as A or better. With the inclusion of A, the strategies in M_{ij} that are worse than A are removed from M_{ij}. Thus the *Cartesian product* is built incrementally while removing non-optimal strategies at every stage. The procedure creates only the set of optimal strategies since for any of the strategies A (n_A, k_A, f_A) reported, there does not exist any winning strategy B (n_B, k_B, f_B) for the challenger such that $n_B < n_A$ and $k_B <= k_A$ or $n_B <= n_A$ and $k_B < k_A$, and in whatever way the defender plays the challenger can ensure a win with n_B alternations and k_B rounds.

As a convention, the distinguishing formula generated is satisfied by the state in the LTS chosen by the challenger in the first round. We note that if the challenger in some round i makes a move a from a state r_i, then r_i satisfies a formula of the form $\langle a \rangle f_i$ while the state from which the defender plays satisfies the negation of the formula. If the LTS chosen by the challenger be the same as the LTS from which the defender makes its move in some round, i.e. there exists at least one alternation in the game, then the distinguishing formula will include subformulae having $[\,]$ and possibly $\bigvee$ constructs. This recursive dynamic programming algorithm of building the optimal strategies based on the optimal strategies from the successors is implemented in `get_distinguishing_formulae`.

4 Benchmarks

Table 1 has running time measurements of ReLTS on LTSs with increasing numbers of states. The tests were run on a machine with CPU speed 2.5GHz and 8 GB RAM. Acyclic LTSs were generated as binary trees with directed edges pointing away from the root, while the cyclic LTSs had additionally a back edge from one of the leaves to the root. All the edges were labelled with the same action. The number of states chosen in our experiments corresponds to the number of nodes in a full binary tree. In the experiments, the relations were checked between two copies of the same LTS. In Table 1, in the columns marked *'no alt'*, the number of alternations was restricted to zero. In the columns marked *'alt'*, the number of alternations was unrestricted. As an indication, the memory usage was 29.9 MB for cyclic LTS with 511 states in the *'no alt'* case, and it was 11.5 MB for acyclic LTS with 255 states in the *'alt'* case. The number of rounds was unrestricted in all experiments.

5 Availability and Possible Evolution

The tool can be downloaded from the project homepage, `http://airbornemihir.github.io/lts_reltool/`.
The present work has a natural extension into timed relations, taking advantage of the zone graph which is a finite representation of the timed LTS. We have made progress on constructing these zone graphs, which we plan to build upon for deciding timed relations. The game playing strategy for timed relations provides a unifying approach to decide several timed preorders, bisimulations and prebisimulations [3].

References

1. X. Chen and Y. Deng. Game characterizations of process equivalences. In *Proceedings of APLAS*, pages 107–121, 2008.
2. S. Conchon, J. C. Filliâtre, and J. Signoles. Designing a generic graph library using ml functors. In *Trends in Functional Programming*, pages 124–140, 2007.
3. S. Guha, S. N. Krishna, C. Narayan, and S. Arun-Kumar. A unifying approach to decide relations for timed automata and their game characterization. In *EXPRESS/SOS*, EPTCS, pages 47–62.
4. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, pages 137–161, 1985.
5. Perdita Stevens. `http://homepages.inf.ed.ac.uk/perdita/cwb/summary.html`.
6. C. Stirling. Local model checking games. In *Proceedings of CONCUR*, pages 1–11, 1995.