ReLTS: A Tool for Checking Generalized Behavioural Relations over LTSs

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

[1] University of Texas at Austin
mihir@cs.utexas.edu

[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

There are several software tools that implement bisimulation-checking on labelled transition systems (LTS). Prominent among these are Auto and Autograph [10,2], Aldébaran [7], the Edinburgh concurrency workbench (several versions e.g. [4], [11]) and the Concurrency Workbench of the New Century [5]. Many of these tools implement algorithms for bisimulation-checking that are derived from the Paige-Tarjan algorithm for partition-refinement. The more recent version of the Edinburgh Concurrency Workbench [11] implements a game-theoretic formulation similar to the bisimulation games described in Stirling [12].

A recent paper [3] describes a parameterised version of the game described in [12], where the game is parameterised by the maximum number of rounds (n) and the maximum number of alternations (k) allowed in a game. This theory is further developed in [8] wherein an infinite hierarchy of *timed games* is described over timed automata. Besides the simulation preorder, the simulation equivalence and bisimilarity, the hierarchy allows for a much finer-grained control on the parameters of the game and therefore allows for many more preorders and equivalences to be described.

ReLTS is a tool that implements this hierarchy of game-based relations. Removing restrictions on the number of alternations and on the number of rounds yields bisimulation game. Similarly removing the restrictions on the number of rounds and disallowing alternations yields a game corresponding to the simulation preorder.

These relations over LTSs can be extended to relations over timed automata[1], by means of evaluating the relations over zone graphs of timed automata. Using `reltool`, a tool for generation of zone graphs, it is possible to evaluate time abstracted bisimulation [14] parameterised over n and k as well.

2 Game Characterization of Bisimilarity

The bisimulation game is played on labelled transition systems.

Definition 1. *Let $(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$ be a labelled transition system (LTS) with Proc being the set of states in the LTS and Act being the set of actions. A strong bisimulation game starting from the pair of states $(s_1, t_1) \in Proc \times Proc$ is a two-player game with a challenger and a defender. The game is played in* rounds *and the* configurations *of the games are some pairs of states (s, t) in $Proc \times Proc$ that are reachable from the* initial configuration *as described below. The initial configuration is a designated pair of states (s_1, t_1). In the beginning of the game, the initial configuration is the current configuration and in each round the players change the current configuration (s_1, t_1) according to the following rules.*

1. *The challenger chooses either s or t of the current configuration which we refer to as the left side and the right side of the configuration respectively and the challenger also chooses an action $\alpha \in Act$.*
2. *The challenger performs an action $s \xrightarrow{\alpha} s'$ or $t \xrightarrow{\alpha} t'$ depending on whether it chooses s or t in the current configuration. Here s' and t' belong to Proc.*
3. *The defender chooses the side not chosen by the challenger and replicates the action, i.e. if the challenger chooses the left side, the defender performs $t \xrightarrow{\alpha} t'$ while if the challenger chooses the right side, the defender performs $s \xrightarrow{\alpha} s'$. The current configuration changes to (s', t') after the move of the defender.*
4. *The game continues to the next round according to the rules described above from the new current configuration (s', t').*

In every round, the challenger can choose a side, an action and a transition. The defender's only choice is to choose an available transition with the same action. A *play* of the strong bisimulation game is a maximal sequence of configurations formed by the players starting from the initial configuration (s_1, t_1). A sequence of configurations is *maximal* if it cannot be extended while following the rules of the game. A play can be finite or infinite. The challenger loses a finite play at the current configuration (s, t) iff $s \not\rightarrow$ and $t \not\rightarrow$. In any infinite play, the challenger loses. This means that the defender is always able to respond to the challenger's move and the challenger cannot identify a difference between the two states of any configuration reachable from the initial configuration. We note that a bisimulation game can have many different plays depending on the

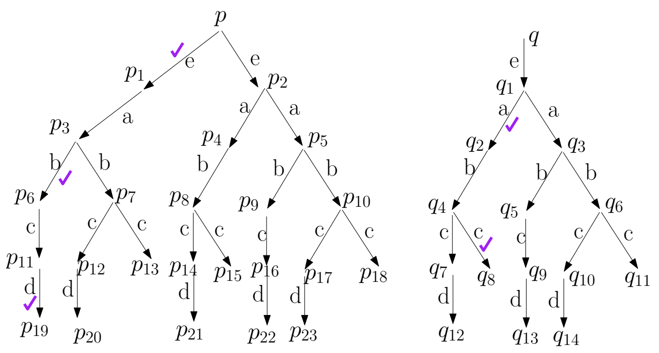

Fig. 1. An example of a strong bisimulation game with five rounds and four alternations

transitions chosen by the challenger and the defender. A given play is either winning for the challenger or for the defender but not for both. The following game characterization of bisimulation is due to Stirling [12] and Thomas [13].

Proposition 1. *States s_1 and t_1 of an LTS are strongly bisimilar iff the defender always has a move following the rules of the game corresponding to any of the transitions chosen by the challenger in every configuration reached starting from the configuration (s_1, t_1).*

We note that if s_1 and t_1 belong to two separate LTSs this is equivalent to considering them in the same LTS that can be constructed by adding an edge between s_1 and t_1. This edge can be labelled with any arbitrary action $\alpha \in Act$. Figure 1 illustrates an example of a strong bisimulation game between two states s_1 and t_1. The ✔indicates the moves chosen by the challenger in various rounds. We note that the play consists of five rounds and four alternations. As mentioned earlier, an alternation is said to happen if the challenger switches side between two successive rounds. The initial configuration is (p, q). In the beginning of round 5, the configuration reached is (p_{11}, q_8) when the challenger makes the move $p_{11} \xrightarrow{d} p_{19}$ while the defender cannot replicate the d action from q_8 and thus loses. Hence p and q are not strongly bisimilar according to Proposition 1.

We note that the challenger needs a play with at least five rounds and four alternations in order to win, i.e. to show that the states p and q in Figure 1 are not strongly bisimilar. Corresponding to a game in which a maximum of n alternations and k rounds are allowed, we can define a relation $\mathcal{R}(n, k)$ such that $(p, q) \in \mathcal{R}(n, k)$ iff the challenger *cannot* distinguish between p and q by playing a game where a maximum of k rounds and n alternations are allowed. The strong bisimulation game is thus $\mathcal{R}(\infty, \infty)$. Also it is easy to see that if $(p, q) \in \mathcal{R}(n, k)$, then $(p, q) \in \mathcal{R}(n + 1, k)$ and $(p, q) \in \mathcal{R}(n, k + 1)$.

3 Architecture of ReLTS

ReLTS is written in Ocaml and it uses the Ocamlgraph library [6] for representing LTS. It uses the dot format for specifying the input LTSs. A strategy of

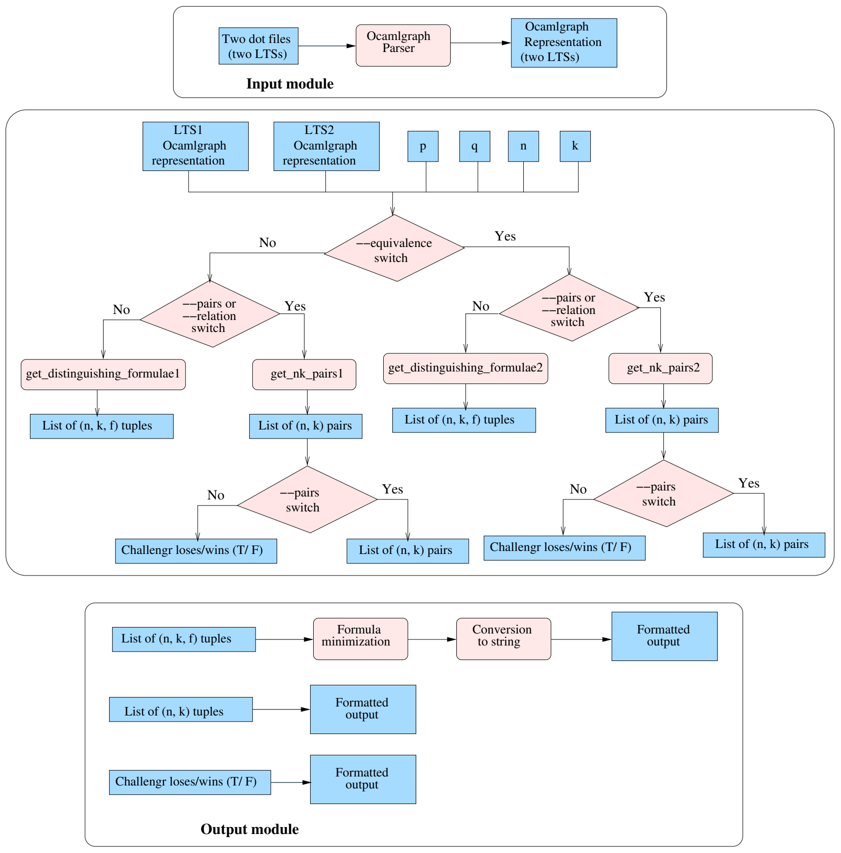

Fig. 2. Components of ReLTS

a player of the game is a sequence of moves made by it. 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 `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. *no_table* stores pairs of states that have been shown to be unrelated. *yes_table* stores the pairs of states seen so far that do not belong to the *no_table*. The

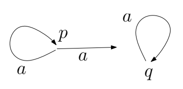

Fig. 3. A simple example

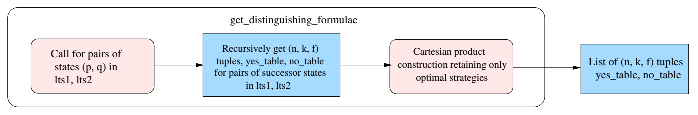
get_distinguishing_formulae
Call for pairs of states (p, q) in lts1, lts2
Recursively get (n, k, f) tuples, yes_table, no_table for pairs of successor states in lts1, lts2
Cartesian product construction retaining only optimal strategies
List of (n, k, f) tuples yes_table, no_table

Fig. 4. Internals of `get_distinguishing_formula`

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.

Figure 2 shows the components and their interaction during an execution of the tool. The command line interface provides a number of options for customising the output from `get_distinguishing_formulae`. 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. The option `--pairs` displays the n and the k corresponding to the optimal strategies, and the option `--relation` further reduces the output to indicating whether or not the relation holds. In both these cases, evaluation of distinguishing formulae is omitted when not requested in the output.

3.1 Example

We refer to Figure 3. 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 generated is $\langle a \rangle [a] \mathtt{ff}$.

Evaluation of time abstracted relations requires the use of reltool. For the user's convenience, a wrapper shell script ta_wrapper.sh is distributed with ReLTS.

4 Algorithm

We describe below the algorithm implemented in ReLTS to check the relation parameterised over n and k between two states p and q. Though partition refinement algorithms [9] are known for bisimulation relation, such algorithms are not known to exist for the generalized relations involving restricting both the number of rounds and the number of alternations and producing distinguishing formula related to optimal strategies.

Number of states	Acyclic LTS	Cyclic LTS
63	0.041	0.1
127	0.459	0.5
255	5.1	8.2
511	79.3	123.4

Table 1. All durations in seconds

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. $merge(L_{p',q_i}, L_{q_i,p'}) = L_{p',q_i} \cup \{(n_A + 1, k_A, \neg f_A) \,|\, (n_A, k_A, f_A) \in L_{q_i,p'}\}$. 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 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 $f_{A_i} \wedge f_{A_j}$.

As a convention, the distinguishing formula generated is satisfied by the state in the LTS chosen by the challenger in the first round. Algorithm 1 given in the appendix shows a pseudocode for the function *get_distinguishing_formula*.

5 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. The number of alternations was restricted to zero. As an indication, the memory usage was 29.9 MB for cyclic LTS with 511 states. The number of rounds was unrestricted in all experiments.

6 Availability and Possible Evolution

The tool can be downloaded from the project homepage, `http://airbornemihir.github.io/lts_reltool/`.
For the generation of zone graphs, reltool is required, which can be obtained from its homepage `http://github.com/airbornemihir/bachelor-thesis`.
The game playing strategy for timed relations provides a unifying approach to decide several timed preorders, bisimulations and prebisimulations, which can be explored in future work.[8].

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. Gérard Boudol, Valérie Roy, Robert de Simone, and Didier Vergamini. Process calculi, from theory to practice: Verification tools. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 1–10, London, UK, 1990. Springer-Verlag.
3. X. Chen and Y. Deng. Game characterizations of process equivalences. In *Proceedings of APLAS*, pages 107–121, 2008.
4. Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15:36–72, January 1993.
5. Rance Cleaveland and Steve Sims. The ncsu concurrency workbench. In *Proceedings of the 8th International Conference on Computer Aided Verification*, pages 394–397, London, UK, 1996. Springer-Verlag.
6. 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.
7. Jean-Claude Fernandez and Laurent Mournier. A tool set for deciding behavioral equivalences. In *Proceedings of the 2Nd International Conference on Concurrency Theory*, CONCUR '91, pages 23–42, London, UK, 1991. Springer-Verlag.
8. 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*, volume 120 of *EPTCS*, pages 47–62, 2013.
9. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
10. Valérie Roy and Robert de Simone. Auto/autograph. In *Proceedings of the 2Nd International Workshop on Computer Aided Verification*, CAV '90, pages 65–75, London, UK, 1991. Springer-Verlag.
11. Perdita Stevens. A practical introduction to games, infinity and the edinburgh concurrency workbench. In *Proceedings of FIW*, pages 35–36. IOS Press, 2005.
12. C. Stirling. Local model checking games. In *Proceedings of CONCUR*, pages 1–11, 1995.
13. Wolfgang Thomas. On the ehrenfeucht-fraïssé game in theoretical computer science. In *TAPSOFT*, pages 559–568, 1993.
14. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18:25–68, 2001.

Algorithm 1 The algorithm in get_distinguishing_formula.

Input: $lts1$ and $lts2$: LTSs, p and q: their respective initial states, n and k: maximum numbers of alternations and rounds allowed in the game, respectively, yes_table and no_table: dynamic programming data structures

Output: A list of winning strategies if the two states are not related

```
1: $ls' := []$
2: for all $(a, p')$ such that $p \xrightarrow{a} p'$ do
3:     for all $q'$ such that $q \xrightarrow{a} q'$ do
4:         if $(p', q', n, k-1) \notin yes\_table$ and $k > 0$ then
5:             $(f', n', k') = get\_distinguishing\_formula(lts1, lts2, p', q', n, k-1, yes\_table, no\_table)$ ▷ n', k': number of alternations and number of rounds in the play that the challenger actually needed to win the game from $p'$, $q'$
6:             if $f' = null$ then                ▷ $q'$ is a match for $p'$.
7:                 break
8:             end if
9:             if $(f', n', k')$ is a better strategy than some existing strategy for $(p', q')$ then
10:                $ls' := ls' :: (f', n', k', p', q')$
11:                Add the entry $(f', n', k', p', q')$ to $no\_table$, removing all other strategies for $(p', q')$ which are worse than this.
12:            end if
13:        end if
14:    end for
15: end for
16: for all $(a, q')$ such that $q \xrightarrow{a} q'$ do
17:     for all $p'$ such that $p \xrightarrow{a} p'$ do
18:         if $(p', q', n-1, k-1) \notin yes\_table$ and $n > 0$ and $k > 0$ then
19:             $(f', n', k') = get\_distinguishing\_formula(lts2, lts1, q', p', n-1, k-1, yes\_table, no\_table)$ ▷ $n'$, $k'$: number of alternations and number of rounds in the play that the challenger actually needed to win the game from $p'$, $q'$
20:             if $f' = null$ then                ▷ $p'$ is a match for $q'$.
21:                 break
22:             end if
23:             if $(f', n', k')$ is a better strategy than some existing strategy for $(p', q')$ then
24:                $ls' := ls' :: (f', n', k', p', q')$
25:                Add the entry $(f', n', k', p', q')$ to $no\_table$, removing all other strategies for $(p', q')$ which are worse than this.
26:            end if
27:        end if
28:    end for
29: end for
30: if $f' = null$ then
31:     return (null, -1, -1)                ▷ $p$ and $q$ are strongly bisimilar
32:     add $(p, q, n, k)$ to $yes\_table$
33: else
34:     Construct $ls$ from the best set of strategies in $ls'$
35:     return $ls$
36: end if
```