1 Introduction

This document describes the algorithm for branching bisimulation reduction, including the algorithm that is used to generate counterexamples.

2 A partitioner for branching bisimulation

The partitioner for branching bisimulation calculates whether states are bisimilar, branching bisimilar and stuttering preserving branching bisimilar. It gets a state space and divides it into a number of non-intersecting subsets of states called blocks. All states in a block are bisimilar and have the same block index. Using this index it is straightforward to calculate whether two states are equivalent (they have the same index) or to construct the state space modulo this equivalence.

The algorithm works exactly as described in [2]. As a preprocessing step for (divergence preserving) branching bisimulation all states that are strongly connected via internal transitions are replaced by a single state. In case of divergence preserving branching bisimulation this state gets a tau loop. In case of ordinary branching bisimulation, there will not be a tau loop.

Then the algorithm for branching bisimulation is started. The state space is partitioned into blocks. Initially, all states are put in one block. Repeatedly, a block is split in two blocks until the partitioning has become stable. For details see [2].

Furthermore, there is an option to obtain counter formulas for two non bisimilar state. The algorithm for this is inspired by [1, 3].

Given two non bisimilar states $s, t \in S$, where S is the set of all states. A *distinguishing formula* is a formula ϕ in Hennessy-Milner logic such that $s \models \phi$ and $t \not\models \phi$. For branching bisimulation there is always a distinguishing formula in the Hennessy-Milner logic extended with the regular $\langle \tau^* \rangle$, and using the regular formula $\langle \tau + nil \rangle \phi := \langle \tau \rangle \phi \vee \phi$.

Following [4, 5] we implement the computation of minimal depth distinguishing formulas. In the implementation two things can be noted different from the references with the theoretical background.

Filtering There is one post-processing step we call backwards filtering. It deals with the following scenario.

Consider the transition systems depicted in Figure 1. We consider the scenario of computing a distinguishing formula $\phi(s, t)$ for the states s and t. By the partitioning algorithm we know that the transition $s \xrightarrow{a} s_1$ is a distinguishing observation. The algorithm has to recursively find the distinguishing formula ϕ' such that $s_1 \in [\![\phi']\!]$ and $t_1, t_2 \notin [\![\phi']\!]$. This way $s \in [\![\langle a \rangle \phi']\!]$ and $t \notin [\![\langle a \rangle \phi']\!]$.

The algorithm starts by recursively computing a distinguishing formula for s_1 and t_1. It might compute $\phi(s_1, t_1) = \langle a \rangle \text{true}$. Since $t_2 \in [\![\langle a \rangle \text{true}]\!]$, it also computes the distinguishing formula $\phi(s_1, t_2) = \langle b \rangle \text{true}$. This results in the formula $\phi(s, t) = \langle a \rangle (\langle a \rangle \text{true} \wedge \langle b \rangle \text{true})$. We see that the addition of the conjunct

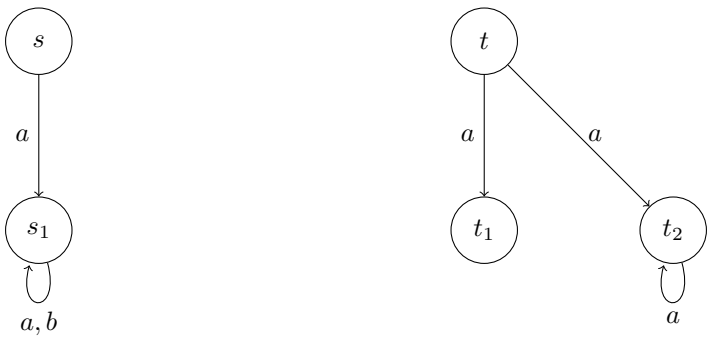

Figure 1: Two LTSs with initial state s and t respectively.

$\phi(s_1, t_2) = \langle b \rangle \text{true}$ made the first conjunct $\phi(s_1, t_1)$ obsolete (since $t_1 \notin [\![\langle b \rangle \text{true}]\!]$). This scenario is very hard to prevent a priori.

To prevent a distinguishing formula with unnecessary conjuncts each conjunct is reconsidered in a FIFO style. We compute the semantics of all other conjuncts and see if it still achieves the goal. If this is the case, we can safely remove that specific conjunct.

Minimal depth partitioning for branching bisimulation According to [5] we compute the partitions conform the correct k-depth relations. Given the partition π_i on level i we want to construct π_{i+1} such that it stable w.r.t. the following signatures.

$$sig(s, \pi_i) := \{(B, a, B') \mid s \xrightarrow{\tau^*} s' \xrightarrow{a} s'', \text{where } s' \in B \in \pi_i, s'' \in B' \in \pi_i \text{ and } (B \neq B' \vee a \neq \tau)\}$$

Performing this partitioning step efficiently is not obvious. We implemented the following algorithm. This algorithm only works since we removed strongly connected τ components in the preprocessing, hence there are no τ cycles.

1: $frontier := \{s \mid s \overset{\tau}{\not\longrightarrow}\}$
2: $done := \emptyset$
3: **while** $frontier \neq \emptyset$ **do**
4: $\quad s := frontier.pop\_front()$
5: $\quad signature := \{(B, a, B') \mid s \xrightarrow{a} s' \text{ s.t. } s \in B, s' \in B', \text{ and } B, B' \in \pi_i \text{ and } (a \neq \tau \vee B \neq B')\}$
6: $\quad$ **for all** $s \xrightarrow{\tau} s'$ **do**
7: $\quad\quad signature := signature \cup sigs[s']$
8: $\quad$ **end for**
9: $\quad sigs[s] := signature$
10: $\quad done := done \cup \{s\}$
11: $\quad$ **for all** $s_p \xrightarrow{\tau} s$ **do**
12: $\quad\quad$ **if** $\{s' \mid s_p \xrightarrow{\tau} s'\} \setminus done = \emptyset$ **then**
13: $\quad\quad\quad frontier.push\_back(s_p)$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end while**

References

[1] R. Cleaveland. On automatically explaining bisimulation inequivalence. In E.M. Clarke and R.P. Kurshan, editors, *Computer Aided Verification (CAV'90)*, volume 531 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 364–372, 1990.

[2] J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M.S. Paterson, editor, Proceedings 17th ICALP, Warwick, volume 443 of Lecture Notes in Computer Science, pages 626-638. Springer-Verlag, 1990.

[3] H. Korver. Computing distinguishing formulas for branching bisimulation. In K.G. Larsen and A. Skou, editors, *Computer Aided Verification (CAV'91)*, volume 575 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 13–23, 1991.

[4] J.J.M. Martens and J.F. Groote. Computing Minimal Distinguishing Hennessy-Milner Formulas is NP-Hard, but Variants are Tractable. In G.-A. P'erez and J.-F. Raskin, editors, *CONCUR '23*, Volume 279 of *LIPIcs*, Schloss Dagstuhl — Leibniz-Zentrum für Informatik, pages 32:1–32:17, 2023.

[5] J.J.M. Martens and J.F. Groote. Minimal Depth Distinguishing Formulas without Until for Branching Bisimulation. to appear in LNCS 2024