Notes on Bisimulation

Emanuele D'Osualdo

February 8, 2017

1 Bisimulation and its Characterisations

1.1 Labelled Transition Systems

Definition 1 (LTS)**.** A *Labelled Transition System* (LTS) over a set of *actions* Act, is a pair $(S, \to)$ where

- S is a set of configurations (it can be infinite)
- $\to \subseteq S \times Act \times S$ is the transition relation

We write $s_1 \xrightarrow{\alpha} s_2$ when $(s_1, \alpha, s_2) \in \to$.

Note that there is a conceptual difference between LTS and finite automata. When Act and S are finite, both definitions describe a graph with labelled edges. Conceptually, however, automata are the *syntax* of a system, with usual semantics expressed in terms of *languages*; LTS are instead objects that are used to express the *semantics* of a system. The fact that the LTS semantics of a finite automaton coincides with the structure of the automaton itself is a trivial corner case which should not confuse you.

While in general infinitely-branching LTS can be considered, for our lecture we only consider finitely-branching LTS, i.e. LTS $(S, \to)$ in which for every state $s \in S$, there are only finitely many states s' and actions α such that $s \xrightarrow{\alpha} s'$.

1.2 Bisimulations

Definition 2 (Bisimulation)**.** Let $(S, \to)$ be an LTS over Act. A binary relation $\mathcal{B}$ over S is called a *(strong) simulation* if for every $s, s', t \in S$ and $\alpha \in Act$ such that $s \xrightarrow{\alpha} s'$ and $s \mathcal{B} t$, there exists a $t' \in S$ such that $t \xrightarrow{\alpha} t'$ and $s' \mathcal{B} t'$.

A binary relation $\mathcal{B}$ over S is a *(strong) bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are (strong) simulations.

We say s *is simulated by* t (written $s \lesssim t$) if there exists a simulation relation containing the pair (s, t).

Similarly, we say s is bisimilar to t (written $s \sim t$) if there exists a bisimulation relation containing (s, t).

When we compare two distinct LTS $(S_1, \to_1)$, $(S_2, \to_2)$ using (bi)simulations, we implicitly instantiate the above definition on the LTS $(S_1 \cup S_2, \to_1 \cup \to_2)$. An *initialised* LTS is a triple $(S, \to, s_0)$ where $(S, \to)$ is an LTS and $s_0 \in S$ is its initial state. We say two initialised LTS are bisimilar if their initial states are bisimilar.

In this note we will only consider strong bisimulation, and we simply refer to it as bisimulation.

Theorem 1. *The relation $\sim$ is an equivalence relation.*

Proof. We need to check that $\sim$ is reflexive, symmetric and transitive. Reflexivity is obvious since equality is a bisimulation. Symmetry follows from the fact that if $\mathcal{B}$ is a bisimulation, $\mathcal{B}^{-1}$ is a bisimulation too. For transitivity it is sufficient to check that if $\mathcal{B}_1$ and $\mathcal{B}_2$ are both bisimulations, then the relation $\mathcal{B}_1\mathcal{B}_2 := \{(s, s') \mid \exists t : s \ \mathcal{B}_1 \ t \wedge t \ \mathcal{B}_2 \ s'\}$ is a bisimulation too. □

Theorem 2. *The relation $\sim$ is a bisimulation.*

Proof. Assume $s \sim t$, then there is a bisimulation $\mathcal{B}$ such that $s \ \mathcal{B} \ t$ and for all $s' \in S$ and $\alpha \in Act$ such that $s \xrightarrow{\alpha} s'$, there exists a $t' \in S$ such that $t \xrightarrow{\alpha} t'$ and $s' \ \mathcal{B} \ t'$, which implies $s' \sim t'$. The other direction is proven by using $\mathcal{B}^{-1}$. □

We now present two characterisations of bisimulation that are both very useful for proofs involving bisimulation and for computing bisimulation relations.

1.3 Stratification of Bisimulation

The first characterisation, which only works for finitely-branching LTS, is called *stratification*.

Definition 3 (Stratification)**.** Fix an LTS $(S, \to)$. For $n \in \mathbb{N}$, we define the binary relation $\sim_n$ over S, inductively as follows:

- $s \sim_0 t$ for every $s, t \in S$
- $s \sim_{n+1} t$ if for every $\alpha \in Act$:
  - i) for every $s' \in S$ such that $s \xrightarrow{\alpha} s'$, there exists a $t' \in S$ such that $t \xrightarrow{\alpha} t'$ and $s' \sim_n t'$;
  - ii) for every $t' \in S$ such that $t \xrightarrow{\alpha} t'$, there exists a $s' \in S$ such that $s \xrightarrow{\alpha} s'$ and $t' \sim_n s'$;

We say s is n-bisimilar to t when $s \sim_n t$.

Proposition 3. *For every n, the relation $\sim_n$ is an equivalence relation.*

Proof. Easy induction on n. □

Theorem 4. *Fix a finitely-branching LTS $(S, \to)$. For $s, t \in S$, $s \sim t$ if and only if $\forall n \in \mathbb{N} : s \sim_n t$. Equivalently, $\sim = \bigcap_{n \in \mathbb{N}} \sim_n$.*

For a proof, see Problem 3, Exercise Sheet 11.

Actually, Theorem 4 holds also in the weaker assumption of *image-finiteness*, instead of finite branching. An LTS over Act is *image-finite* when for every $\alpha \in Act$, the relation $\xrightarrow{\alpha}$ is finitely branching.

Note that $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \ldots$ and the expression $\bigcap_{n\in\mathbb{N}} \sim_n$ is the greatest fixpoint of the relation transformer induced by Definition 3, by Kleene's theorem.

This characterisation readily provides an algorithm for computing the bisimulation relation over a finite LTS: as the following theorem shows, the sequence the stratified relations saturates after at most n steps for an n-configurations LTS.

Theorem 5. *In a LTS with n configurations, $\sim = \sim_n = \sim_{n-1}$.*

Proof. Let $(S, \to)$ be an LTS with $|S| = n$. Observe that, if $\sim_k = \sim_{k+1}$ for some $k \in \mathbb{N}$, then $\sim_k = \sim_j$ for all $j \geq k$. This means that $\sim_n = \sim_{n-1}$ implies $\sim = \sim_n$ so we only need to prove the former equation. Consider the sequence of relations $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \ldots$; since each of them is an equivalence relation, we can count the number of induced equivalence classes for each relation. The number of classes is non-decreasing along the sequence and is 1 for $\sim_0$. Surely, at any point there cannot be more than n classes and if $\sim_k$ and $\sim_{k+1}$ have the same number of classes then they are the same relation. Now, towards a contradiction, assume that there is no $k < n$ such that $\sim_k = \sim_{k+1}$. Then we have that $\sim_{n-1}$ has n classes and $\sim_n$ would need to have at least $n + 1$ classes which is a contradiction. □

1.4 Bisimulation Games

The second characterisation uses two-player games that are a variant of Ehrenfeucht-Fraïssé games (see the Advanced Automata Theory lecture notes for background).

Definition 4 (Bisimulation Game)**.** Fix two LTS $(S_0, \to_0)$, $(S_1, \to_1)$. A *bisimulation game* is a game played by Spoiler against Duplicator. The game starts from two positions $s \in S_0$ and $t \in S_1$ and proceeds in turns. At every turn with current positions $s_0 \in S_0$, $s_1 \in S_1$:

1. Spoiler moves by choosing one of the two LTS, say S_i; from S_i it selects a transition $s_i \xrightarrow{\alpha} t_i$.
2. Duplicator responds by finding a transition $s_j \xrightarrow{\alpha} t_j$ in the other LTS S_j $(j = 1 - i)$.

The next turn is played from t_i, t_j.

A player wins if the other player cannot move in some turn. An infinite play is won by Duplicator.

A *strategy* for a player is a partial function from the game's history so far to the choice the player needs to make in the current move. A strategy for player P is *winning* from s, t if P always wins the game from s, t by following the strategy.

Proposition 6. *i) Winning strategies for the bisimulation game are positional, i.e. can be described by functions from the current positions to the player's move. ii) Every bisimulation game is determined, i.e. either one or the other player has a winning strategy.*

Theorem 7. *Duplicator has a winning strategy from s, t if and only if $s \sim t$. More precisely, Duplicator has a winning strategy for the k-turn game from s, t if and only if $s \sim_k t$.*

2 Bisimulation for Petri Nets

2.1 Decidability issues

It is natural to ask whether bisimulation is decidable for some class of models of computation. For finite-state systems, we can compute bisimilarity using Theorem 5. For LTS representing the reachability graph of Turing machines (with only action τ), bisimulation is undecidable. This is easily shown by reducing the halting problem to bisimilarity of the initial state of a Turing machine and the state Ω of the LTS consisting of the transition $\Omega \xrightarrow{\tau} \Omega$.

The question we address in this section is thus: is there an infinite-state model for which bisimulation is decidable? We consider the case of LTS generated by Petri Nets.

2.2 Labelled Petri Nets

Definition 5 (Labelled Petri net)**.** Fix a set of actions Act. A *labelled Petri net* is a pair (N, λ) where $N = (S, T, W)$ is a Petri net and $\lambda \colon T \to Act$ is a function that assigns labels (actions) to the transitions of N.

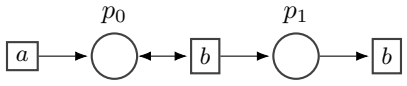
The *LTS induced by* (N, λ) is the reachability graph of N where the edges are relabelled using λ, i.e. for two markings $M, M' \in \mathbb{N}^{|S|}$, $M \xrightarrow{\alpha} M'$ if in the firing relation $M \xrightarrow{t} M'$ with $\lambda(t) = \alpha$. A labelled Petri net is marked if the underlaying net is marked.

Naturally, λ can assign the same action to different transitions, making it impossible for the environment to distinguish them.

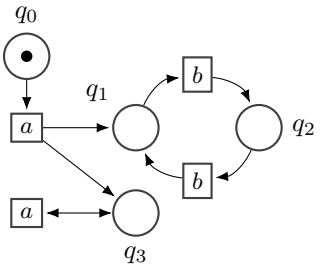
For two marked labelled Petri nets (N_1, λ_1) and (N_2, λ_2), we write $(N_1, \lambda_1) \sim (N_2, \lambda_2)$ to mean that the two LTS induced by the nets, restricted to the marking reachable from the respective initial markings, are strongly bisimilar.

Example 1. Consider the two Petri nets:

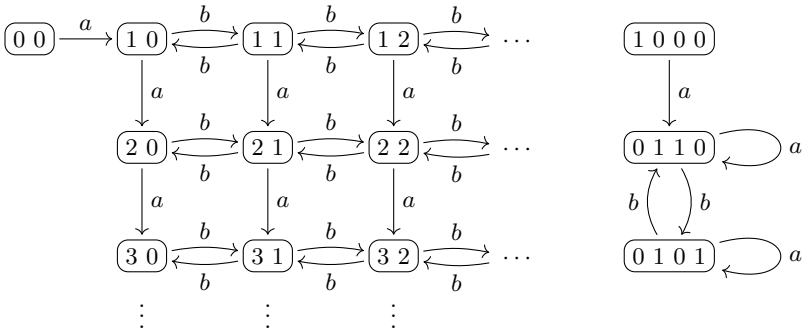
$(N_1, \lambda_1) =$



$(N_2, \lambda_2) =$



Their LTS are respectively



It is easy to show that the two initial markings are strongly bisimilar, thus $(N_1, \lambda_1) \sim (N_1, \lambda_1)$.

2.3 Undecidability of Strong Bisimulation for Petri Nets

The material in this section is adapted from:

> **P. Jančar, 1995.**
> *Undecidability of bisimilarity for Petri nets and some related problems.*
> Theoretical Computer Science. Volume 148, Issue 2.

Theorem 8. *Given two marked labelled Petri Nets* (N_1, λ_1) *and* (N_2, λ_2)*, it is undecidable whether* $(N_1, \lambda_1) \sim (N_2, \lambda_2)$*.*

Proof. We define two PN $(N^f_{\mathrm{CM}}, \lambda)$ and $(N^{\bar{f}}_{\mathrm{CM}}, \lambda)$ from a 2-counter machine CM such that

$$(N^f_{\mathrm{CM}}, \lambda) \sim (N^{\bar{f}}_{\mathrm{CM}}, \lambda) \quad \Longleftrightarrow \quad \text{CM does not halt.} \qquad (*)$$

A counter machine is a sequence of instructions

$$0\colon I_0; \quad 1\colon I_1; \quad \cdots \quad h-1\colon I_{h-1};$$

for some $h \in \mathbb{N}$.

Each instruction I_i is of one of the two forms

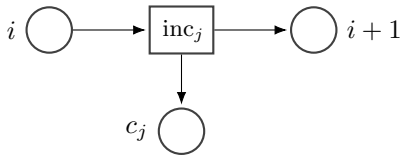
- c_j`++;` which has the effect of incrementing counter c_j (for $j \in \{0, 1\}$)
- `if(`c_j `> 0)` c_j`--; else goto` k`;` which has the effect of decrementing the counter c_j if it is strictly positive and continue with the next instruction, or jump to program location $k \in \{1, \ldots, h\}$ if the counter is zero.

We adopt the convention that jumping to program location h corresponds to halting.

From CM we construct the two identical nets $(N_{\mathrm{CM}}^{f}, \lambda)$ and $(N_{\mathrm{CM}}^{\bar{f}}, \lambda)$ which differ only for their initial marking M_0^f and $M_0^{\bar{f}}$ which we specify later. Their places, transitions and labelings are the same and are constructed as follows.

We have a place i for each program location of the CM $i \in \{1, \ldots, h\}$ (including the halting location h). These places will be mutually exclusive: they are all 1-bounded and no two places will have a token at the same time. We also have two unbounded places c_1 and c_2, one for each counter. Finally, we have two auxiliary dual places that we call f and $\bar{f}$ (for "$f$lag").

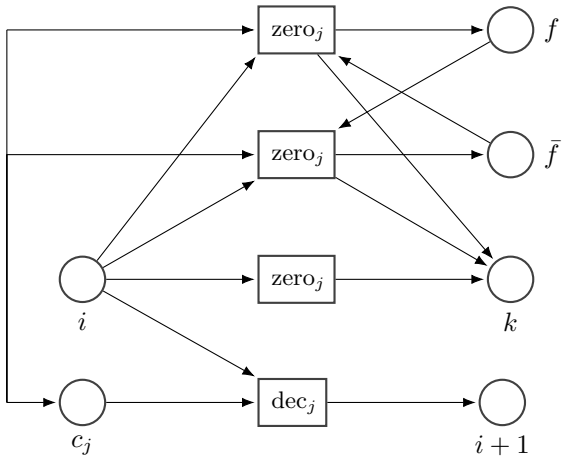
An increment instruction at location i, that is $i\colon c_j$`++`, is encoded as an inc_j-labelled transition that transfers a token from place i to $i + 1$ while inserting a token in place c_j:



A test-and-decrement instruction at location i,

$$i\colon \texttt{if(}c_j \texttt{ > 0) } c_j\texttt{--; else goto } k\texttt{;}$$

is implemented by a slightly more complicated gadget:

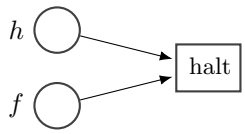


You can read this construction as follows. The two lower transitions are the

natural encoding of the instruction to Petri nets. This is usually called a "weak" encoding because it allows to jump to the k branch even when the counter is not zero, which we call "cheating". Note however that these two transitions are all we need to be able to simulate every "honest" (i.e. non-cheating) run of CM.

We added the two other zero_j transitions, which we call "definitely cheating": they can jump to the zero branch k only when the counter is non-zero (note the arcs in and from the counter), an intentional mistake. They have the side-effect of flipping the value of the flag.

Termination is encoded by a transition labelled with the halt action.



We add the place f to the preset of the halting transition to help us distinguish the behaviour of the two nets.

Finally, both initial markings M_0^f, $M_0^{\bar{f}}$ have a token in the place 0, encoding the first program location. Moreover, M_0^f has a token in f whereas $M_0^{\bar{f}}$ has a token in $\bar{f}$.

It remains to show the claim $(*)$:

$$(N_{\text{CM}}^f, \lambda) \sim (N_{\text{CM}}^{\bar{f}}, \lambda) \quad \iff \quad \text{CM does not halt.}$$

"$\Rightarrow$": Assume the CM does halt. Spoiler can win the game from M_0^f, $M_0^{\bar{f}}$ by always picking the honest transitions. Duplicator has no other choice but to pick the same transitions from $M_0^{\bar{f}}$. When the honest run terminates, Spoiler can pick the halt-transition but Duplicator cannot: the honest transitions do not change the tokens in $f, \bar{f}$ so the halt-transition is only enabled in N_{CM}^f.

"$\Leftarrow$": Assume the CM does not halt. Duplicator can always win from M_0^f, $M_0^{\bar{f}}$ by following this strategy:

- As long as Spoiler does not cheat, i.e. doesn't use a zero_j-transition when $c_j > 0$, Duplicator has no choice but to pick the same transitions. This is always possible. If Spoiler does never cheat, the game will be infinite and Duplicator wins.
- When Spoiler first cheats, Duplicator picks a zero_j-transition so that the next markings coincide, i.e. flip $f, \bar{f}$ if Spoiler does not flip $f, \bar{f}$ and do not flip $f, \bar{f}$ otherwise. From then on, Duplicator can always pick the same transitions as Spoiler. □

2.4 Decidability of Bisimulation with a finite LTS

The previous negative result means that we cannot algorithmically use strong bisimulation on Petri nets as a tool to prove correctess. However, when one of the two nets is finite-states (i.e. bounded), strong bisimulation becomes decidable. This can be useful to validate an infinite-state implementation against a finite-state specification.

The material of this section is adapted from

> **Petr Jančar, Javier Esparza, and Faron Moller, 1999.**
> *Petri nets and regular processes.* Journal of Computer and System Sciences. Volume 59, Issue 3.

Theorem 9. *Given a PN (N, λ_N) and a bounded PN (A, λ_A), it is decidable whether $(N, \lambda_N) \sim (A, \lambda_A)$.*

Proof. Let s_0 be the initial marking of A. Let $n = |\mathrm{Reach}_A(s_0)|$ be the (finite) size of the LTS for A from s_0. We first observe that for markings $M \in \mathrm{Reach}(N)$, $s \in \mathrm{Reach}_A(s_0)$

$$M \sim s \quad \iff \quad M \sim_n s \wedge \mathrm{Reach}_N(M) \subseteq B$$

where $B = \{M' \mid M' \sim_n s' \text{ for some } s' \in \mathrm{Reach}_A(s_0)\}$.

The left to right implication is trivial. For a proof of the other direction, consider the relation $\sim_n$ over A. Since there are n configurations in $\mathrm{Reach}_A(s_0)$, we have that $\sim_n = \sim_{n-1}$ by Theorem 5. Now if we consider $\sim_n$ over B, it cannot be composed of more than n classes, otherwise we would have two markings M_1, M_2 in B that are not n-bisimilar but are n-bisimilar to the same s, by the pidgeon principle. This contradicts transitivity of $\sim_n$. Note that the same does not apply to markings of N outside B. Overall, we get that $\sim_n = \sim_{n-1}$ over $B \cup \mathrm{Reach}_A(s_0)$. With this fact it is easy to check that $\sim_n$ is indeed a bisimulation over $\mathrm{Reach}_N(M) \cup \mathrm{Reach}_A(s_0)$.

Now, we need to show that we can decide $\boxed{1}$ $M \sim_n s$ and $\boxed{2}$ $\mathrm{Reach}_N(M) \subseteq B$. By an easy induction on n, $\boxed{1}$ is decidable.

As for $\boxed{2}$, we define

- the (finite!) set of n-bounded markings $\mathcal{L}_n := \{M \mid \forall p : M(p) \leq n\}$,
- $M\lfloor_n := \begin{cases} n & \text{if } M(p) \geq n \\ M(p) & \text{otherwise,} \end{cases}$
- ${}^n\!\upharpoonright L := \{N \mid N\lfloor_n = L\}$ for $L \in \mathcal{L}_n$.

Wlog., we assume that N is ordinary, i.e. has weights 1 or 0. Under this assumption it is easy to show that $M \sim_n M\lfloor_n$, see Exercise Sheet 11, Problem 1. From this we can infer that

$$M \in B \quad \iff \quad M\lfloor_n \in B \quad \iff \quad {}^n\!\upharpoonright (M\lfloor_n) \subseteq B.$$

Since $\mathcal{L}_n$ is finite, we have that B is a finite union $B = {}^n\!\upharpoonright L_1 \cup \cdots \cup {}^n\!\upharpoonright L_k$, and consequently its complement $\overline{B} = {}^n\!\upharpoonright L'_1 \cup \cdots \cup {}^n\!\upharpoonright L'_{k'}$ is a finite union too,

with $\mathcal{L}_n = \{L_1, \ldots, L_k, L'_1, \ldots, L'_{k'}\}$. Furthermore, the set $\{L'_1, \ldots, L'_{k'}\}$ can be computed by checking $L \sim_n s$ for each $L \in \mathcal{L}_n$ and $s \in \mathrm{Reach}_A(s_0)$.

Overall, we can decide $\mathrm{Reach}(M) \subseteq B$ by deciding whether $\mathrm{Reach}(M) \cap \overline{B} = \emptyset$, which in turn can be computed by checking if we can reach some $M' \in {}^{n}\uparrow L'_i$, for some $i \in \{1, \ldots, k'\}$.

The latter problem can be reduced to reachability of L'_i in N with transitions added that can delete tokens from any place p with $L'_i(p) = n$. Note that this is not a coverability problem since for the places p where $L'_i(p) < n$, we want the reachable marking to coincide with L'_i. □