Simulation Games

Motivation

There are at least two distinct purposes for which it is useful to compute simulation relationships between the states of automata.
Firstly, with the use of simulation relations it is possible to mimick the behaviour of another automaton and also to establish language containment among nondeterministic automata.
Secondly, simulation relations can be used to reduce the state space of an automaton by obtaining its quotient with respect to the equivalence relation underlying the simulation preorder.

4 Different kinds of simulations

In the following there are presented four different kinds of simulation games for a given Büchi automaton A = $\langle \Sigma, Q, q_I, \Delta, F \rangle$:

1) The ordinary simulation game, denoted $G_A^o(q_0, q_0')$,
2) The direct (strong) simulation game, denoted $G_A^{di}(q_0, q_0')$,
3) The delayed simulation game, denoted $G_A^{de}(q_0, q_0')$,
4) The fair simulation game, denoted $G_A^f(q_0, q_0')$.

Each of the games is played by two players, Spoiler and Duplicator, in rounds as follows. At the start, round 0, two pebbles, Red and Blue, are placed on the two vertices q_0 and q'_0. Then Spoiler chooses a transition $(q_i, a, q_{i+1}) \in \Delta$ and moves Red to q_{i+1}.
Duplicator, responding, must choose a transition $(q'_i, a, q'_{i+1}) \in \Delta$ and moves Blue to q'_{i+1}.
Here it can be the case that no a-transition starting from q'_i exists. In this case the game halts and Spoiler wins.

Either the game halts, in which case Spoiler wins, or the games produces two infinite runs: $\pi = q_0 a_0 q_1 a_1 q_2 \cdots$ and $\pi' = q'_0 a_0 q'_1 a_1 q'_2 \cdots$, built from the transitions taken by the two pebbles.
For each of the 4 simulation games there exist different rules to determine the winner:

1.) Ordinary simulation: Duplicator wins in any case as long as the game does not halt. (Fairness conditions are ignored)
2.) Direct simulation: Duplicator wins, iff, for all i, if $q_i \in F$, then also $q'_i \in F$
3.) Delayed simulation: Duplicator wins, iff, for all i, if $q_i \in F$, then there exists j ≥ i such that $q'_j \in F$
4.) Fair simulation: Duplicator wins iff there are infinitely many j such that $q'_j \in F$ or there are only finitely many i such that $q_i \in F$

A strategy for Duplicator in one of the 4 simulation games from above is a function: $f : (\Sigma Q)^+ \to Q$ which, given the history of the game (actually the choices of Spoiler) up to a certain point, determines the next move of Duplicator. A strategy f for Duplicator is a winning strategy if, no matter how Spoiler plays, Duplicator always wins.

Simulation Relation

A state q' ordinary, direct, delayed, fair simulates a state q if there is a winning strategy for Duplicator. The simulation relation, a reflexive, transitive relation (preorder or quasi-order), is denoted by $q \preceq_\star q'$, where * stands for one of the 4 simulations ordinary(o), direct(di), delayed(de) and fair(f).
The relations are ordered by containment:
$\preceq_{di} \subseteq \preceq_{de} \subseteq \preceq_f \subseteq \preceq_o$
For direct, delayed and fair holds: if $q \preceq_\star q'$ then $L(A[q]) \subseteq L(A[q'])$.

Bisimulation Relations

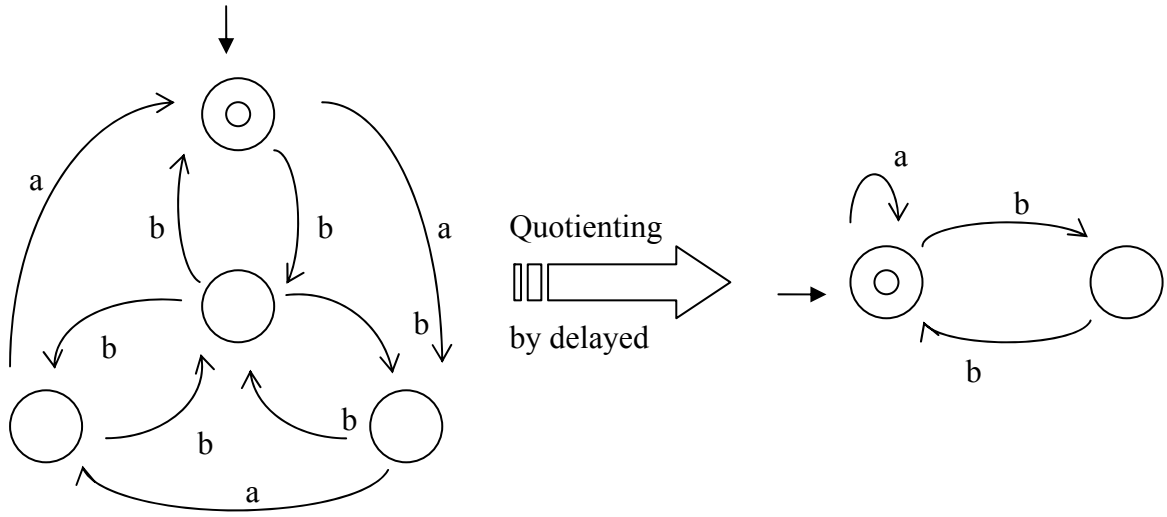
For all the mentioned simulations there are corresponding notions of bisimulation, defined via a modification of the simulation games.
The bisimulation game differs from the simulation game in that Spoiler gets to choose in each round which of the two pebbles, Red or Blue, to move and Duplicator has to respond with the move of the other pebble.
Bisimulations define the following equivalence relation containment:
$\approx^{bi}_{di} \subseteq \approx^{bi}_{de} \subseteq \approx^{bi}_{f} \subseteq \approx^{bi}_{o}$.
For bisimulations there are also winning rules for the bisimulation games that are similar to the rules for the 4 different simulation games:

1.) Ordinary: the same rules as for the ordinary simulation game
2.) Fair: If an accept state appears infinitely often on one of the two runs π and π', then an accept state must appear infinitely often on the other as well.
3.) Delayed: If an accept state is seen at position i of either run, then an accept state must be seen thereafter at some position j ≥ i of the other run.
4.) Direct: If an accept state is seen at position i of either run, it must be seen at position I of both runs.

Quotienting

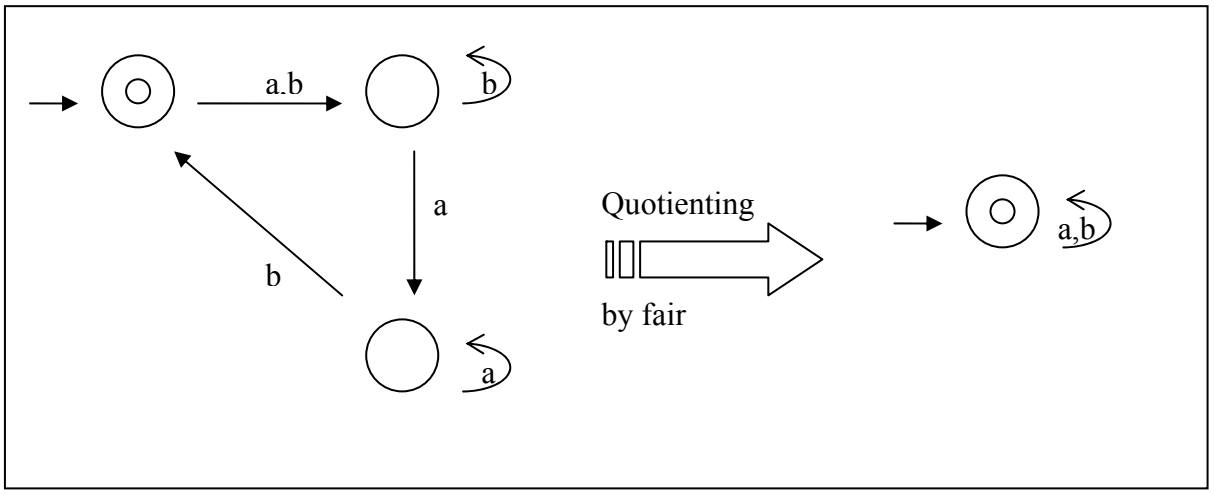
A major motivation for studying relations is state-space reduction, the basic idea being that states that simulate each other are merged leading to a reduction of the number of states. This process is usually called quotienting.
Quotienting with respect to delayed simulation preserves the recognized language, but this is not true for with fair simulation as we will see later on.
For a Büchi automaton A, and an equivalence relation ≈ on the states of A, let [q] denote the equivalence class of $q \in Q$ with respect to ≈.
The quotient of A with respect to ≈ is the automaton $A/\approx = \langle \Sigma, Q/\approx, \Delta_\approx, [q_I], F/\approx \rangle$, where $\Delta_\approx = \{([q], a, [q']) \mid \exists\, q_0 \in [q],\ q'_0 \in [q'], \text{ such that } (q_0, a, q'_0) \in \Delta\}$
For applying the simulation relations, corresponding to each simulation preorder, an equivalence relation $\approx_o, \approx_{di}, \approx_{de}, \approx_f$ is defined, where $q \approx_\star q'$, iff $q \preceq_\star q'$ and $q' \preceq_\star q$.
The quotient with respect to $\approx_{di}$, $\approx_{de}$ preserves the language of any automaton.

An example shows quotienting for delayed simulation. In this example both automata, the original one and the quotient, accept the same language.



The obtained quotient is smaller than the original automaton and accepts the same language as the first one. The states around the middle state are in one equivalence class, because one can stay there with infinitely many a's. With a b one comes to another state and there, if reading again another b, one comes to the first state. Whenever Spoilers makes an a-move, Duplicator can make one as well. If Spoiler then makes a b-move to the outside, Duplicator answers with a move to the upper accepting state. In this case Duplicator visits infinitely often an accepting state and everything is fine. Thus the quotient automaton has only two states.

The quotient with respect to $\approx_o$, $\approx_f$ does not preserve the language of any automaton, which is shown in the example below:



Quotienting is done in this example via the equivalence relation $q \approx_f q'$ that holds for all states q, q' from the set Q. The left automaton is accepting words with a and b infinitely alternating. The right automaton accepts words like a^ω or b^ω, but the original automaton doesn't.

So here it can be seen, that the quotient of the original automaton with respect to $\approx_f$ does not preserve the language of the original automaton.

Parity Games

A parity game graph $G = \langle V_0, V_1, E, p \rangle$ has two disjoint sets of vertices V_0 and V_1, whose union is V. It also has an edge set $E \subseteq V \times V$ and a priority function $p\colon V \to \{0, \ldots, d-1\}$ that assigns a priority to each vertex.
Such a parity game is played by two players, Zero and One. The play starts by placing a pebble on vertex v_0. Thereafter, the pebble is moved according to the following rule:
With the pebble currently on a vertex v_i and $v_i \in V_0$ (V_1), Zero (One, respectively) plays and moves the pebble to a neighbour v_{i+1}, such that $(v_i, v_{i+1}) \in E$.
If ever the rule above cannot be applied, i.e., someone can't move because there are no outgoing edges, the game ends and the player who cannot move loses.
Otherwise, the game goes on forever, and defines a path $\pi = v_0 v_1 v_2 \cdots$, called a play of the game.
The winner of the game is determined as follows. Let k_π be the minimum priority that occurs infinitely often in the play π; Zero wins if k_π is even, whereas One wins if k_π is odd.

Now the game graphs G^*_A are built, following the same general pattern, with some minor alterations. The first game graph will be $G^f_A = \langle V^f_0, V^f_1, E^f_A, p^f_A \rangle$ with only three priorities (0,1,2).
Formally G^f_A is defined by

The set of vertices for Zero:
$V^f_0 = \{v_{(q,q',a)} \mid q, q' \in Q \wedge \exists q''((q'', a, q) \in \Delta)\}$,
This set of vertices contains all vertices, where Spoiler has moved the turn before and it is now Duplicator's turn to imitate the label on the 3rd position of the tupel, because Spoiler has taken this label the move before. So this label remembers the move of Spoiler, q is the position of the Spoiler-pebble and q' is the position of the Duplicator-pebble.

The set of vertices for One:
$V^f_1 = \{v_{(q,q')} \mid q, q' \in Q\}$,
This set of vertices contains all vertices, where Duplicator has moved the turn before and it is now Spoiler's turn. Here is no 3rd position in the tupel, because Spoiler does not have to imitate a run move from Duplicator, that has to answer to the moves of Spoiler.

The set of the edges for Zero and One:
$E^f_A = \{(v_{(q_1,q_1',a)}, v_{(q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta\} \cup \{(v_{(q_1,q_1')}, v_{(q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta\}$,
The moves of the first set are the moves of Duplicator. Duplicator is in state $v(q_1,q_1',a)$ and has to take a transition with a label a to move from state q_1' to a state q_2' if there exists the transition (q_1', a, q_2'). Only the state of Duplicator changes here, the Spoiler's state q_1 remains the same.
The second set contains all the moves of Spoiler. Spoiler s in state $v(q_1,q_1')$ and can choose any transition (q_1,a,q_2) form the transition set to move his pebble to another state. In this case only the Spoiler's state is changed.

The priority function:

$$p_A^f(v) = \begin{cases} 0, & \text{if } (v = v_{(q,q',a)} \text{ or } v = v_{(q,q')}) \text{ and } q' \in F \\ 1, & \text{if } v = v_{(q,q')},\ q \in F, \text{ and } q' \notin F, \\ 2, & \text{otherwise.} \end{cases}$$

This function means, that if neither Spoiler nor Duplicator have infinitely often an accepting state in their runs that the minimum parity number that occurs infinitely often is the 2. In this case Duplicator wins.
If Spoiler has seen infinitely often an accept state but not Duplicator, the minimum parity number that occurs infinitely often is the 1 and so in this case Spoiler wins.
Last but not least, if Duplicator sees infinitely often an accepting state no matter what Spoiler does, the minimum parity number occurring is the 0 and here also Duplicator wins.

The graph G^f_A can be modified to obtain G^o_A and G^{di}_A, both of which require only trivial modification to G^f_A. The parity game graph G^o_A is exactly the same as G^f_A, except that all nodes will receive priority 0. This means, that Duplicator wins as long as he can mimick the moves of Spoiler and he only loses, if he comes to a dead-end and can't move anymore.

The parity game graph G^{di}_A is just like G^o_A, meaning every vertex has priority 0, but some edges are eliminated:

$$E_A^{di} = E_A^f \setminus \{(v, v_{(q_1,q_1')}) \mid q_1 \in F \wedge q_1' \notin F\}$$

Finally to define G^{de}_A the game graph has to be modified somewhat more. For each vertex of G^f_A there will be at most two corresponding vertices in G^{de}_A:

$$V_0^{de} = \{v_{(b,q,q',a)} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge \exists q''((q'', a, q) \in \Delta)\}$$
$$V_1^{de} = \{v_{(b,q,q')} \mid q, q' \in Q \wedge b \in \{0,1\} \wedge (q' \in F \rightarrow b = 0)\} \ .$$

The extra bit b encodes whether or not, thus far in the simulation game, the Red pebble was witnessed an accept state without Blue having witnessed one since then.
The edges of G^{de}_A are as follows:

$$\begin{aligned} E_A^{de} = &\{(v_{(b,q_1,q_1',a)}, v_{(b,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \notin F\} \\ &\cup \{(v_{(b,q_1,q_1',a)}, v_{(0,q_1,q_2')}) \mid (q_1', a, q_2') \in \Delta \wedge q_2' \in F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(b,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \notin F\} \\ &\cup \{(v_{(b,q_1,q_1')}, v_{(1,q_2,q_1',a)}) \mid (q_1, a, q_2) \in \Delta \wedge q_2 \in F\} \end{aligned}$$

Last but not least the priority function of G^{de}_A is described as:

$$p_A^{de}(v) = \begin{cases} b, & \text{if } v = v_{(b,q,q')}, \\ 2, & \text{if } v \in V_0. \end{cases}$$

Here, priority 1 will be assigned to only those vertices in V_1 that signify that an "unmatched" accept has been encountered by Red.

References

- Carsten Fritz, Thomas Wilke: Simulation Relations for Alternating Parity Automata and Parity Games. *DLT 2006, LNCS 4036, pp. 59-70*, Springer-Verlag (2006)

- Kousha Etessami, Thomas Wilke, Rebecca A. Schuller: Fair Simulation Relations, Parity Games and State Space Reduction for Büchi Automata. *ICALP 2001, LNCS 2076, pp. 694-707*, Springer-Verlag (2001)

- Carsten Fritz: Simulation-Based Simplification of omega-Automata. PhD thesis, Technische Fakultät der Christian Albrecht Universität zu Kiel (2005) available at http:/e-diss.uni-kiel.de/diss_1644/