CDM

Bisimulations

Klaus Sutner

Carnegie Mellon University

1 Bisimulations

By a transition system $\mathcal{T}$ we mean an edge-labeled digraph (where we allow multiple edges and loops). More precisely, consider a set Q of vertices, E of edges and alphabet Σ together with three maps

$$\begin{aligned} \mathsf{src}, \mathsf{trg} &: E \to Q \\ \mathsf{lab} &: E \to \Sigma \end{aligned}$$

We will always assume that distinct edges differ either in source, target or label. Hence we can think of E as a ternary relation: $E \subseteq Q \times \Sigma \times Q$, the transition relation of $\mathcal{T}$. We call $(\mathsf{src}(e), \mathsf{lab}(e), \mathsf{trg}(e))$ a transition and often write

$$p \xrightarrow{a} q \qquad \text{for} \qquad \mathsf{src}(e) = p, \mathsf{lab}(e) = a, \mathsf{trg}(e) = q$$

We are mostly interested in the case where Q is finite, but the definitions make perfect sense in general.

Intuitively we can think of

- Q as a set of (process) states,
- Σ as a set of permissible actions, and
- $\rightarrow$ as the specification of possible transitions.

It is often necessary to filter out transitions labeled by $a \in \Sigma$:
$E_a = \{\, p \rightarrow q \mid p \xrightarrow{a} q \in E \,\}$.

Hence, we may write a transition system as

$$\mathcal{T} = \langle Q, \Sigma, (\xrightarrow{a})_{a \in \Sigma} \rangle$$

A path in a transition system is a sequence $\pi = e_1, \ldots, e_n$ of edges such that $\mathsf{src}(e_{i+1}) = \mathsf{trg}(e_i)$. $n \geq 1$ is the length of the path, $\mathsf{src}(e_1)$ is its source and $\mathsf{trg}(e_n)$. We allow an empty path $p \in Q$ of length 0, with p being its source and target. We write $\mathsf{Path}(\mathcal{T})$ or $\mathsf{Path}_{\mathcal{T}}$ for the collection of all paths in $\mathcal{T}$. Likewise, $\mathsf{Path}(\mathcal{T}, p)$ or $\mathsf{Path}_{\mathcal{T}}(p)$ denotes the paths with source p.

Note that there is a natural partial concatenation operation on $\mathsf{Path}(\mathcal{T})$: paths π and π' can be joined to form a new path $\pi\pi'$ as long as $\mathsf{src}(\pi') = \mathsf{trg}(\pi)$. Thus we have a partial monoid.

The label or trace of a path π is the sequence of its edge lables, considered to be an element of the free monoid $\Sigma^\star$:

$$\mathsf{lab}(\pi) = \mathsf{lab}(e_1)\mathsf{lab}(e_2)\ldots\mathsf{lab}(e_n)$$

The label of the empty path is ε, the unit in $\Sigma^\star$.

Definition

A simulation from $\mathcal{T}_1$ by $\mathcal{T}_2$ is a binary relation $\rho \subseteq Q_1 \times Q_2$ such that for all $p\ \rho\ q$ the following holds:

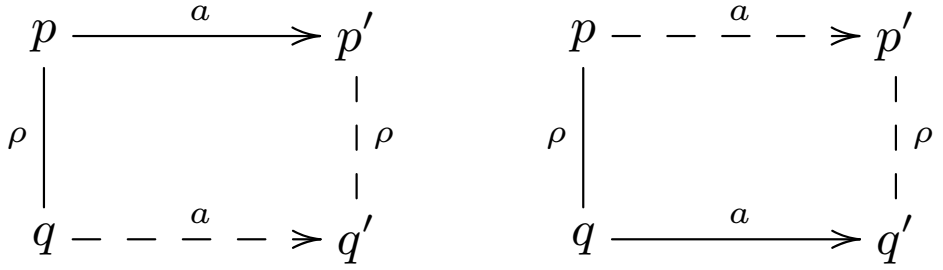
$$p \xrightarrow{a} p' \quad \text{implies} \quad \exists\, q \xrightarrow{a} q' (p'\ \rho\ q')$$

A bisimulation is a simulation ρ such that ρ^{-1} is also a simulation.

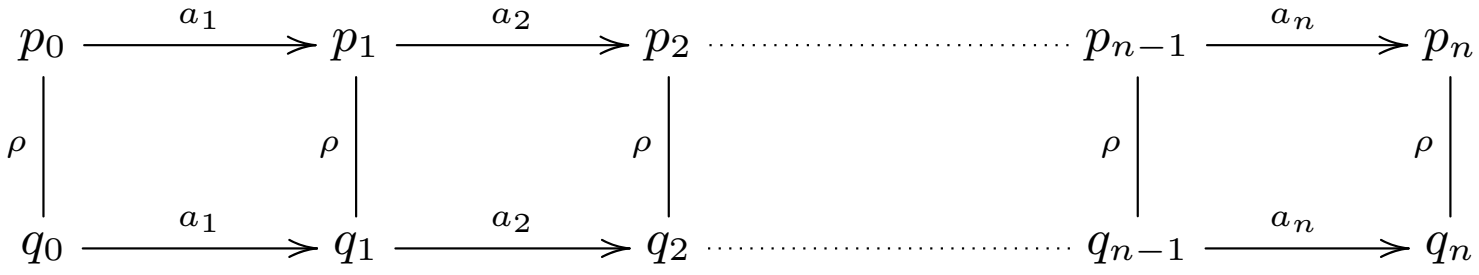
This is the most natural framework for simulations: one system, here $\mathcal{T}_2$, can mimick another, here $\mathcal{T}_1$. For more general systems such as Turing machines or cellular automata the details can be very complicated, but for transition system things are fairly straightforward.

Two systems with a bisimulation are in some sense equivalent.

Transitions starting at equivalent states can be extended on the other side.



This generalized immediately to whole computations.



This is slightly counterintuitive, but it turn out to be technically easier to deal with bisimulations on a *single* transition system. There is no loss in generality: we can simply consider the disjoint sum of the two systems, with state set $Q = Q_1 \uplus Q_2$.

In this setting, one can express the conditions for a bisimulation nicely in terms of relational composition:

$$\rho \circ \xrightarrow{a} \subseteq \xrightarrow{a} \circ \rho$$
$$\rho^{-1} \circ \xrightarrow{a} \subseteq \xrightarrow{a} \circ \rho^{-1}$$

Make sure to prove that this definition is equivalent to our original one.

Lemma

Bisimulations are equivalence relations.
Bisimulations are closed under union. Hence there is a uniquely determined maximum bisimulation.

Write $\theta_{\mathcal{T}}$ for this unique maximum bisimulation on $\mathcal{T}$. In terms of equivalence relations this is the coarsest partition that is a bisimulation.

This suggests one might try to use a partition refinement algorithm to compute $\theta_{\mathcal{T}}$. The starting partition depends on the specific context.

A standard problem in automata theory is to test whether two DFA are equivalent (accept the same language). One way to do this is to minimize both machines and check for isomorphism.

A better approach is to try to construct a bisimulation. Suppose the two given DFAs have disjoint state sets Q_1 and Q_2 and set $Q = Q_1 \cup Q_2$. For simplicity write δ for $\delta_1 \cup \delta_2$.

In the following description, we assume ρ to be an implementation for dynamic equivalence relations: union/find is the standard approach.

Algorithm

push (q_{01}, q_{02}) into S
initialize ρ to (q_{01}, q_{02})

while $S \neq \emptyset$ **do**
    pop (p, q) from S
    if $\neg p \; \rho \; q$ **then**
        add (p, q) to ρ
        forall $a \in \Sigma$ **do**
            add $(\delta(p, a), \delta(q, a)$ to S

return ρ saturates $F_1 \cup F_2$

In practice, one would exit with answer No as soon as a bad pair (p, q) is encountered.

Running time is near-linear and better than any minimization-based algorithm.

As the equivalence testing example indicates, in applications to finite state machines, one often has to deal with states of different kinds (final and non-final states in the example).

This can be modeled by a partition of the nodes, usually expressed in terms of node labels, a function

$$\lambda : Q \to C$$

where C is a finite set of "colors."

We are then interested in bisimulations that refine the give partition. In other words, the bisimulation has to saturate the given partition.

Given a node-labeled transition system $\mathcal{T} = \langle Q, \Sigma, E; \lambda \rangle$, we can reduce the computation of the maximum bisimulation to a system with a one-symbol alphabet; really, just an unlabeled, directed graph. To this end,

- Split each transition $p \xrightarrow{a} q$ in $\mathcal{T}$ into two directed edges

$$p \to (p, a, q) \qquad (p, a, q) \to q$$

- This produces a new (unlabeled) system with states $Q' = Q \cup E$.
- Extend the label function by setting $\lambda'(p) = \lambda(p)$, $\lambda'(p, a, q) = (\lambda(p), a)$.

Then construct the maximum bisimulation for $\mathcal{T}' = \langle Q', \Sigma, \to; \lambda' \rangle$.

Define an acyclic rooted digraph (ARD) to be an acyclic digraph G together with a special point r (the root), such that all of G is reachable from r.

Every well-founded pure set S can be represented by an ARD $G(S)$:

- If $S = \emptyset$, then $G(S)$ has a single node r.
- If $S = \{S_1, \ldots, S_n\}$, then $G(S)$ is obtained by choosing a new root node r, and attaching the roots of the $G(S_i)$ as children of r.

Similarly we can translate any ARD G into a set $S(G)$. Both conversions are single-valued, but note that there can be many different APGs that produce the same set.

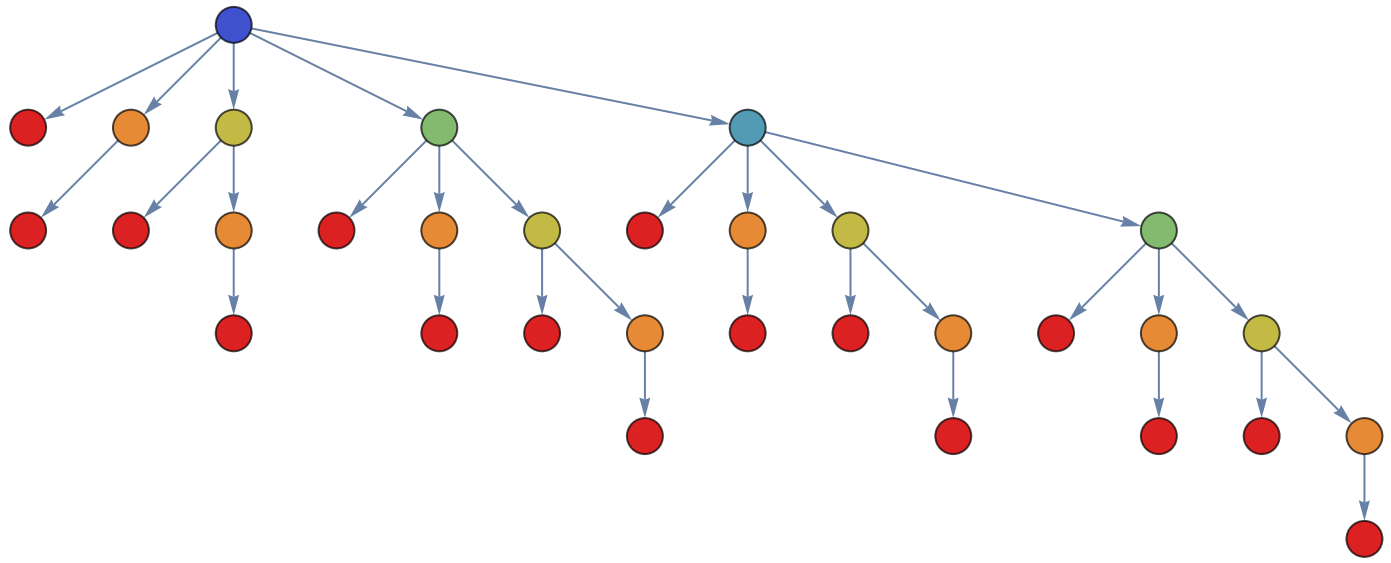
Sets representing finite ordinals (aka natural numbers) can be defined by

$$N_0 = \emptyset$$
$$N_n = \{N_0, N_1, \ldots, N_{n-1}\}$$

Here is the set representing the natural number 5.

$$\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\}\}$$

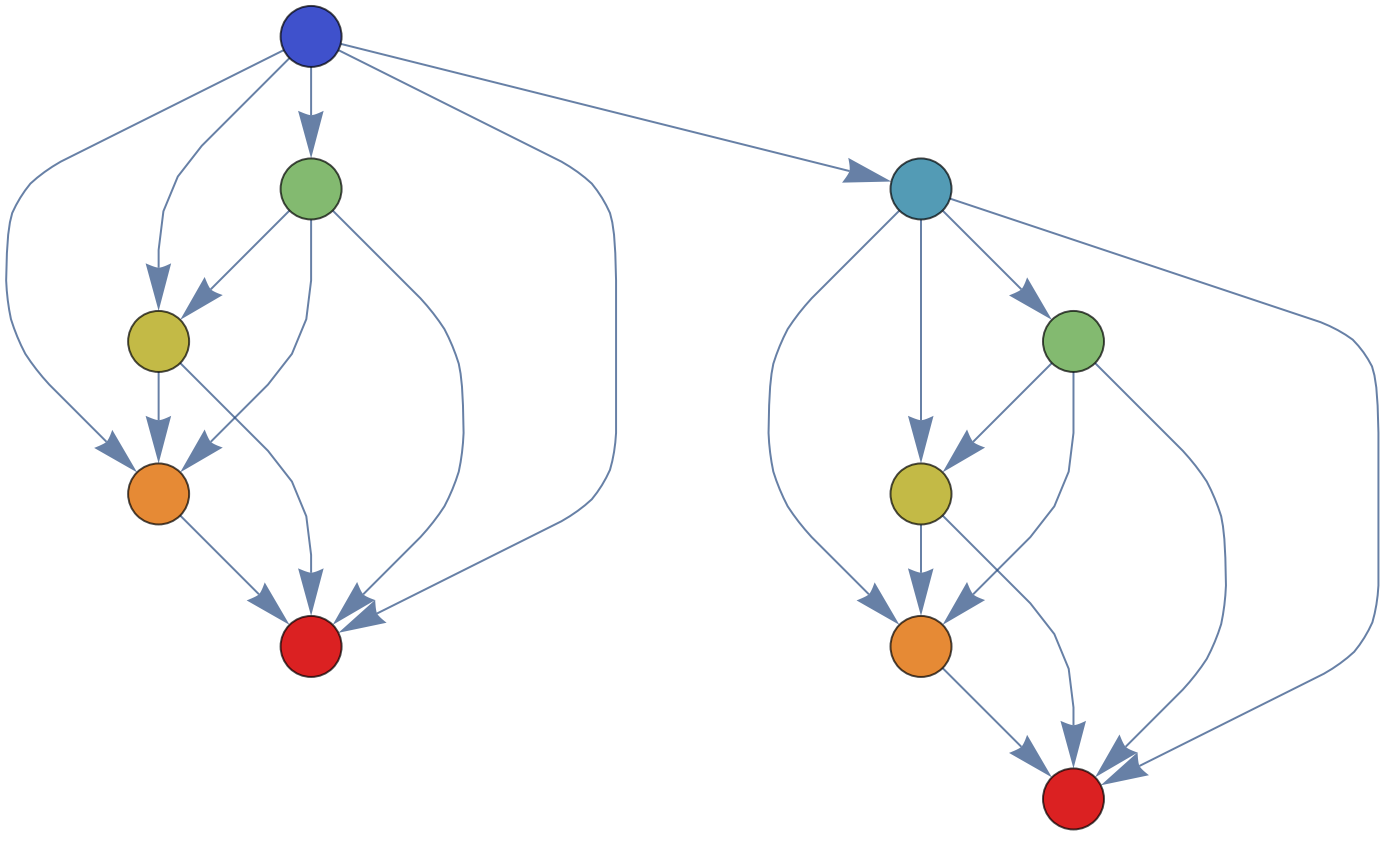
In textual form, a nightmare, but the structure is not too bad in tree representation:



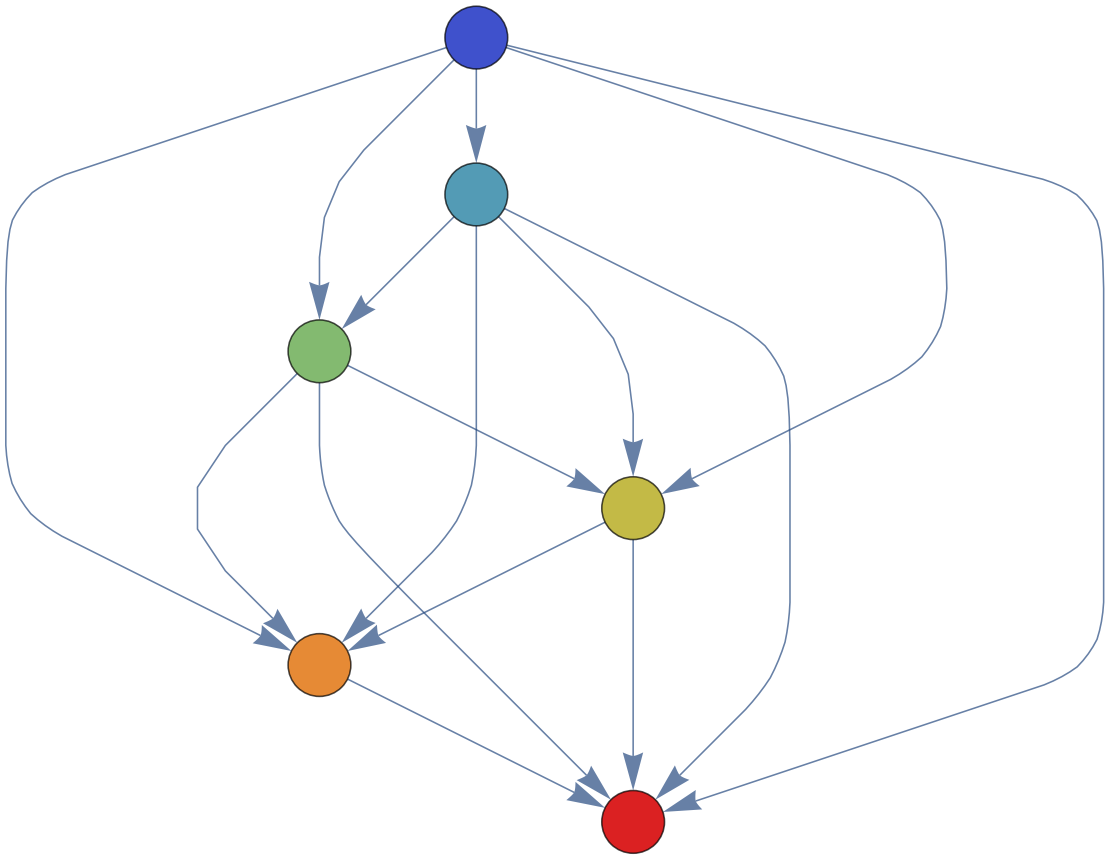
Colors correspond to rank, and it is natural to partition the nodes into leaves and interior nodes.

There are 2^{4-k} paths from the root to a rank k point.

Half Merged



Fully Merged



The number of paths from the root to a rank k point is still 2^{4-k}.

Suppose we have two blocks B and C in a partition Π.

If there are nodes b and b' in B such that $b \to c$ for some $c \in C$, but no such edge exists for b', we must split B by C:

$$B^+ = \{\, b \in B \mid \exists\, c \in C\, (b \to c) \,\}$$
$$B^- = \{\, b' \in B \mid \forall\, c \in C\, (b' \not\to c) \,\}$$

We keep on splitting until no further splits are possible.

The details are quite messy, but in the acyclic case it is fairly straightforward to organize this method efficiently.

ARD Algorithm

Input: ARD G, partition Π_C (the color partition).

Precomputation: compute the rank of each vertex.

refine Π_C by rank, order blocks by increasing rank
say $\Pi = B_1, \ldots, B_m$

for $i = 1, \ldots, m$ **do**
  for $j = i{+}1, \ldots, m$ **do**
    split B_j by B_i

return Π

Note that m will increase during the execution of this algorithm.

Claim: The splitting algorithm for ARDs is linear time.

Sketch of proof.

This requires a little fumbling.

The key observation that an edge can be used only once in the splitting process.

A clever data structure is needed to represent the partition (various arrays with pointers).

□