

Optimizing LTS-Simulation Algorithm

MEMICS'09

Lukáš Holík¹, Jiří Šimáček^{1,2}

email: {holik,isimacek}@fit.vutbr.cz, simacek@imag.fr

¹ FIT BUT, Božetěchova 2, 61266 Brno, Czech Republic

² VERIMAG, UJF, 2. av. de Vignate, 38610 Gières, France

Overview

- ❖ Motivation
- ❖ Basic definitions
- ❖ Existing simulation algorithms
- ❖ Optimizations
- ❖ Results (LTS)
- ❖ Tree automata simulations
- ❖ Results (TA)
- ❖ Conclusion

Motivation

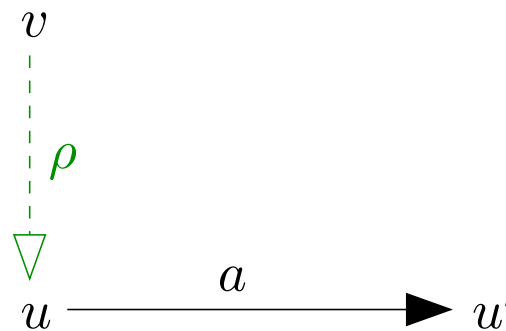
- ❖ A labeled transition systems (LTS) is one of the basic formalisms for
 - description of various systems in practice (LTL, CTL model checking)
 - intermediate representation of systems (regular model checking)
- ❖ LTSs can be very large
 - a possible solution is to collapse states according to a suitable equivalence relation
- ❖ A good candidate is simulation equivalence
 - strongly preserves logics like $ACTL^*$, $ECTL^*$, and LTL
 - a good compromise between the bisimulation equivalence and the language equivalence with respect to its reduction power and computation cost
 - useful for minimization of similar formalisms such as word/tree/Büchi automata
 - can be used as hint for language inclusion checking of given automata (if a state q of the automaton simulates a state r , then $L(r) \subseteq L(q)$)

LTS and Simulation Relation

❖ A **labeled transition system (LTS)** is a tuple $T = (S, \Sigma, \{\delta_a | a \in \Sigma\})$, where S is a finite set of states, Σ is a finite set of labels, and for each $a \in \Sigma$, $\delta_a \subseteq S \times S$ is an a -labeled transition relation.

❖ A **simulation** over T is a binary relation ρ on S such that:

$$(u, v) \in \rho \wedge u' \in \delta_a(u) \Rightarrow \exists v' \in \delta_a(v). (u', v') \in \rho$$

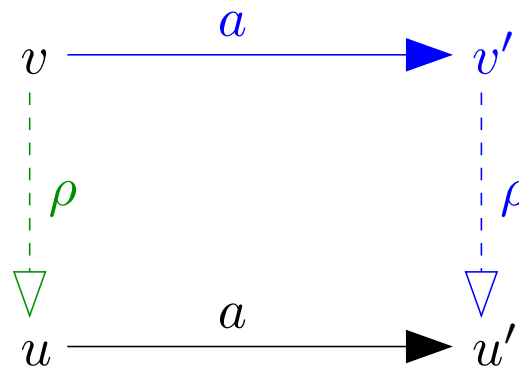


LTS and Simulation Relation

❖ A **labeled transition system (LTS)** is a tuple $T = (S, \Sigma, \{\delta_a | a \in \Sigma\})$, where S is a finite set of states, Σ is a finite set of labels, and for each $a \in \Sigma$, $\delta_a \subseteq S \times S$ is an a -labeled transition relation.

❖ A **simulation** over T is a binary relation ρ on S such that:

$$(u, v) \in \rho \wedge u' \in \delta_a(u) \Rightarrow \exists v' \in \delta_a(v). (u', v') \in \rho$$



Existing Simulation Algorithms

❖ Existing algorithms mostly maintains an **overapproximation** Sim of the simulation:

- initialize Sim as the **full relation** (every state simulates every other state)
- **gradually remove pairs of states** which violate the simulation property
- after reaching the **fixpoint**, Sim is the simulation

❖ A solution using **remove sets** requires:

$\mathcal{O}(|S||\delta|)$ time and $\mathcal{O}(|S|^2)$ space for KS

[Henzinger et al. FOCSS'95]

$\mathcal{O}(|S||\delta| + |\Sigma||S|^2)$ time and $\mathcal{O}(|\Sigma||S|^2)$ space for LTS

❖ A solution using remove sets and **state partitioning** requires:

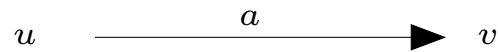
$\mathcal{O}(|P_{Sim}||\delta|)$ time and $\mathcal{O}(|P_{Sim}||S|)$ space for KS

[Ranzato, Tapparo LICS'07]

$\mathcal{O}(|P_{Sim}||\delta| + |\Sigma||P_{Sim}||S|)$ time and $\mathcal{O}(|\Sigma||P_{Sim}||S|)$ space for LTS

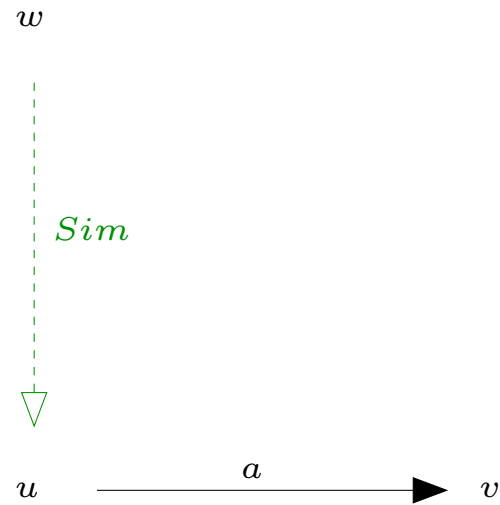
[Abdulla et al. TACAS'08]

Labeled HHK



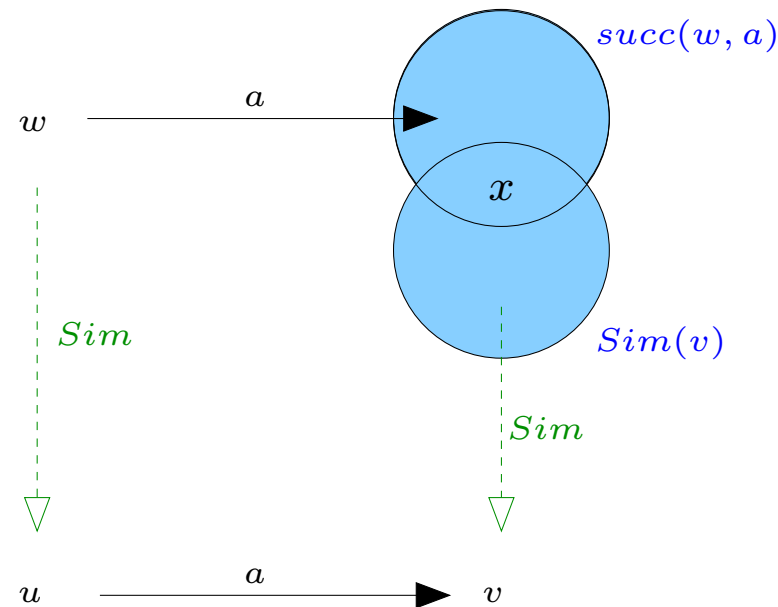
❖ Suppose that u can go to v via a .

Labeled HHK



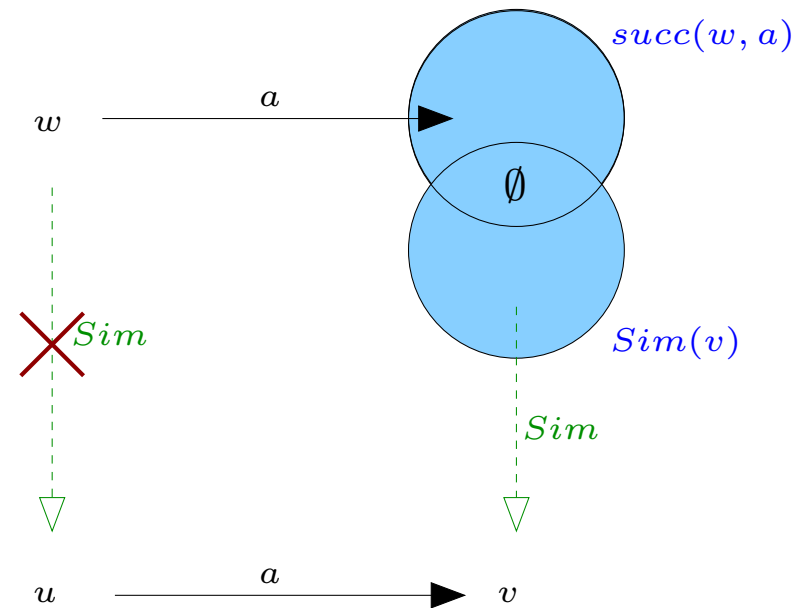
- ❖ Suppose that u can go to v via a , and there exists some w which simulates u .

Labeled HHK



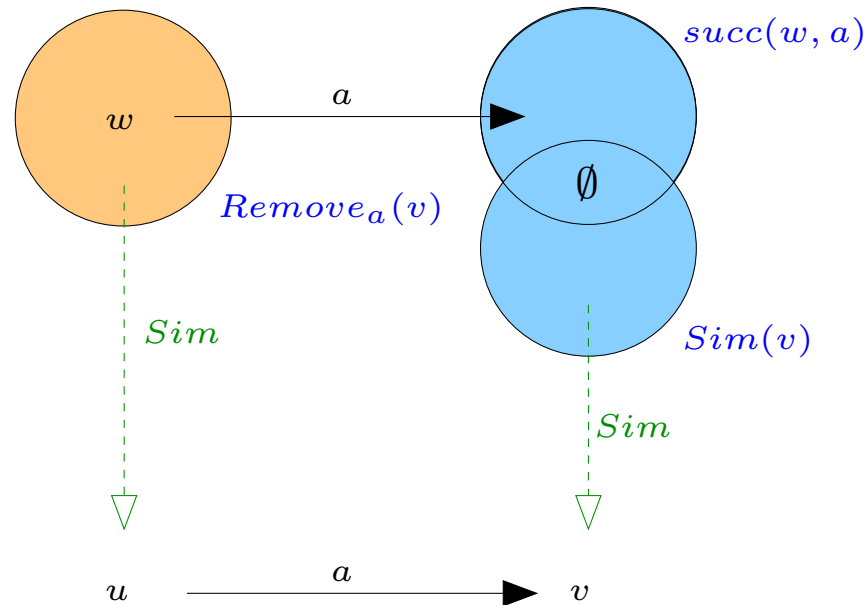
- ❖ Suppose that u can go to v via a , and there exists some w which **simulates** u .
- ❖ Because w simulates u and u has an a -labeled transition to v , it must be the case that there exists **some state** x in the set of a -successors of w ($\text{succ}(w, a)$), which simulates v (from the definition).

Labeled HHK



❖ If there is **no such a state** $(\text{succ}(w, a) \cap \text{Sim}(v) = \emptyset)$, then w cannot simulate u and the relation Sim needs to be updated.

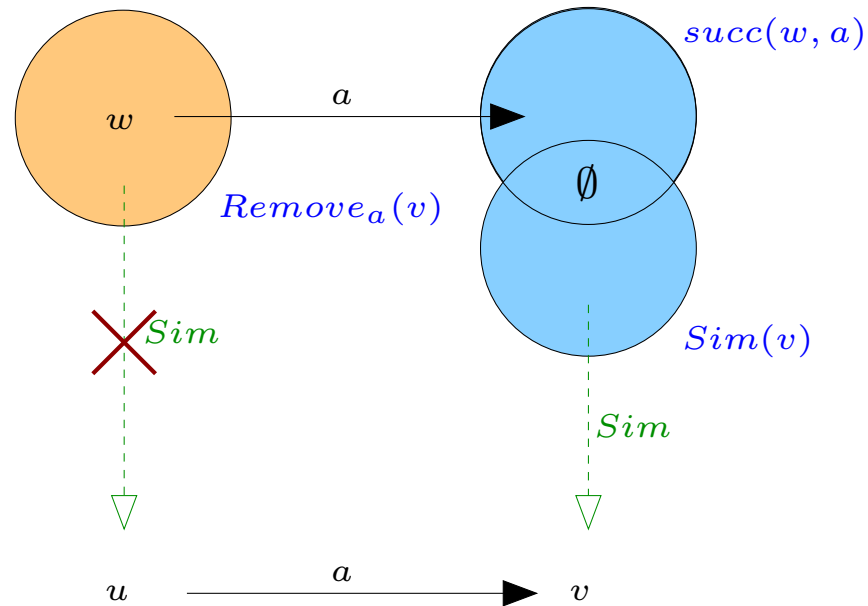
Labeled HHK



❖ If there is **no such a state** ($\text{succ}(w, a) \cap \text{Sim}(v) = \emptyset$), then w cannot simulate u and the relation Sim needs to be updated.

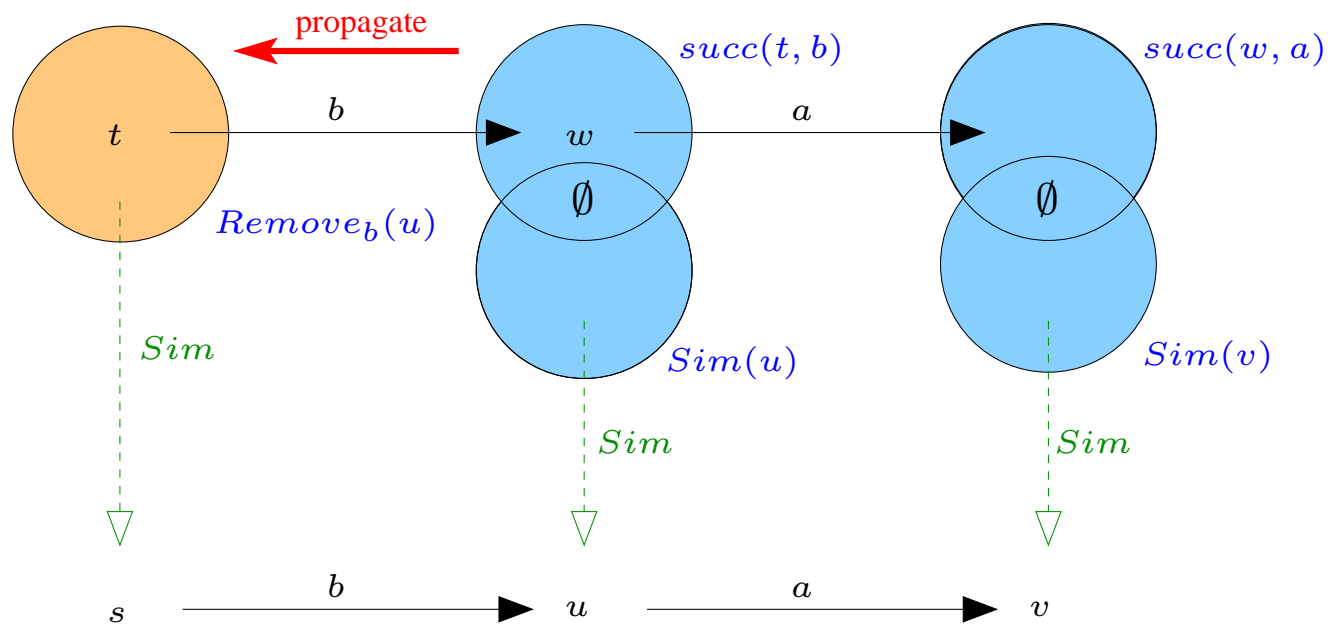
❖ However, this update is not performed directly, but using so called **remove sets**. Here $\text{Remove}_a(v)$ is a set, which contains states that do not have **a -labeled transitions** going into some state which **simulates** v .

Labeled HHK



- ❖ After computing the set $Remove_a(v)$, we update the relation Sim (by iterating over all a -labeled predecessors of v).

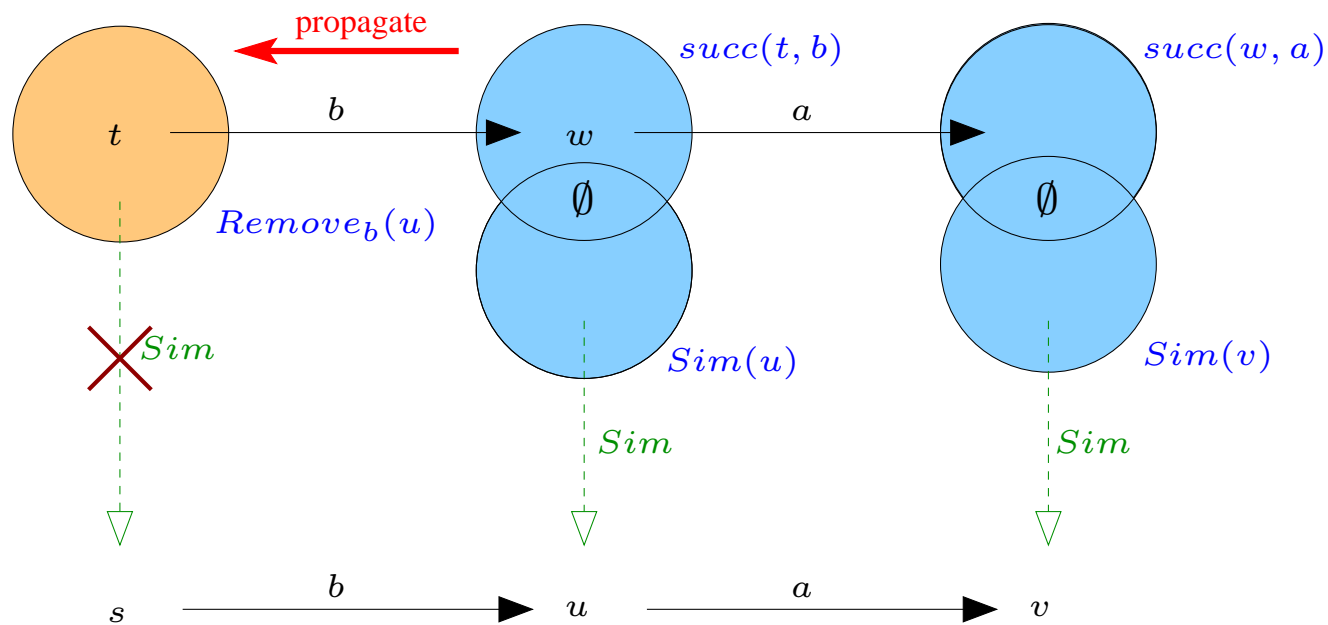
Labeled HHK



❖ After computing the set $Remove_a(v)$, we update the relation Sim (by iterating over all a -labeled predecessors of v).

❖ However, this update could cause that some states **no longer satisfy** the simulation property. This change is propagated by generating new remove sets.

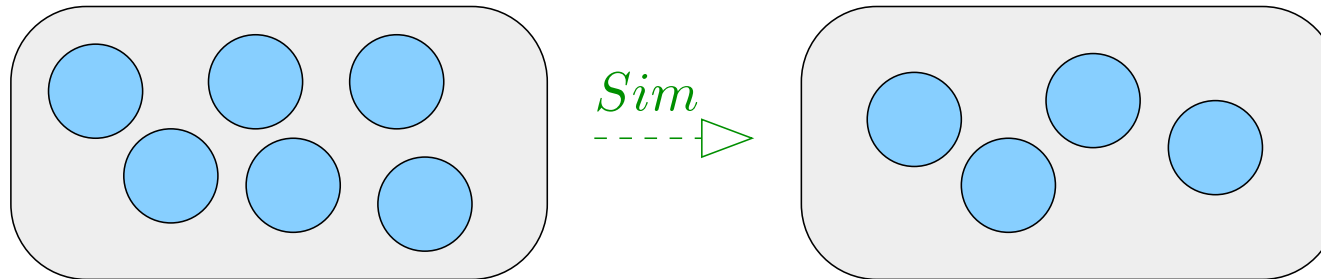
Labeled HHK



❖ After processing all remove sets that are generated during the computation, the algorithm terminates.

❖ At the end Sim contains the largest relation, which satisfies the required property (i.e. is the largest simulation over a given LTS).

From labeled HHK to labeled RT (IRT)

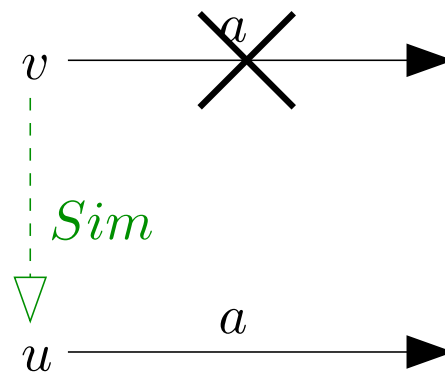


- ❖ Instead of computing relation over individual states, we compute **relation over partitions (called blocks)** of the given state space ([Ranzato, Tapparo LICS'07]).
- ❖ Each block contains one equivalence class of *Sim* in the current iteration.
- ❖ As algorithm progresses, these **blocks are refined (split)** on the fly whenever the change to *Sim* requires it. Every time some block is split into two, the **relation is extended** as well.
- ❖ **Remove sets are computed for blocks**, not for individual states.

From IRT to optimized IRT

- ❖ Although the proposed optimizations are somewhat **simple**, they work **amazingly well** in practice.
- ❖ We can **prune the relation Sim during the initialization** phase instead of assuming that each state can simulate any other state at the beginning.
- ❖ More specifically, let $out(v)$ be a set of all symbols a such that v has some a -labeled outgoing transition ($out(v) = \{a \in \Sigma \mid \delta_a(v) \neq \emptyset\}$). Then, from the definition of Sim , we have:

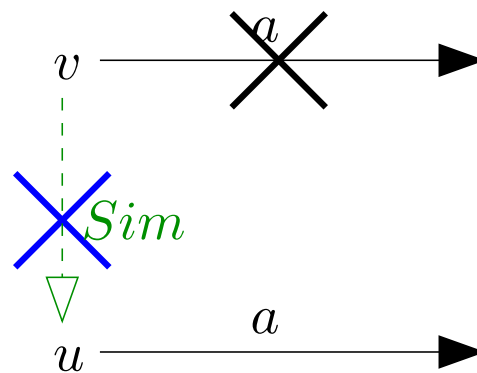
$$out(v) \not\subseteq out(u) \Rightarrow (u, v) \notin Sim$$



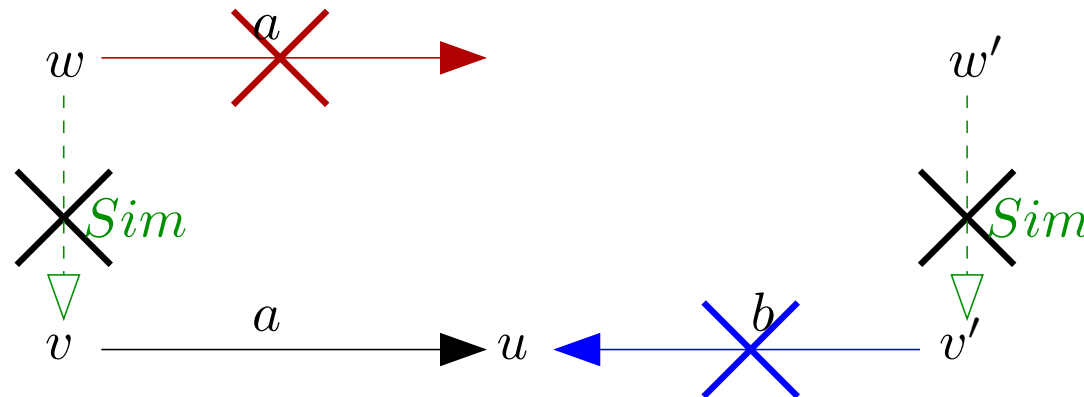
From IRT to optimized IRT

- ❖ Although the proposed optimizations are somewhat **simple**, they work **amazingly well** in practice.
- ❖ We can **prune the relation Sim during the initialization** phase instead of assuming that each state can simulate any other state at the beginning.
- ❖ More specifically, let $out(v)$ be a set of all symbols a such that v has some a -labeled outgoing transition ($out(v) = \{a \in \Sigma \mid \delta_a(v) \neq \emptyset\}$). Then, from the definition of Sim , we have:

$$out(v) \not\subseteq out(u) \Rightarrow (u, v) \notin Sim$$



From IRT to optimized IRT II



- ❖ As a result, we have to **compute the set $Remove_b(u)$ only if there exists some b -labeled transition going into u** . If not, then we know that all possible pairs, which could be removed by processing $Remove_b(u)$, were already removed during the initialization.
- ❖ Moreover, we **only add states to $Remove_a(u)$ if they have some a -labeled transition** for the same reason.
- ❖ The advantage of this approach is that **partitioning the state space during the initialization phase is much faster** (cheaper) than during the computation phase (no remove sets are generated, etc.)

Complexity of optimized IRT

❖ Time complexity of original IRT is $\mathcal{O}(|\Sigma||P_{Sim}||S| + |P_{Sim}||\delta|)$. Using proposed optimizations we get

$$\mathcal{O}\left(|\Sigma||P_{Out}|^2 + |\Sigma||S| + |P_{Sim}|^2 + \sum_{B \in P_{Sim}} \left(\sum_{a \in \text{in}(B)} (|\delta_a^{-1}(S)| + |\delta_a|) \right) + \sum_{v \in \text{Out}(B)} |\delta^{-1}(v)|\right)$$

❖ In the case of space complexity, original IRT requires $\mathcal{O}(|\Sigma||P_{Sim}||S|)$, whereas optimized version needs

$$\mathcal{O}(|P_{Sim}|^2 + |\Sigma||S| + \sum_{B \in P_{Sim}} \sum_{a \in \text{in}(B)} |\delta_a^{-1}(S)|)$$

❖ The most significant gains occur, when the initial partitioning contains many equivalence classes (i.e. states have heterogenous sets of outgoing labels).

Results

❖ We implemented both **IRT** and **optimized IRT** in **OCaml** and we performed the experiments using **AMD Opteron 8389 2.90 GHz PC with 128 GiB** of memory (the memory limit for a process was set to **approximately 20 GiB**).

❖ We used several randomly generated LTSs and some LTSs obtained from the run of ARMC verifier.

source	LTS			LRT		OLRT		Gain
	$ S $	$ \Sigma $	$ \delta $	time	space	time	space	
random	256	16	416	0.12	9.6	0.02	1.9	6x
random	4096	16	3280	13.82	714.2	2.02	78.2	6x
random	16384	16	26208	o.o.m.		268.85	4514.9	-
random	4096	32	6560	62.09	1844.2	4.36	121.4	15x
random	4096	64	13120	158.38	3763.2	6.59	211.2	26x
pc	1251	43	49076	7.52	418.1	2.63	119.0	3x
rw	4694	11	20452	81.28	3471.8	19.25	989.3	4x
lr	6160	35	90808	390.91	12640.8	45.69	1533.6	8x

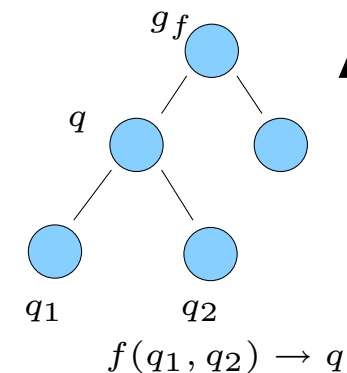
Tree Automata

❖ A (finite, bottom-up) **tree automaton (TA)** is a quadruple $A = (Q, \Sigma, \Delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is a set of final states, Σ is a ranked alphabet with a ranking function $r : \Sigma \rightarrow \mathbb{N}$, and $\Delta \subseteq Q^* \times \Sigma \times Q$ is a set of transition rules of the form

$$f(q_1 \dots q_n) \rightarrow q \in \Delta,$$

where $r(f) = n$.

❖ The **finite tree automaton** is a **generalization** of the **finite word automaton** that performs its computation **over trees** instead of strings.



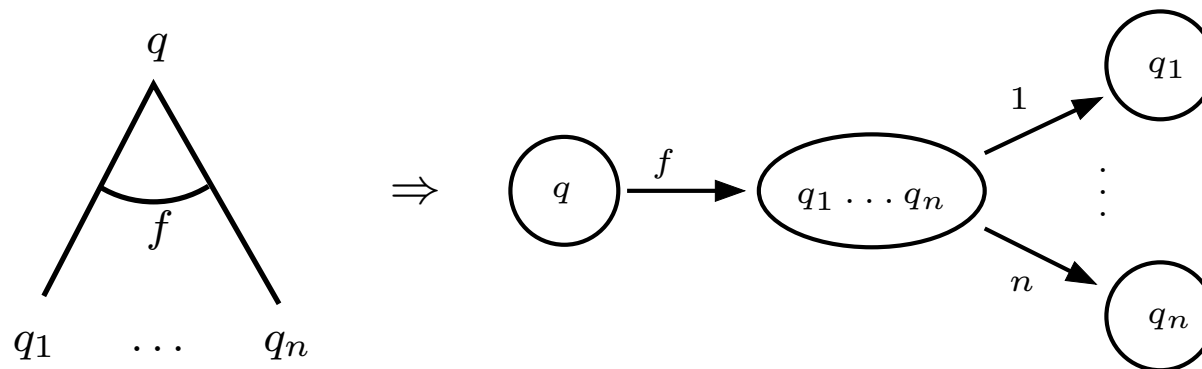
❖ A **downward simulation** over **tree automata** is similar to the simulation over **word automata**. Additionally, there also exists **upward simulation** (which is beyond the scope of this presentation).

Tree Automata Simulations

❖ A **downward simulation** D is a binary relation on Q such that if $(q, r) \in D$, then for all $f(q_1 \dots q_n) \rightarrow q \in \Delta$, there exists $f(r_1 \dots r_n) \rightarrow r \in \Delta$ such that $(q_i, r_i) \in D$ for each $i : 1 \leq i \leq n$.

❖ As shown in [Abdulla et al. TACAS'08], the problem of computing simulations over tree automata can be translated into the problem of computing simulations over LTS.

❖ The idea of TA-to-LTS translation (downward simulation):



❖ Our optimized algorithm runs **asymptotically better** on the LTSs generated this way.

Results II

❖ In this case, we used several randomly generated TAs and some TAs obtained from the run of ARTMC verifier.

source	TA				LTS			LRT		OLRT		Gain
	$ Q $	$ \Sigma $	r_m	$ \Delta $	$ S $	$ \Sigma $	$ \delta $	time	space	time	space	
random	16	16	2	245	184	18	570	0.06	6.2	0.02	1.4	3x
random	32	16	2	935	655	18	2165	0.87	74.4	0.21	14.4	4x
random	64	16	2	3725	2502	18	8568	26.63	1417.9	3.50	195.4	8x
random	32	32	2	1164	719	34	2511	2.67	166.6	0.23	16.8	11x
random	32	64	2	2026	925	66	3780	12.17	623.5	0.56	25.4	21x
ARTMC	47	132	2	837	241	134	1223	0.84	70.6	0.05	6.2	16x
ARTMC	variable							517.98	116.2	80.84	22.1	6x

Conclusion

- ❖ We devised an improved algorithm for computing simulations over LTS, which
 - is suitable for systems with large alphabets
 - is asymptotically better than IRT when used for computing simulations over tree automata
 - performs very well in practice
 - it is faster than IRT by one order of magnitude (on average)
 - it consumes one order of magnitude less memory
 - in some cases we can get as much as two orders of magnitude in both space and time