Automata-based Techniques for the Verification of Programs with Dynamic Linked Data Structures

Ahmed Bouajjani
LIAFA - University of Paris 7, France

joint work with

Peter Habermehl **Pierre Moro**
LIAFA - University of Paris 7, France

Adam Rogalewicz **Tomas Vojnar**
FIT - Brno University of Technology, Czech Rep.

Programs and Properties

Programs

- Finite-state data domains,
- Sequential programs without procedure calls (finite control),
- Dynamic creation of objects, destructive updates and pointer manipulation

Properties

- Absence of intrinsic errors (e.g., null pointer dereference, use of undefined pointers, reference to deleted elements, etc.)
- Shape properties (e.g., at some given control point of the program, the heap is always a doubly linked list)

Our approach

- Reduce the verification of shape properties to control point reachability
  - Augment the program by testers of shape properties
- Automata-based framework:
  - Encode configurations as words/trees
  - Use finite-state word/tree automata to represent sets of configurations
  - Encode program statements as word/tree transducers
- Apply *abstract regular model checking* techniques to solve the problem

  Symbolic reachability analysis + (refinable) abstractions on automata

Regular Model Checking: A Generic Automata-based Framework

[Pnueli & al., 97], [Wolper, Boigelot, 98], [B., Nilsson, Jonsson, Touili, 00], [B., 01]

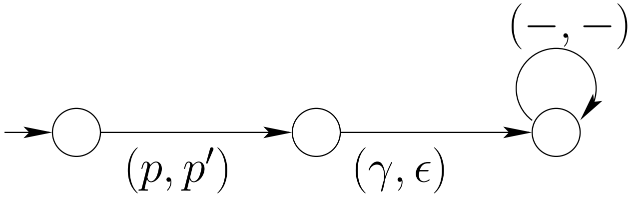
Finite State Automata $=$ BDD for Infinite State Systems

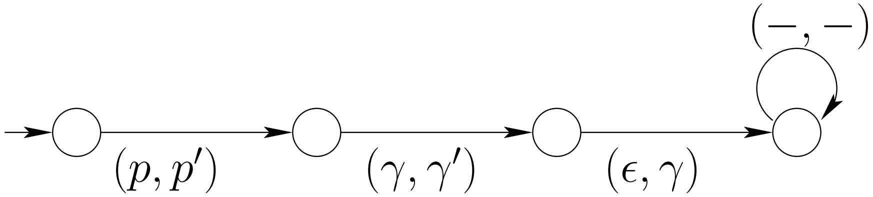
Regular Model Checking: A Generic Automata-based Framework

[Pnueli & al., 97], [Wolper, Boigelot, 98], [B., Nilsson, Jonsson, Touili, 00], [B., 01]

Finite State Automata = BDD for Infinite State Systems

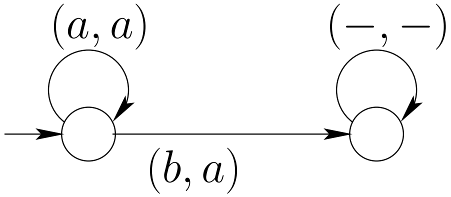
Pushdown systems (pop/push rules)

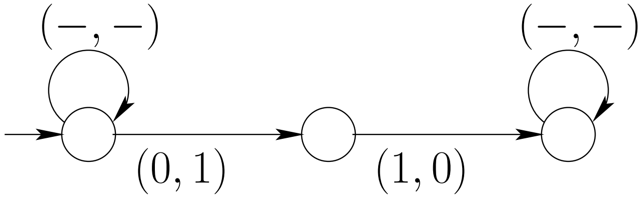




$$p\gamma \mapsto p' \qquad\qquad p\gamma \mapsto p'\gamma'\gamma$$

Parametrized networks





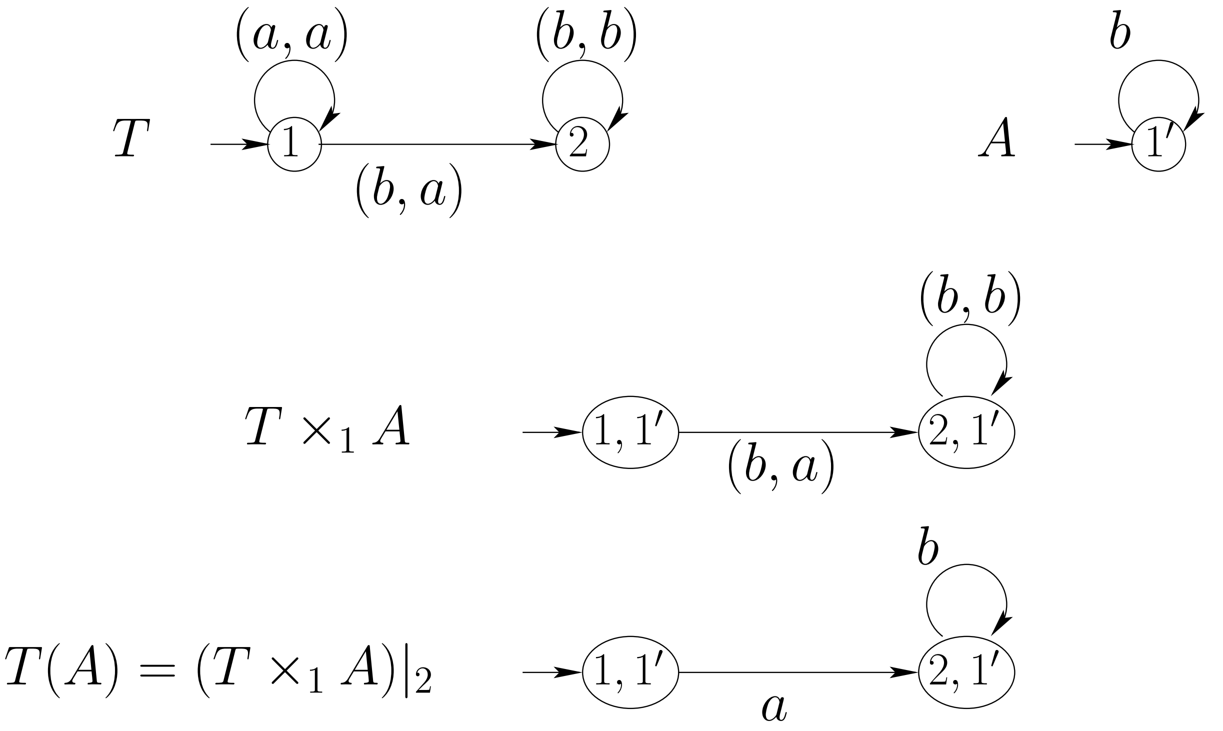
$$(\forall j < i.\ S[j] = a) \wedge S[i] = b \,/\, S[i] := a \qquad\qquad OutToken_i \,||\, InToken_{i+1}$$

Regular Model Checking: A Generic Automata-based Framework

[Pnueli & al., 97], [Wolper, Boigelot, 98], [B., Nilsson, Jonsson, Touili, 00], [B., 01]

Finite State Automata = BDD for Infinite State Systems

Computing post/pre images: Automata composition



Regular Model Checking: A Generic Automata-based Framework

[Pnueli & al., 97], [Wolper, Boigelot, 98], [B., Nilsson, Jonsson, Touili, 00], [B., 01]

Finite State Automata = BDD for Infinite State Systems

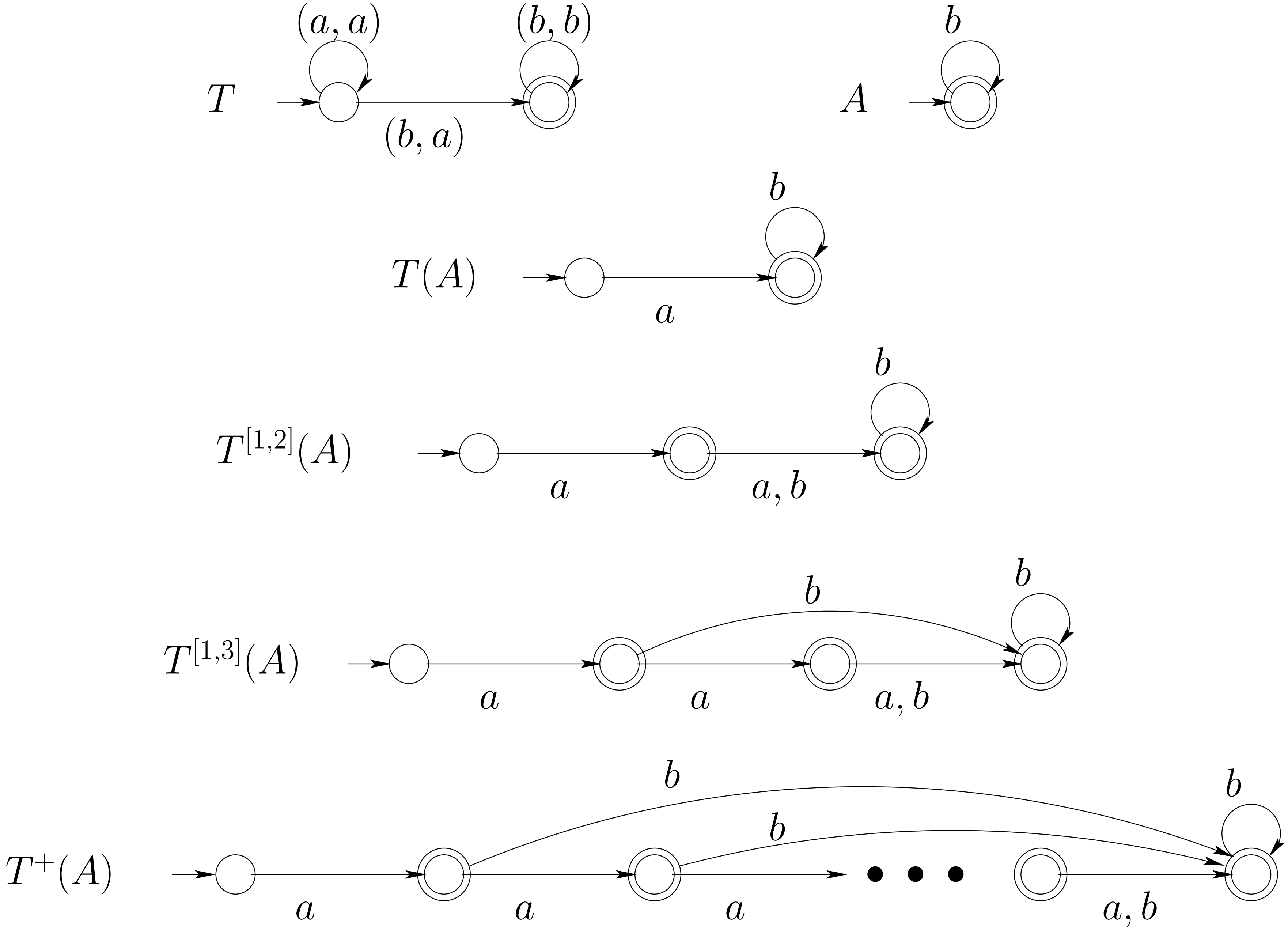
Reachability analysis: Computing transitive closures

$$\text{Safety} \quad \leadsto \quad T^*(Init) \cap Bad = \emptyset$$

Problems to face:

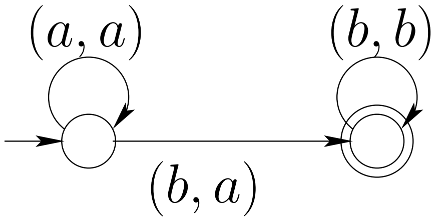
- Non regularity / Non constructibility of T^* and T^*-images
- Termination of the constructions
- State explosion of the automata / transducers

Termination Problem

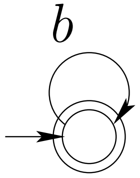


Termination problem: Acceleration ?

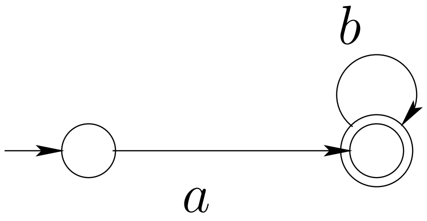
T



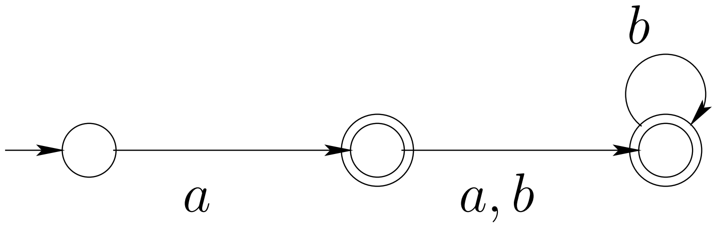
A



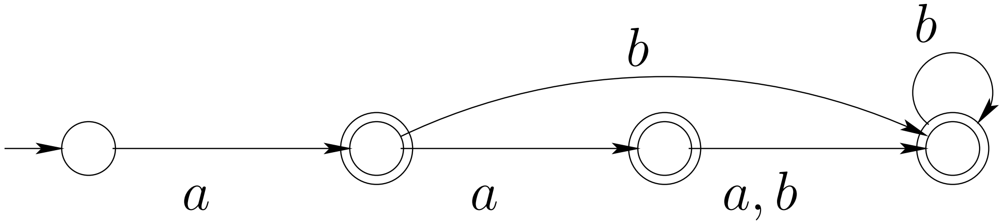
$T(A)$



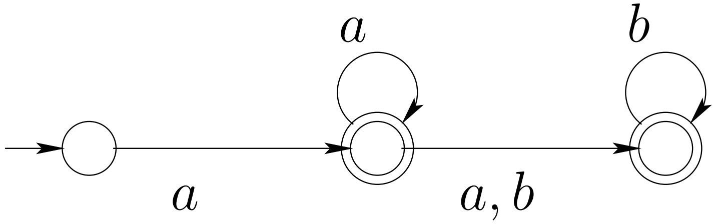
$T^{[1,2]}(A)$



$T^{[1,3]}(A)$



$T^{+}(A)$



Abstract Regular Model Checking

Given a transducer τ, and 2 automata I (initial) and B (bad), check:

$$\tau^*(I) \cap B = \emptyset$$

1. Define a *finite-range abstraction function* α on automata
2. Compute iteratively $(\alpha \circ \tau)^*(I)$,
3. If $(\alpha \circ \tau)^*(I) \cap B = \emptyset$ then answer YES

Abstract Regular Model Checking

Given a transducer τ, and 2 automata I (initial) and B (bad), check:

$$\tau^*(I) \cap B = \emptyset$$

$\Longrightarrow$ *Abstract-Check-Refine loop*

1. Define a *finite-range abstraction function* α on automata
2. Compute iteratively $(\alpha \circ \tau)^*(I)$,
3. If $(\alpha \circ \tau)^*(I) \cap B = \emptyset$ then answer YES

4. Otherwise, let θ be the computed symbolic path from I to B,
5. Check if θ includes a *concrete counterexample*,
   - If yes, then answer NO,
   - Otherwise, define a *refinement* of α which *excludes* θ, and goto (2).

Abstractions based on state space collapsing

- We define *parametrized* families of abstractions $\{\alpha_p\}_{\mathbb{P}}$:
- We consider equivalence relations $\simeq_p$ on states of automata, and we define

$$\alpha_p(A) = A/\simeq_p$$

- We require that for every $p \in \mathbb{P}$, the equivalence $\simeq_p$ is *finite-index*

⇒ For every p, α_p has a finite image (defines a finite abstract domain)

⇒ Abstract fixpoint computations always terminate

Instance 1: Abstraction based on *bounded length languages*

- Let $k \in \mathbb{N}$. Then, given an automaton A, consider the equivalence

$$q_1 \simeq_k q_2 \text{ iff } L(A, q_1)^{\leq k} = L(A, q_2)^{\leq k}$$

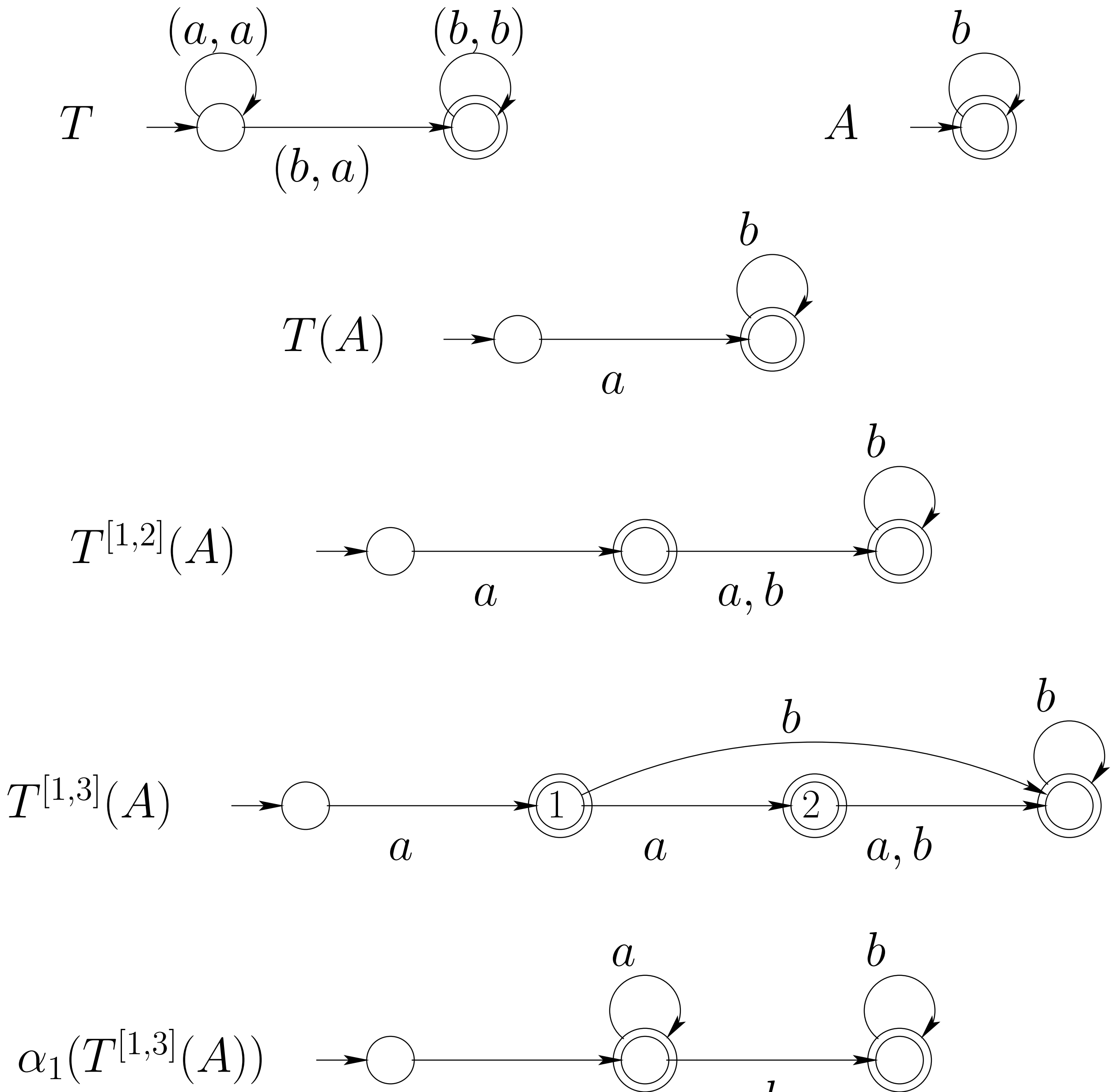
where

$$L(A, q)^{\leq k} = L(A, q) \cap \{w \in \Sigma^* \ : \ |w| \leq k\}$$

- For every $k \in \mathbb{N}$, let $\alpha_k(A) = A/ \simeq_k$

- The image of α_k is finite = all *minimal automata* for the finite languages of words of lengths $\leq k$.

Instance 1: Abstraction based on *bounded length languages*



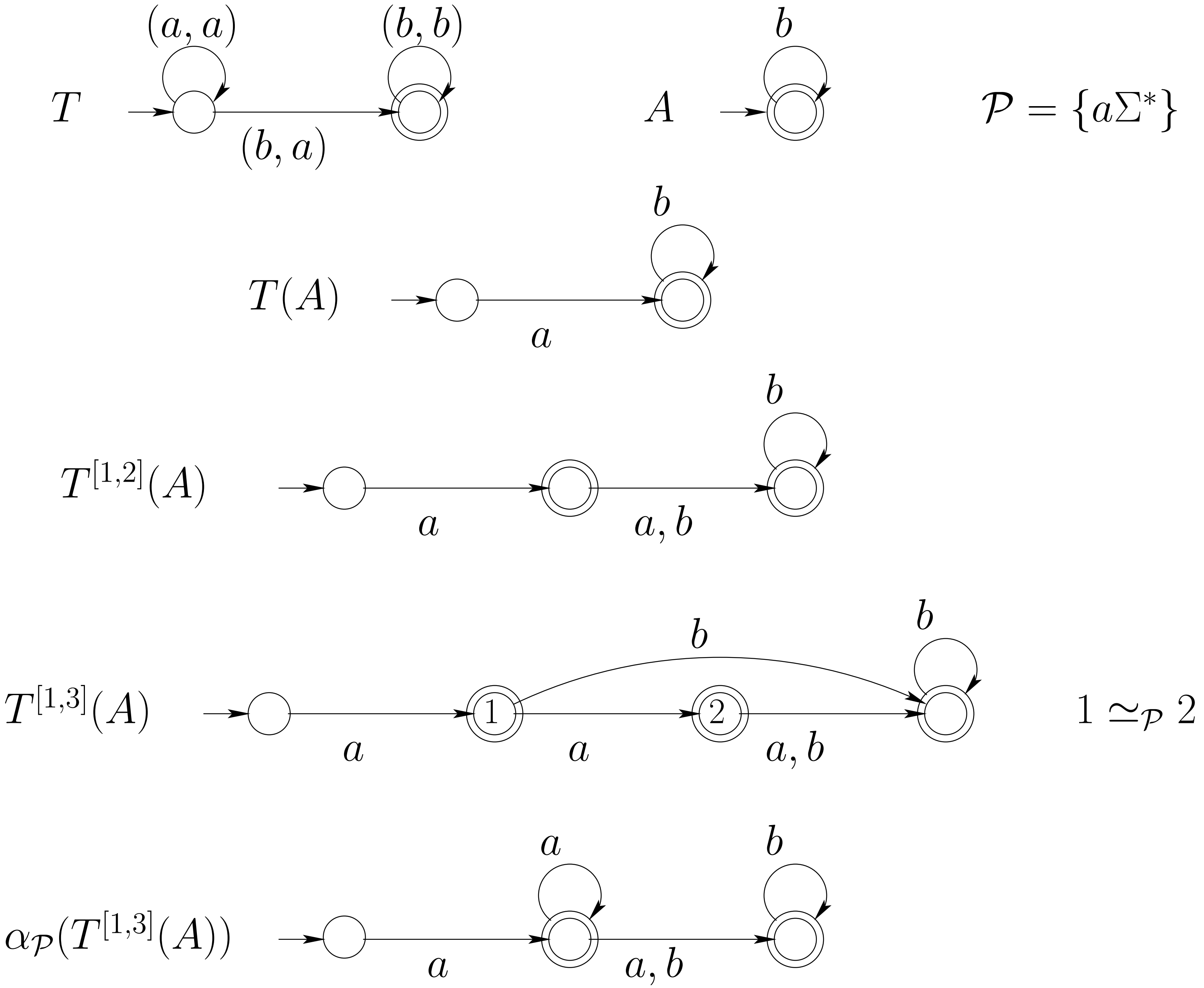
$L_{\leq 1}(1) = L_{\leq 1}(2)$

Instance 2: Predicate Automata Abstraction

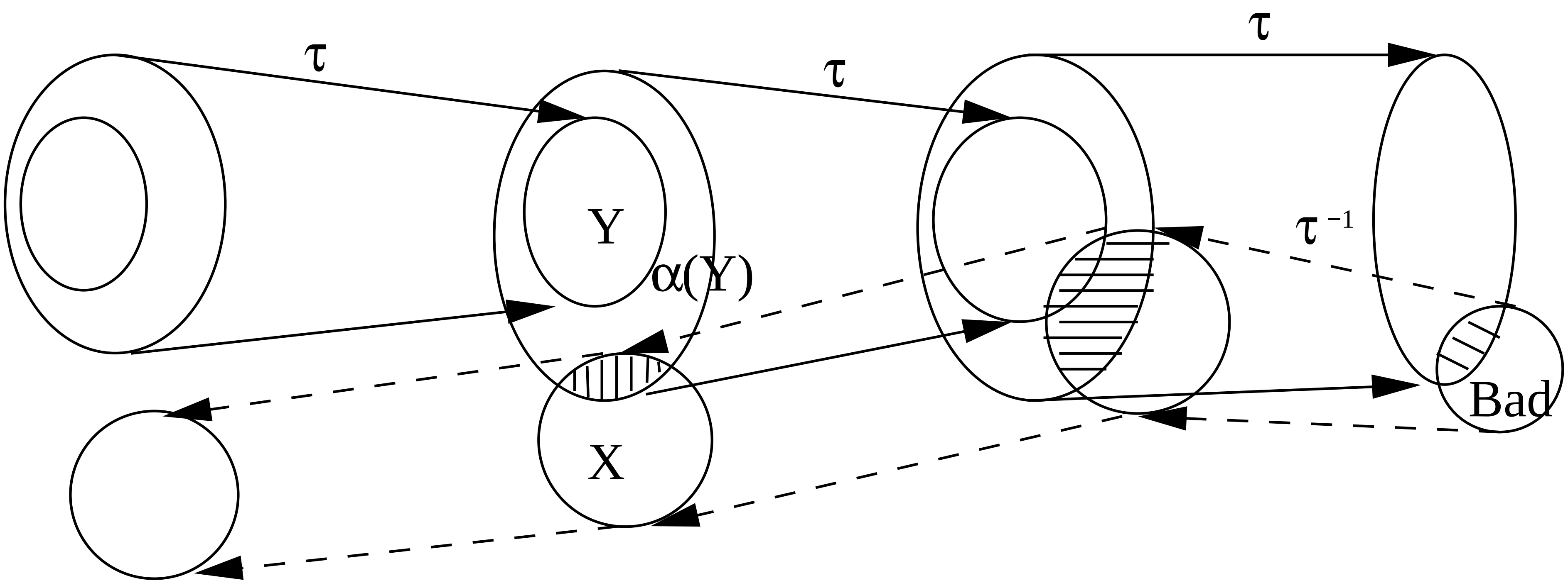
- Predicate = finite-state automaton (regular language)
- Let $\mathcal{P} = \{A_1, \ldots, A_n\}$ be a set of *predicate automata*.
- Let $A = (\Sigma, Q, q_0, F, \delta)$ be finite-state automaton. For every $q \in Q$, and for every $A_i \in \mathcal{P}$, let
$$q \models A_i \text{ iff } L(A, q) \cap L(A_i) \neq \emptyset$$
- Let $\simeq_{\mathcal{P}} \subseteq Q \times Q$ be the equivalence relation such that
$$\forall q, q' \in Q.\ q \simeq_{\mathcal{P}} q' \text{ iff } \forall A_i \in \mathcal{P}.\ q \models A_i \Leftrightarrow q' \models A_i$$
- We define
$$\alpha_{\mathcal{P}}(A) = A / \simeq_{\mathcal{P}}$$

$\Rightarrow$ The image of $\alpha_{\mathcal{P}}$ is finite = all automata with at most $2^{|\mathcal{P}|}$ states.

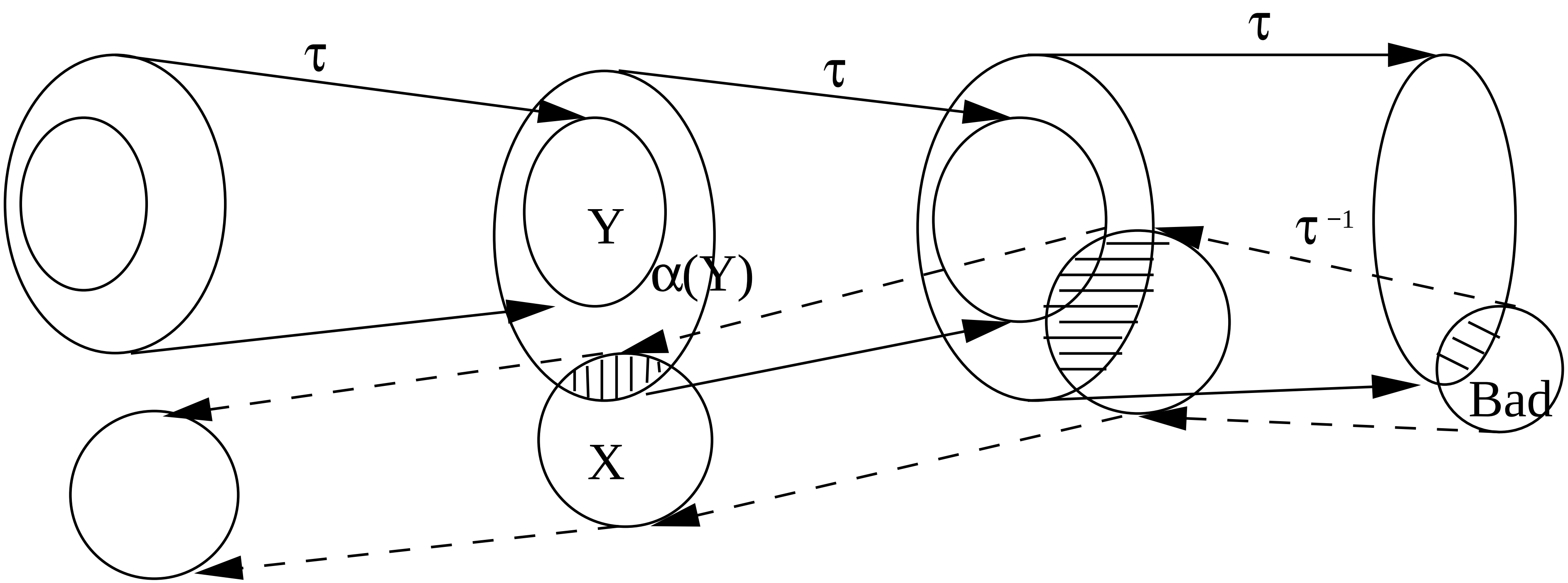
Instance 2: Predicate Automata Abstraction



Counter-example guided abstraction refinement



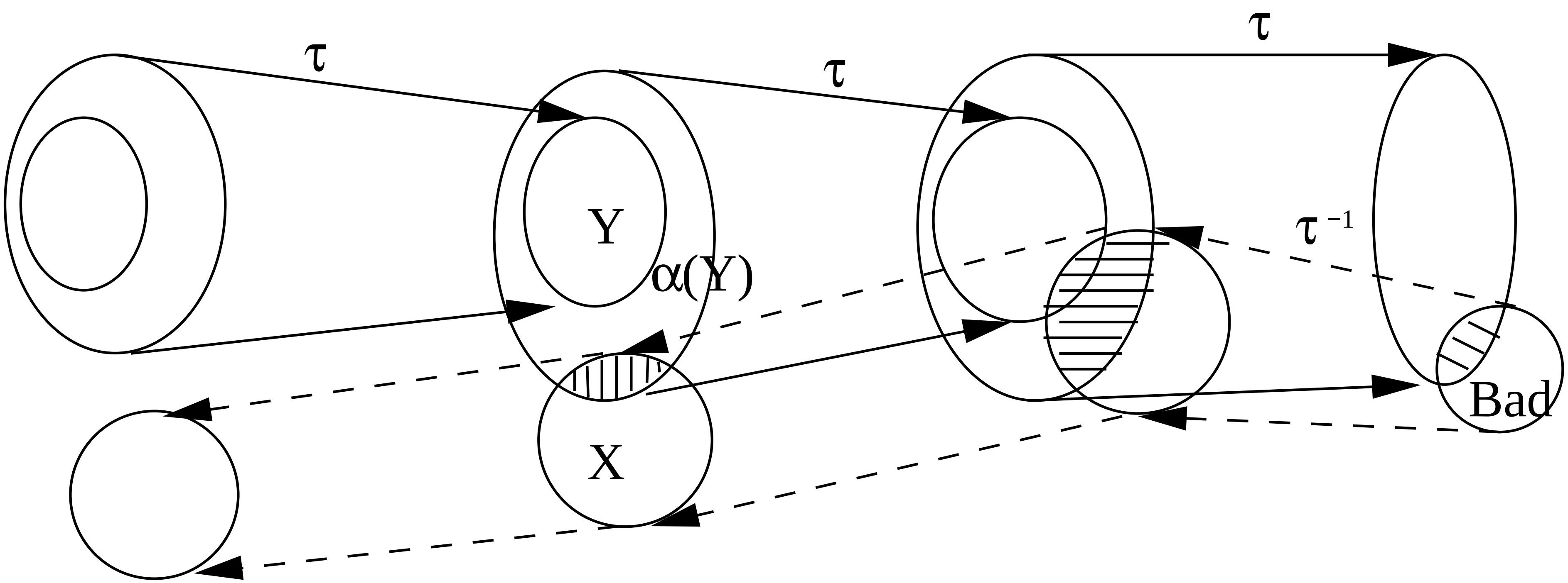
Counter-example guided abstraction refinement



Refinement of α:

Find $\alpha' \subseteq \alpha$ such that: $Y \cap X \neq \emptyset$ implies $\alpha'(Y) \cap X \neq \emptyset$.

Counter-example guided abstraction refinement

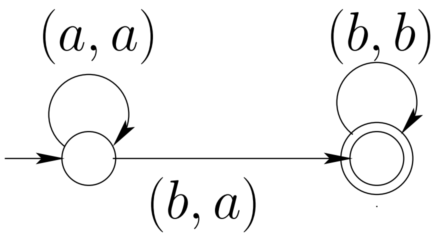


Refinement of α:

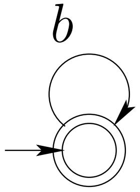
Find $\alpha' \subseteq \alpha$ such that: $Y \cap X \neq \emptyset$ implies $\alpha'(Y) \cap X \neq \emptyset$.

$\Rightarrow$ Take $\mathcal{P}' = \mathcal{P} \cup \{(\alpha(Y) \cap X, q) \ : \ q$ is a state in $\alpha(Y) \cap X\}$

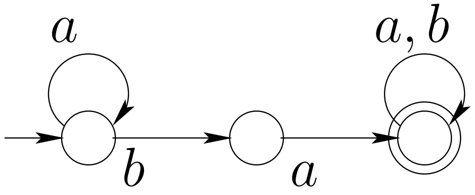
T



A

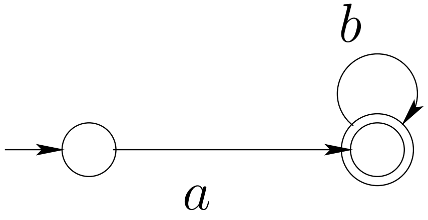


B

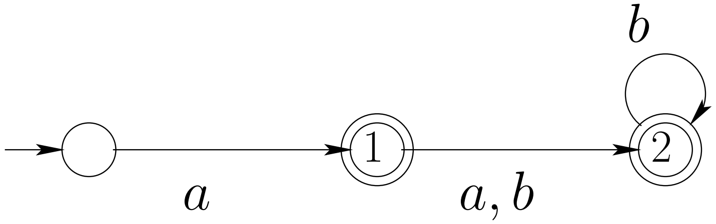


$\mathcal{P} = \{\Sigma^*, \emptyset\}$

$T(A)$

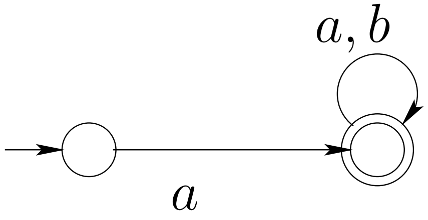


$T^{[1,2]}(A)$

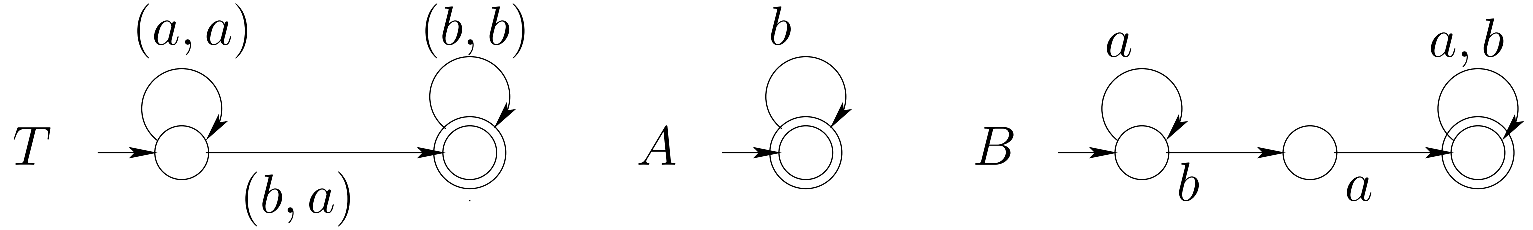


$1 \simeq_{\mathcal{P}} 2$

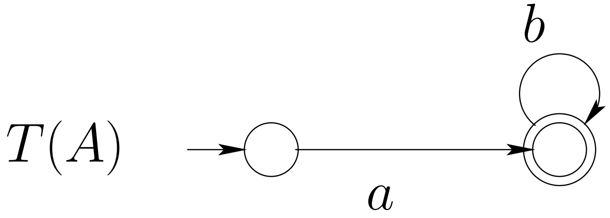
$\alpha_{\mathcal{P}}(T^{[1,2]}(A))$

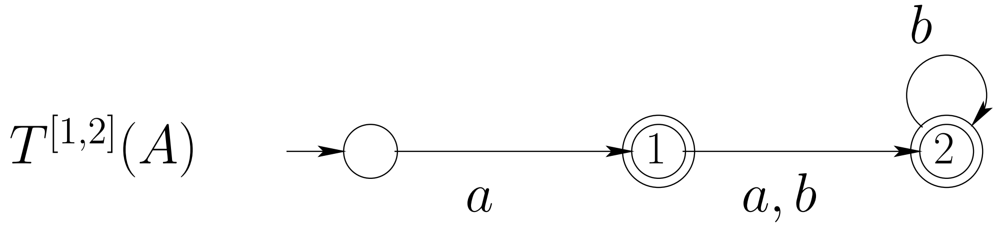


.



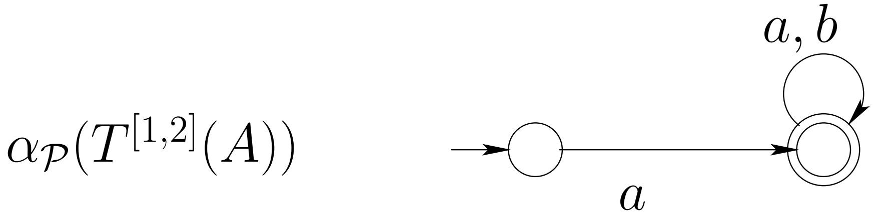
$\mathcal{P} = \{\Sigma^*, \emptyset\}$

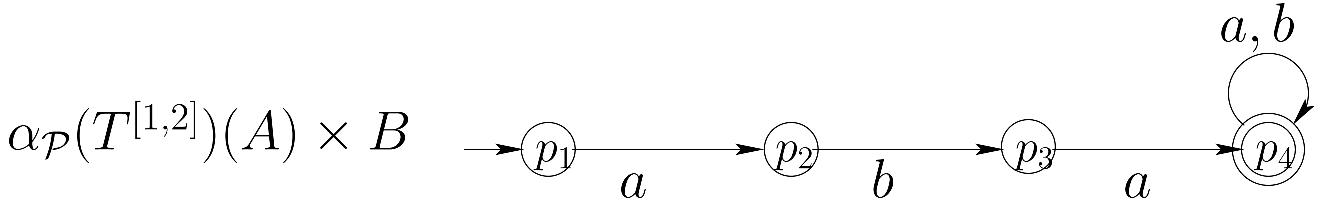




$1 \simeq_{\mathcal{P}} 2$

$1 \not\simeq_{\mathcal{P}'} 2$





$\mathcal{P}' = \mathcal{P} \cup \{p_1, p_2, p_3, p_4\}$

ARMC: References

- The case of words: [B., Habermehl, Vojnar, CAV'04]
- The case of trees: [B., Habermehl, Rogalewicz, Vojnar, Infinity'05]
- Generic framework: parametrized networks of processes, counter systems, FIFO channel systems, etc

ARMC: References

- The case of words: [B., Habermehl, Vojnar, CAV'04]
- The case of trees: [B., Habermehl, Rogalewicz, Vojnar, Infinity'05]
- Generic framework: parametrized networks of processes, counter systems, FIFO channel systems, etc
- ⇒ Application to programs with dynamic linked data structures

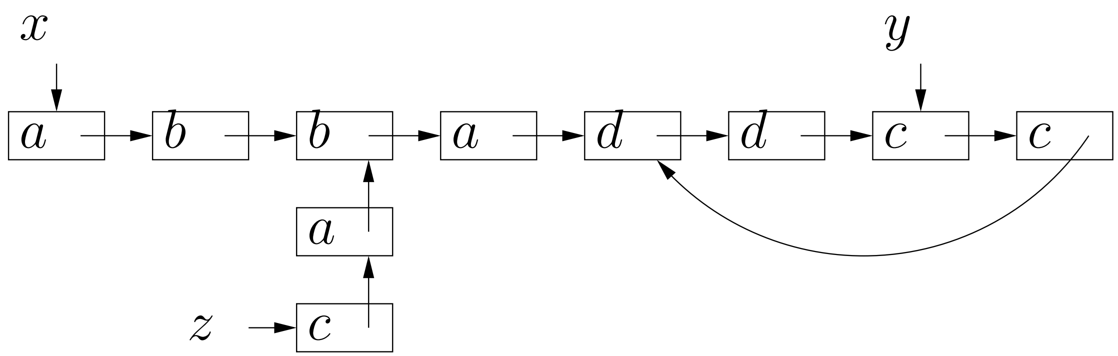
The 1-next selector case

[B., Habermehl, Moro, Vojnar, TACAS'05]

The 1-next selector case

[B., Habermehl, Moro, Vojnar, TACAS'05]

- Heaps are collections of lists with sharing and (non-nested) cycles
- # of sharing points is proportional to # of program variables

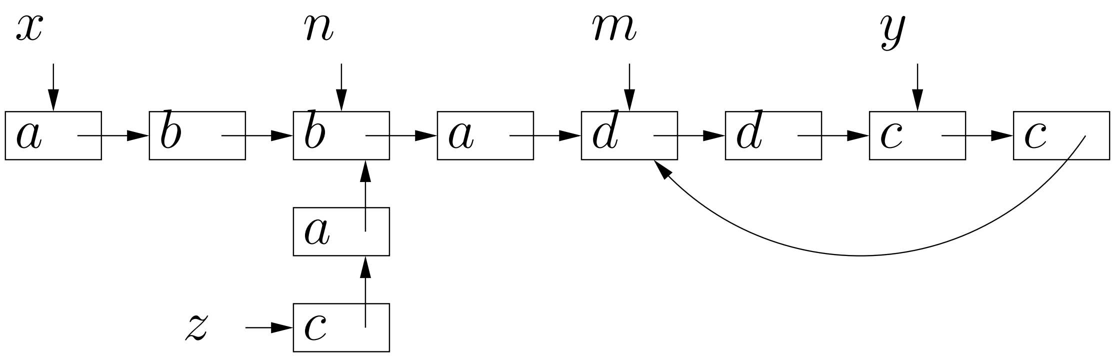


.

The 1-next selector case

[B., Habermehl, Moro, Vojnar, TACAS'05]

- Heaps are collections of lists with sharing and (non-nested) cycles
- # of sharing points is proportional to # of program variables



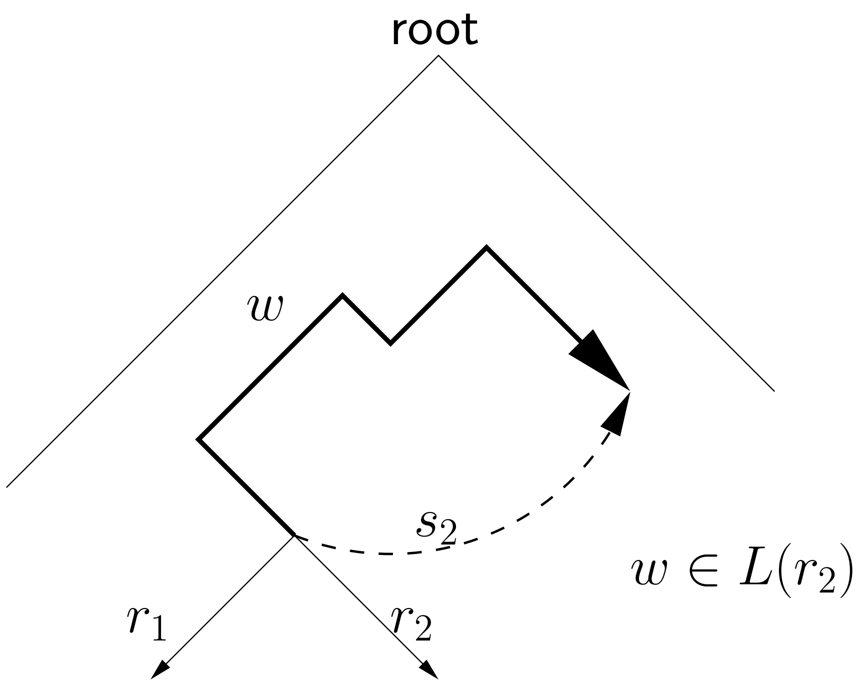
$$x, a \rightarrow b \rightarrow n_t, b \rightarrow a \rightarrow m_t, d \rightarrow d \rightarrow y, c \rightarrow c, m_f \,\|\, z, c \rightarrow a \rightarrow n_f$$

- $\Rightarrow$ a finite (known) number of markers is needed
- Program statements correspond to rewrite steps ($\leadsto$ transducers)
- Automatic management of the markers by the transducers

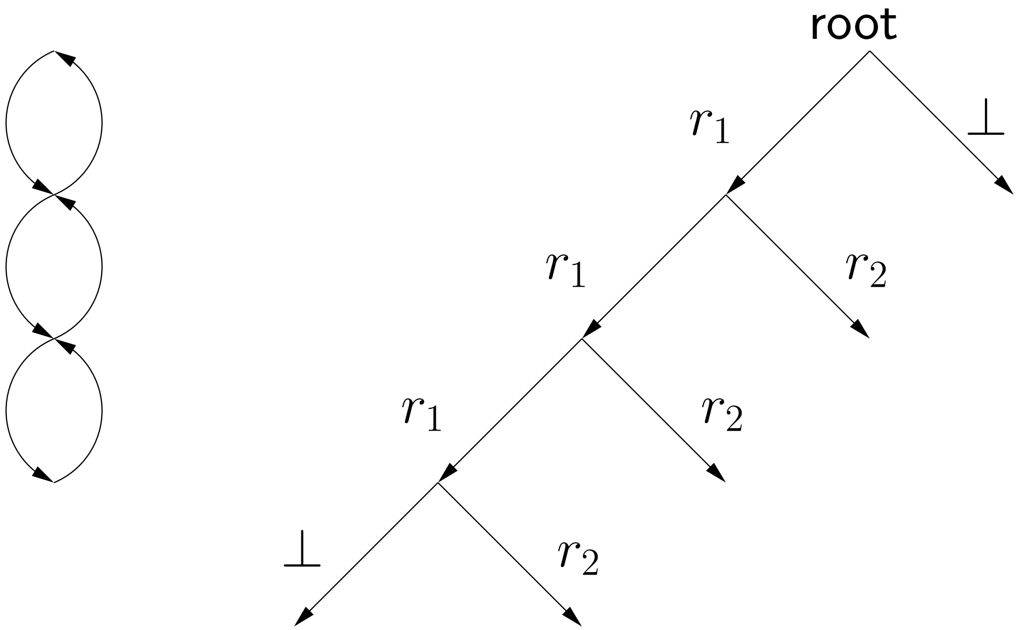
The case of several next selectors

[B., Habermehl, Rogalewicz, Vojnar, SAS'06]

- Several selectors $S = \{s_1, \ldots s_n\}$
- Σ a finite set of labels (data)
- M a finite number of markers (special labels, e.g., root)
- Heap encoded as a n-ary tree backbone + routing expressions
- Routing expression = regular expression over $[1, n] \cup [\overline{1}, \overline{n}] \cup \Sigma \cup M$



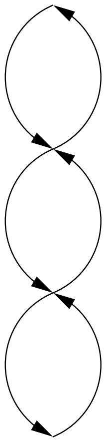
Examples of encodings

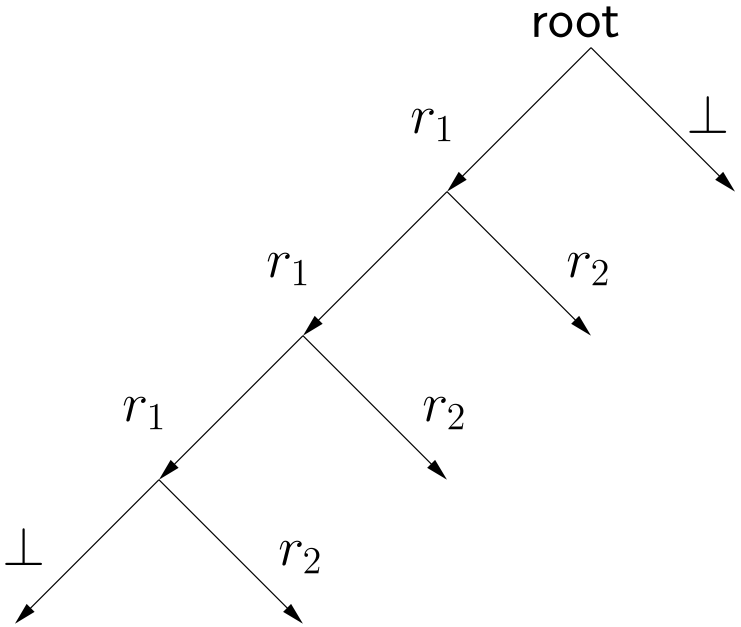


$r_1 = 1$

$r_2 = \overline{1}$

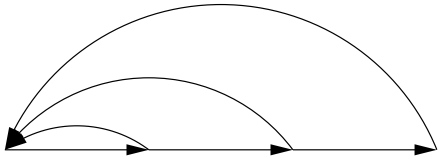
Examples of encodings





$r_1 = 1$

$r_2 = \overline{1}$



$r_1 = 1$

$r_2 = \overline{1}^{+}\text{root}$

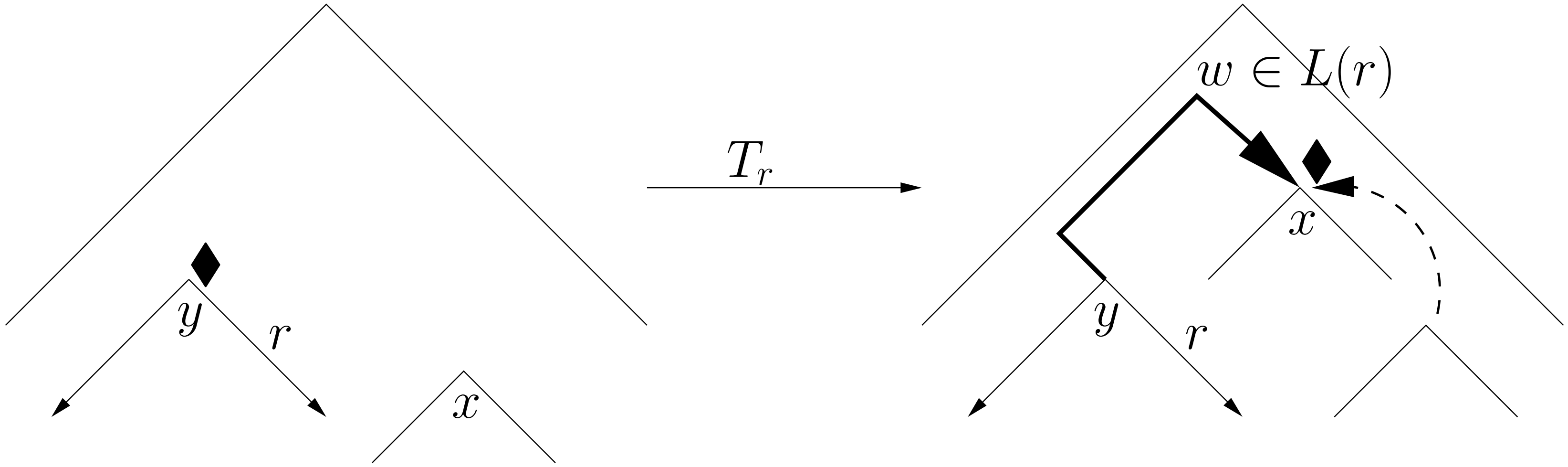
Encoding program statements: Nondestructive updates

Consider the case $x = y \rightarrow s$

- Routing expressions are encoded as tree transducers:

  Given a tree where a source node n is labeled by a special mark ♦, the transducer T_r associated with r moves the mark from n to another node m such that n and m are related by a path in r.

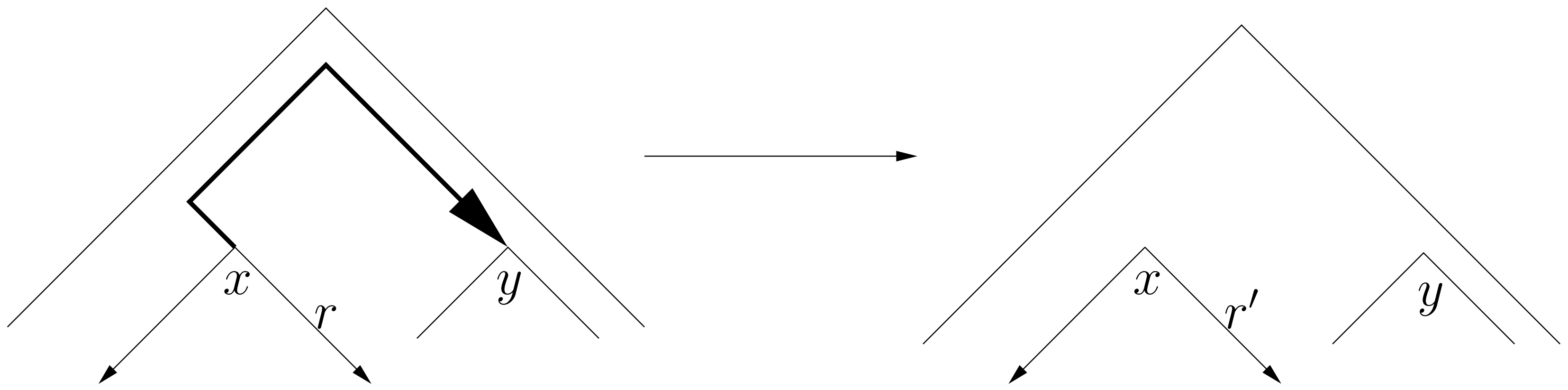
- The mark ♦ is put on the node of y, the transducer T_r is applied, and then x is moved to the marked node. (Composition of transducers.)



Encoding program statements: Destructive updates

Consider the case $x \rightarrow s = y$

- With each statement $x \rightarrow s = y$, there is one associated routing expression
- the s-next pointer below x is labeled by the routing expression of this statement
- the shortest paths relating occurrences of x and y in the trees are added to the routing expression (operation on tree automata building a tree transducer encoding of the new routing expression).



$$r' = r \cup Paths(x, y)$$

- Abstraction on automata ⇒ finite number of routing expressions.

Experimental results

- Implementation based on MONA tree automata libraries
- Experimentation on a 64bit Opteron 2.8 GHz

Example	Time	Abstraction method	$\|Q\|$	N_{ref}
SLL creation	1s	predicates	25	0
SLL reverse	5s	predicates	52	0
DLL delete	6s	finite height	100	0
DLL insert	10s	neighbour	106	–
DLL reverse	7s	predicates	54	0
DLL insertsort	2s	predicates	51	0
Inserting into trees	23s	predicates	65	0
Depth-first search	11s	predicates	67	1
Linking leaves in trees	40s	predicates	75	2
LL insert	5s	predicates	55	0
Deutsch-Schorr-Waite tree traversal	47s	predicates	126	0
TL insert	11mn 25s	finite height	277	0
TL delete	1mn 41s	predicates	420	0

SLL = singly linked lists, DLL = doubly linked lists
LL = list of lists
Task = DLL + pointer to the head. TL = list of tasks.

Related work

- TVLA approach [Sagiv, Reps, Wilhelm, ... , 1998- ...]
- PALE approach [Klarlund, Moller, Schwartzbach, 1993-2001]

- Logic-based approaches:
  Separation logic [O'Hearn, ..., 1999-...],
  Logic of Reachable Patterns [Yorsh, Rabinovich, Sagiv, Meyer, B., 06], etc.
- Several (specialized) techniques for the case of lists (and few for trees)

Conclusion and future work

- Automata-based techniques for shape analysis
- Promising experimental results

- Other abstraction techniques (specialized for the considered domain)
- Improving automata technology
- Data over infinite domains
- Other families of graphs