# Abstract Regular Tree Model Checking

Adam Rogalewicz

- Supervised by
  - Milan Ceska
- Join work with:
  - Tomas Vojnar
  - Peter Habermehl – LIAFA Paris
  - Ahmed Bouajjani – LIAFA Paris

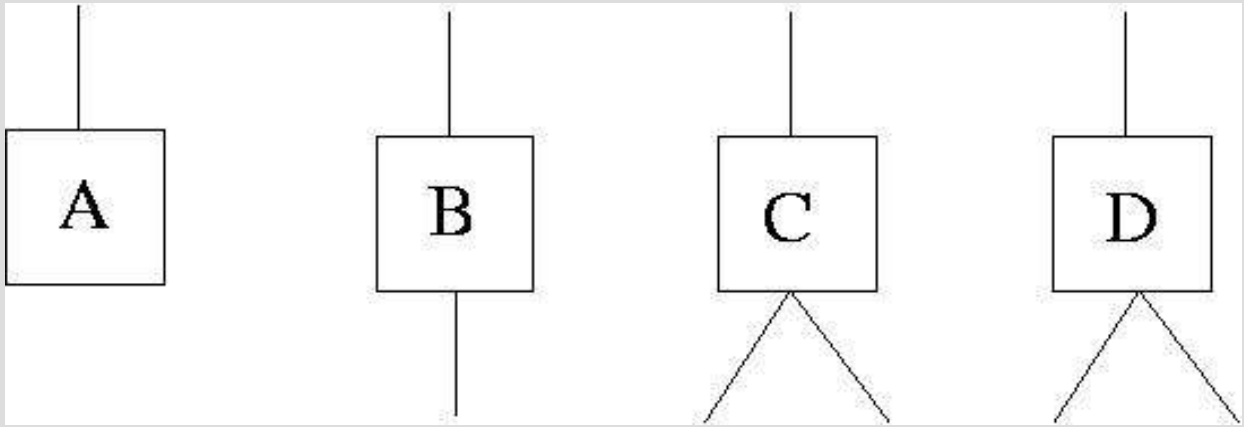
# Program

- Tree automata and transducers
- Abstract regular tree model checking
- Verification of programs with pointers

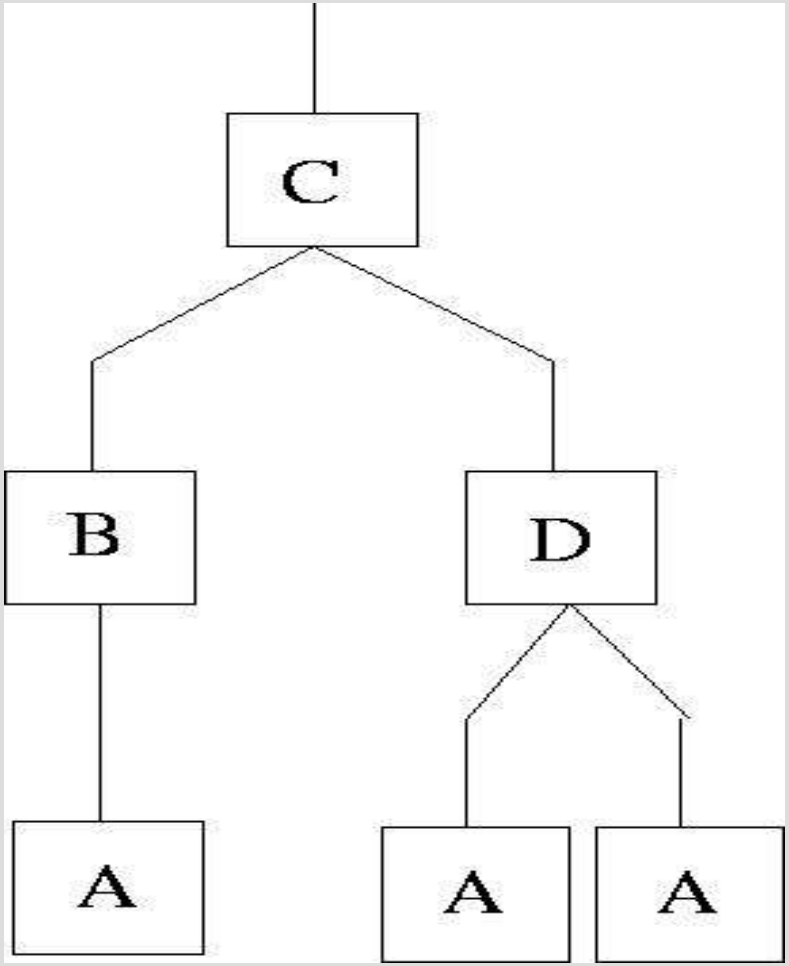
# Tree Languages - Alphabet

- Ranked Alphabet

A:0 B:1 C:2 D:2



# Tree Languages - Trees



# Tree Automata

States:

p,q,r  (r - final state)

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r

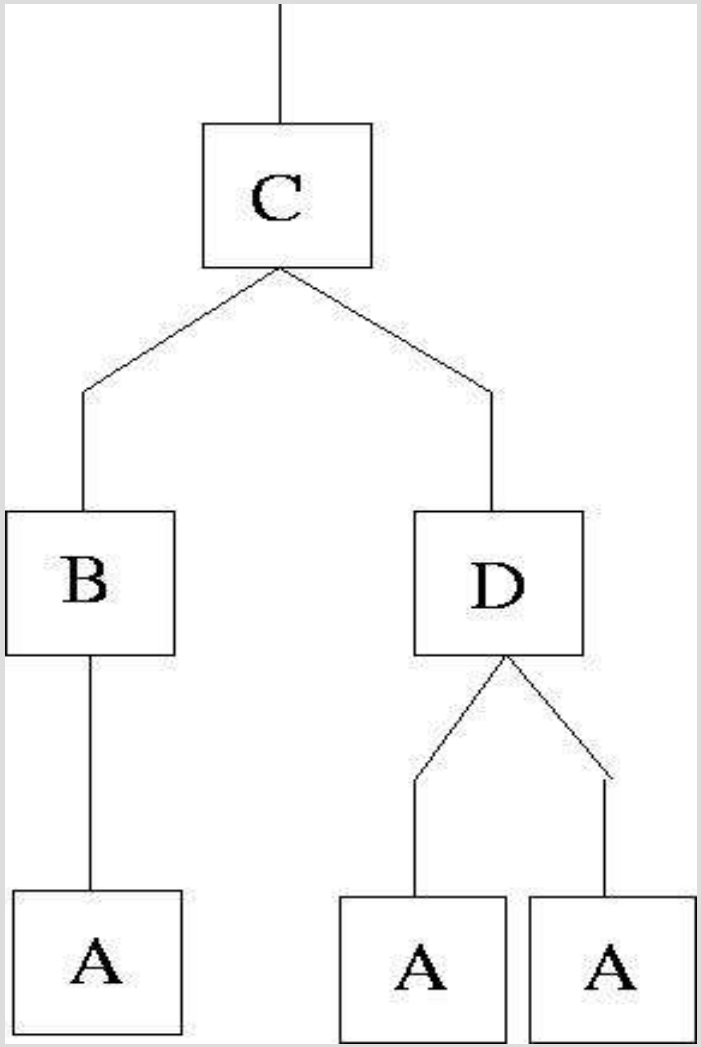
# Tree Automata

States:

p,q,r  (r - final state)

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r



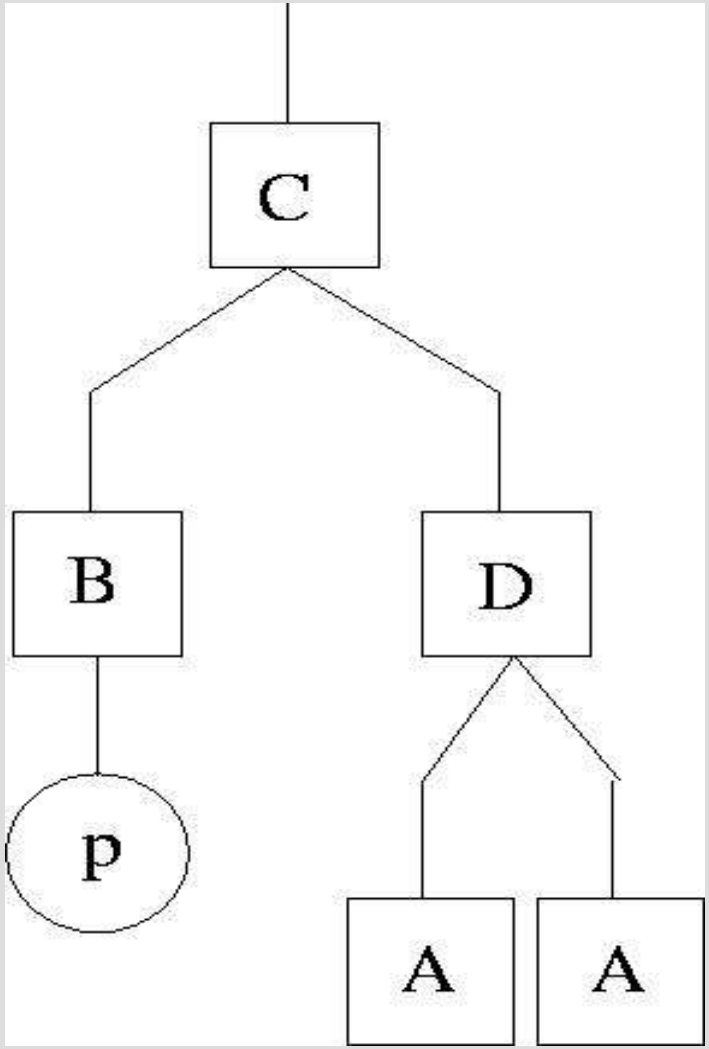
# Tree Automata

States:

p,q,r  (r - final state)

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r



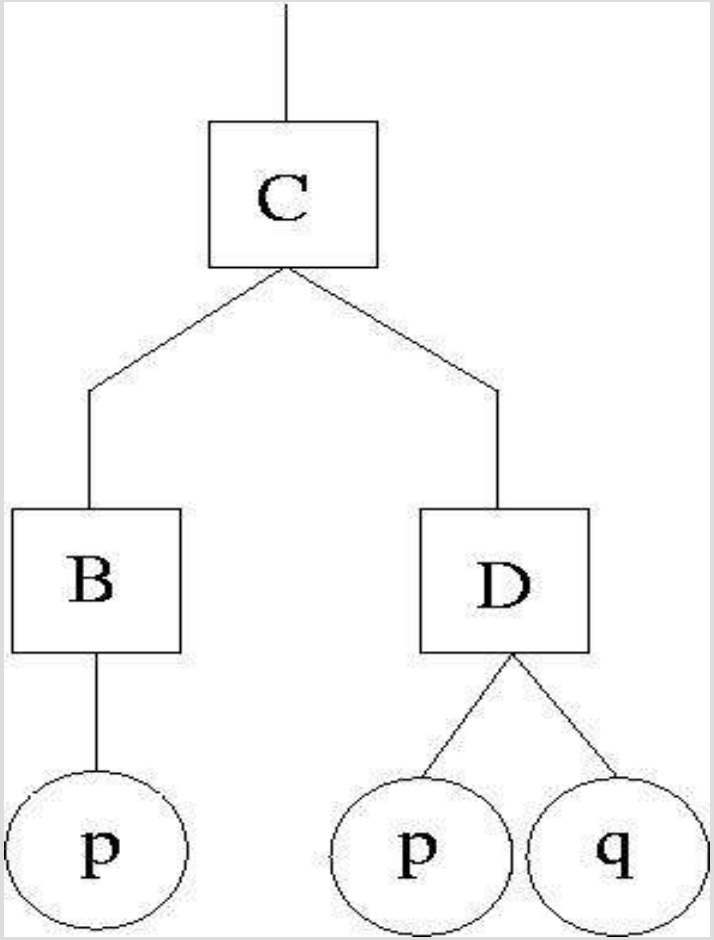
# Tree Automata

States:

p,q,r  (r - final state)

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r



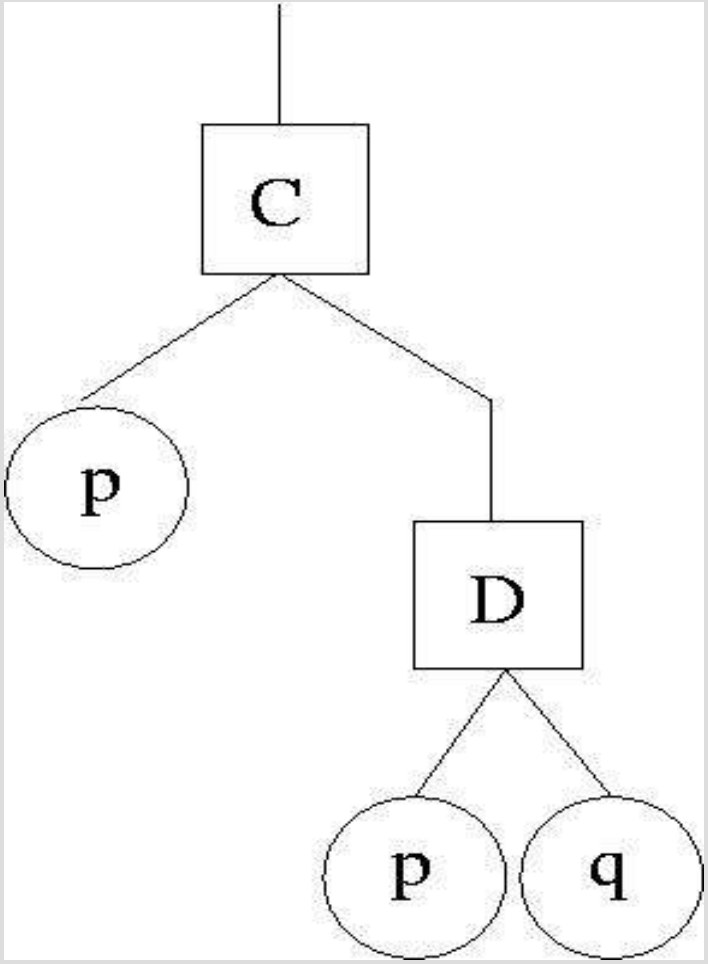
# Tree Automata

States:

p,q,r  (r - final state)

Rules:

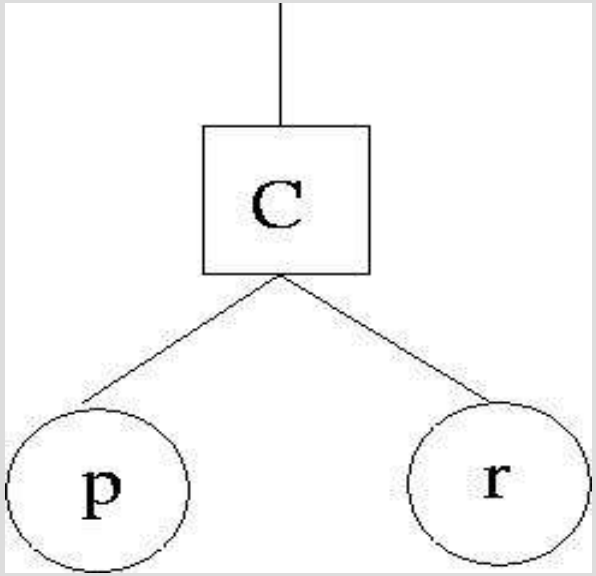
- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r



# Tree Automata

States:

p,q,r  (r - final state)



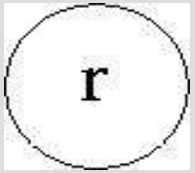

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r

# Tree Automata

States:

p,q,r (r - final state)



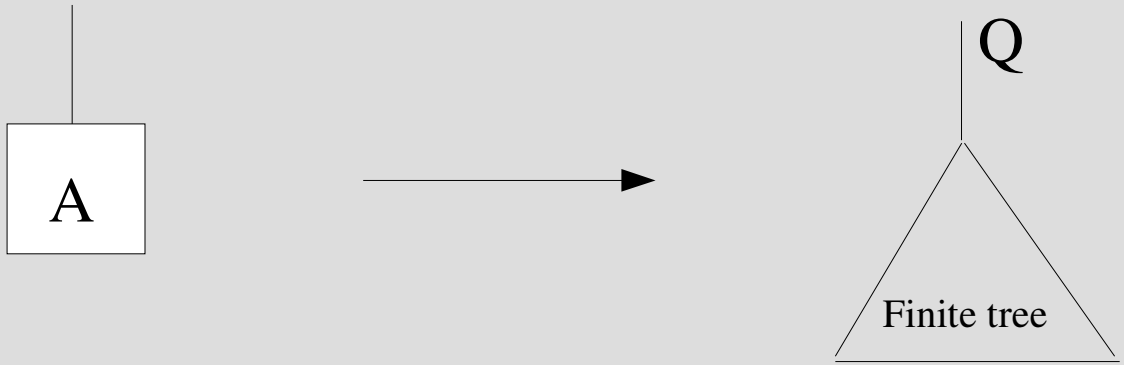

Rules:

- a -> p
- a -> q
- b(p) -> p
- d(p,q) -> r
- c(p,r) -> r

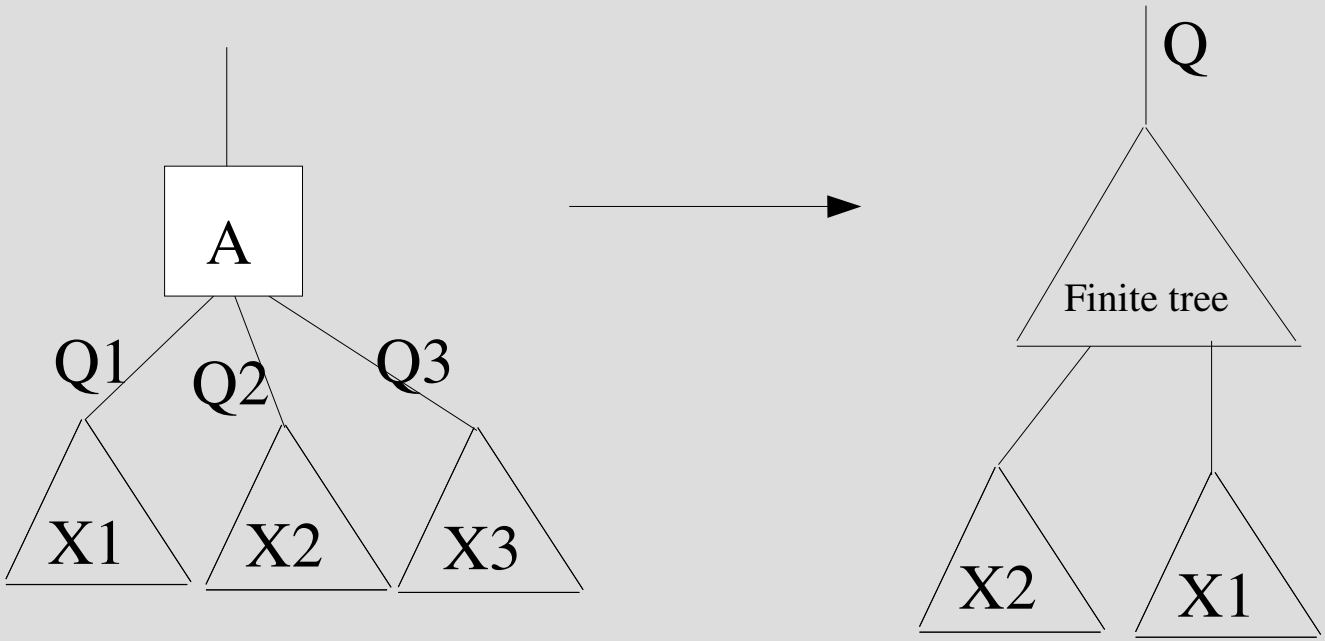
# Tree Transducers

- Finite state machines
- INPUT: a tree automaton
- OUTPUT: a tree automaton
- Describe relation between automata

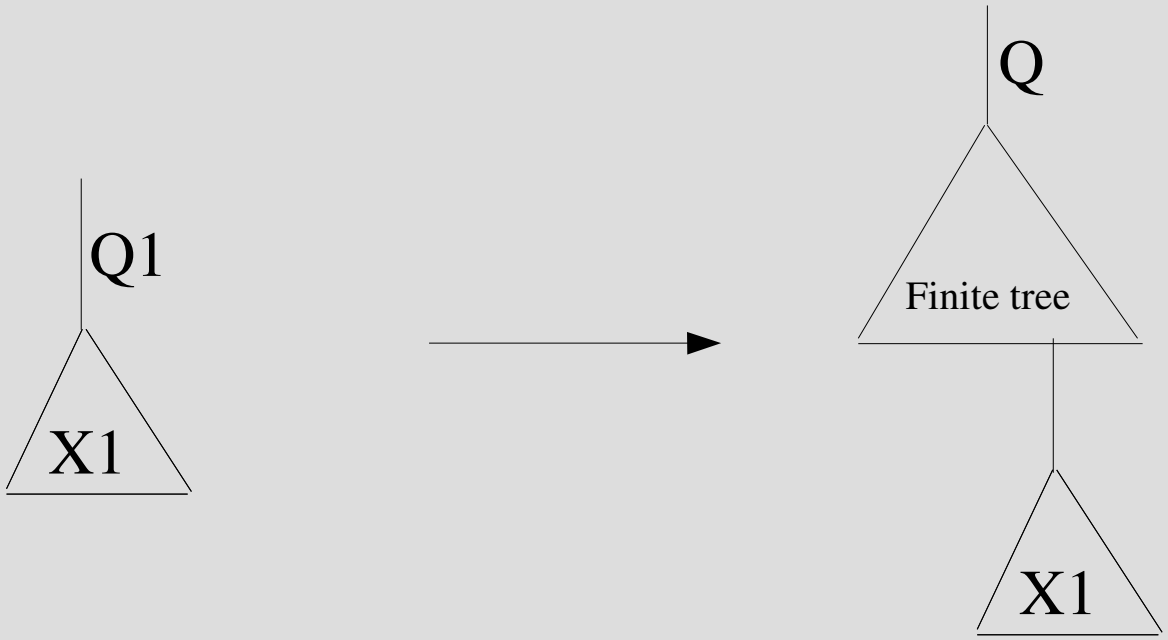
# Tree Transducers - Rules (1)

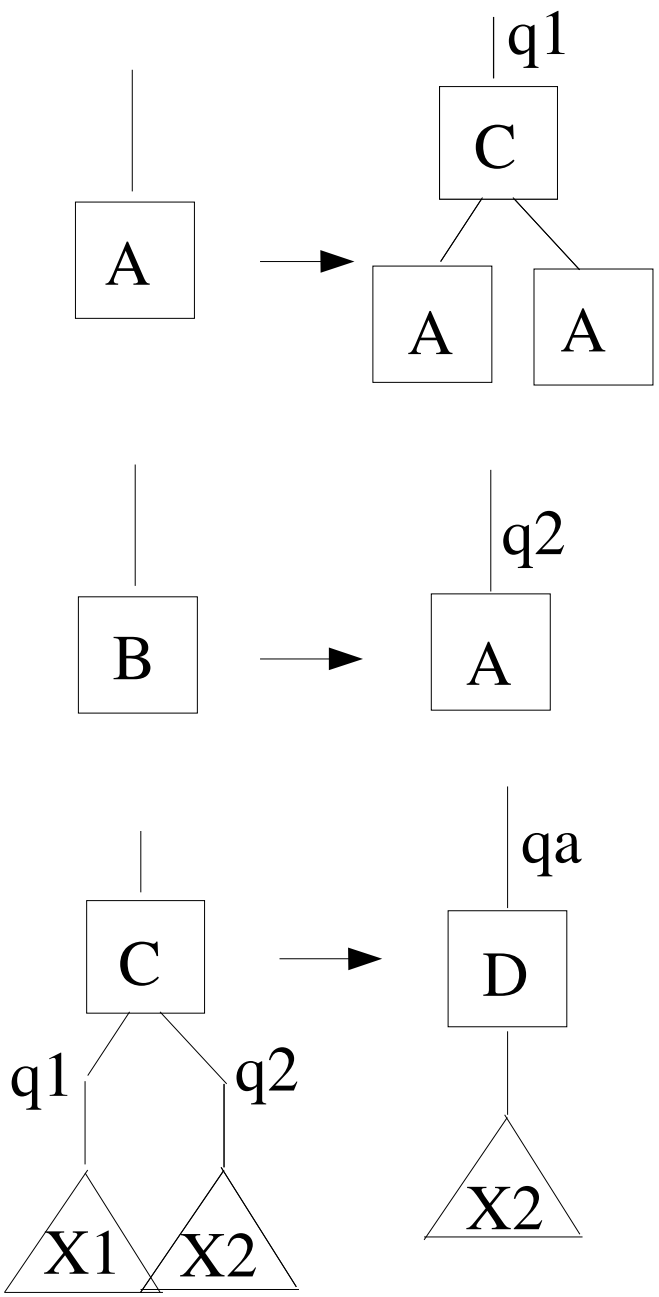


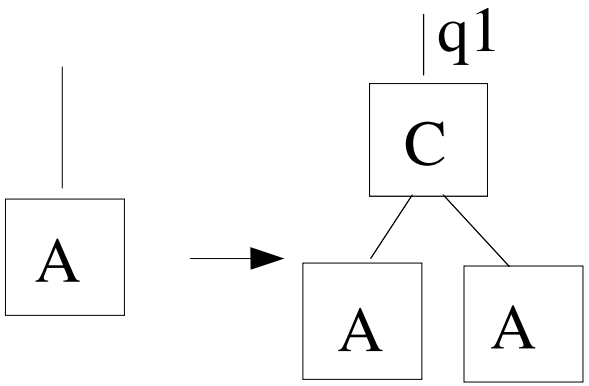
# Tree Transducers - Rules (2)

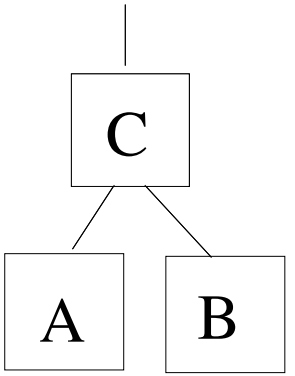


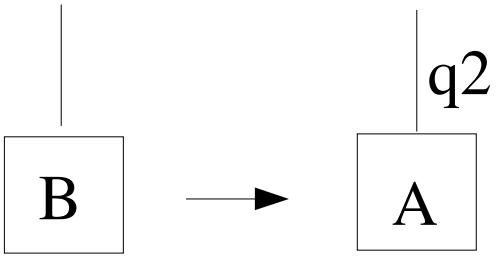
# Tree Transducers - Rules (3)

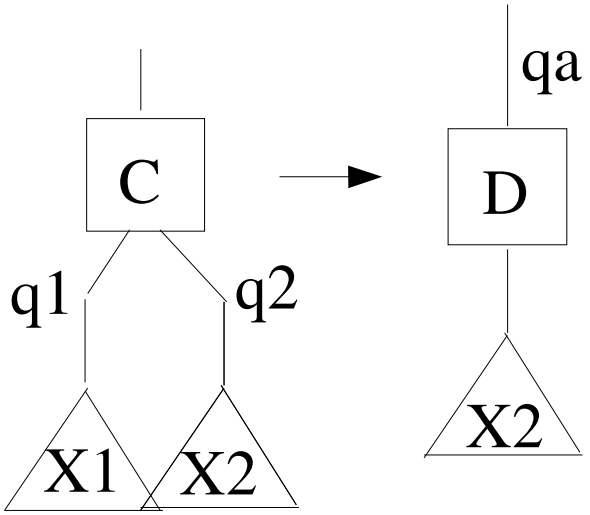


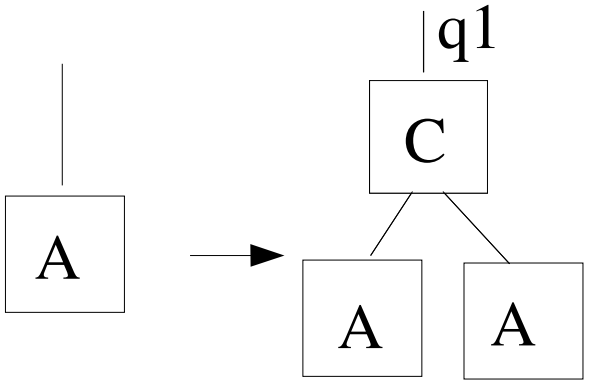

q1
C
A
A
A
q2
B
A
qa
C
D
q1
q2
X1
X2
X2

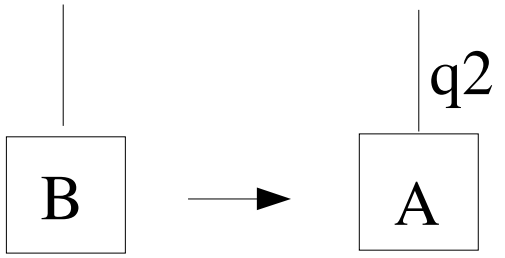

q1
C
A
A
A

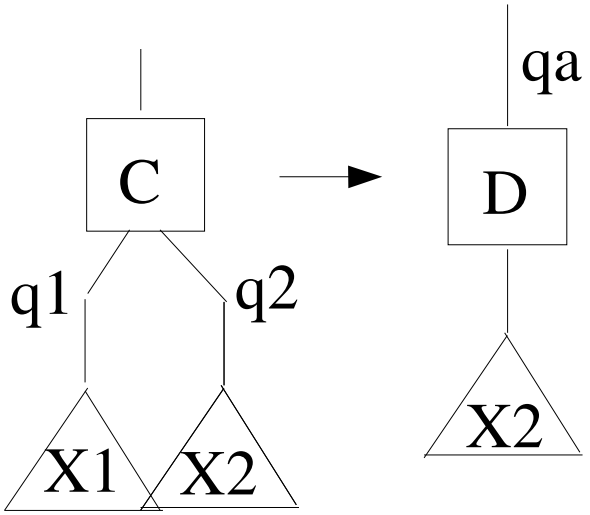

C
A
B

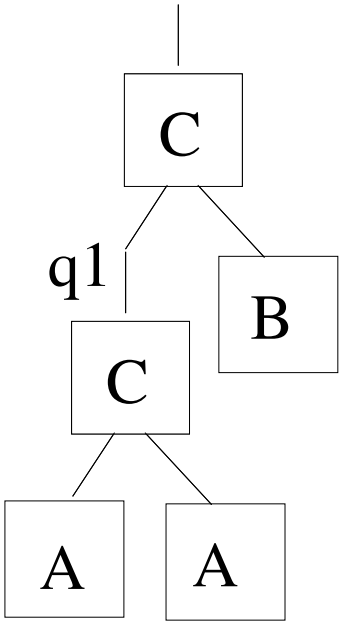

q2
B
A

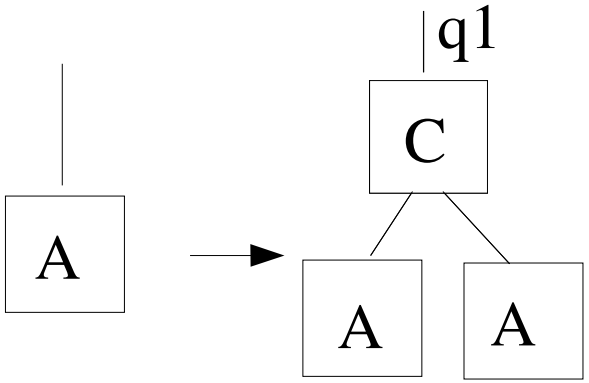

qa
C
D
q1
q2
X1
X2
X2

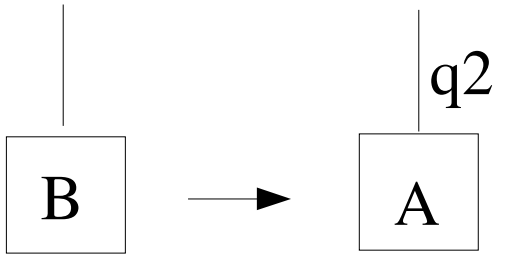

q1
C
A
A
A

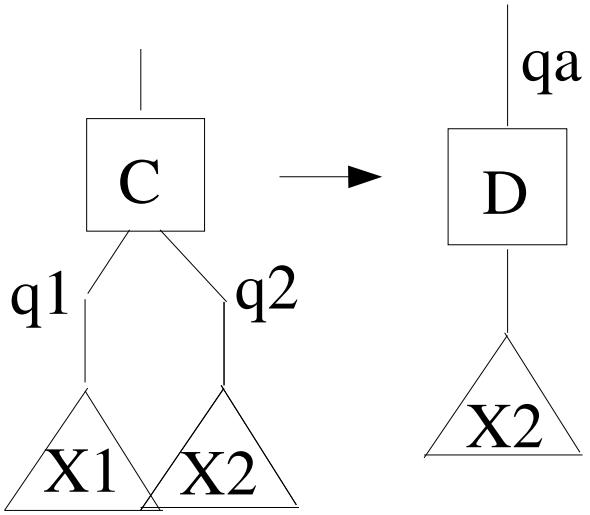

q2
B
A

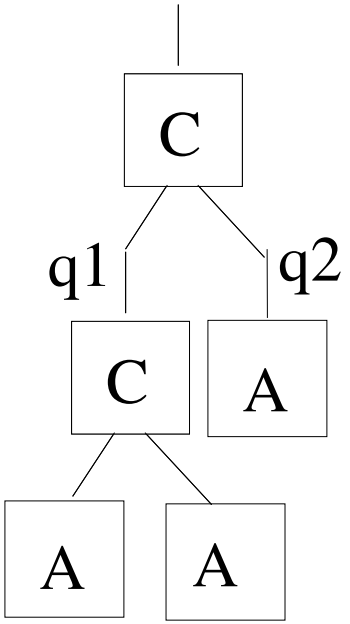

qa
C
D
q1
q2
X1
X2
X2

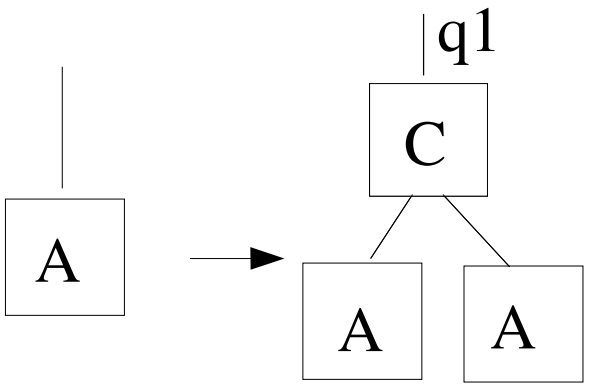

C
q1
B
C
A
A

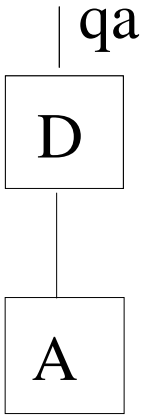

q1
C
A
A
A

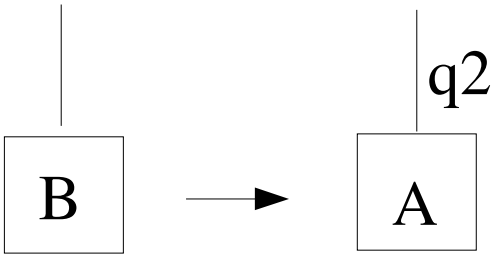

q2
B
A

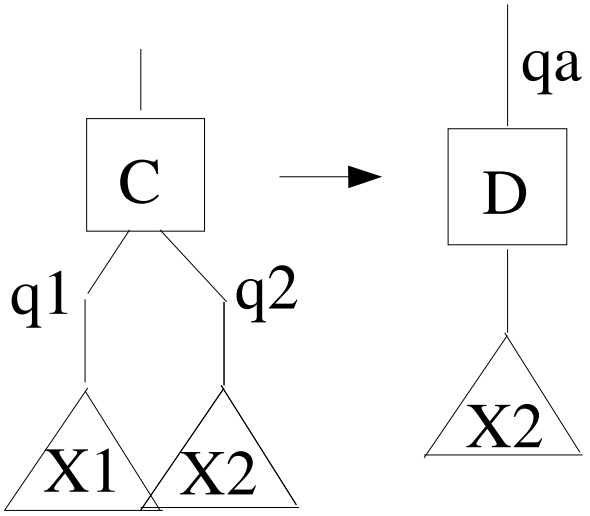

qa
C
D
q1
q2
X1
X2
X2


C
q1
q2
C
A
A
A


q1
C
A
A
A


qa
D
A


q2
B
A


qa
C
D
q1
q2
X1
X2
X2

# Linear Tree Transducers

- Rules can not duplicate parts of trees

- Closed for composition
- Allow to compute post- and pre- for a given tree automaton

# Structure Preserving Tree Transducers

- Just change symbols in nodes
- Inverse relation is also structure preserving transducer
- It is enaugh for many interesting problems

# Regular Tree Model Checking

- Program configuration = tree
- Set of configurations = tree automata
- Behaviour = tree transducer

- Set of initial configurations = tree automata
- Set of bad configurations = tree automata

# Regular Tree Model Checking: Verification Problem

$$\tau^*(\mathrm{Init}) \cap \mathrm{bad} = \emptyset$$

# Regular Tree Model Checking: Verification Problem

$$\tau^*(\text{Init}) \cap \text{bad} = \varnothing$$

- In general undecidable
- Partial methods
  - Widening
  - Creation of history transducers
  - Abstractions on automata

# Abstraction on Automata

- GOAL: Simplify a automaton
- Abstraction function $\alpha$

$$L(A) \subseteq L(\alpha(A))$$

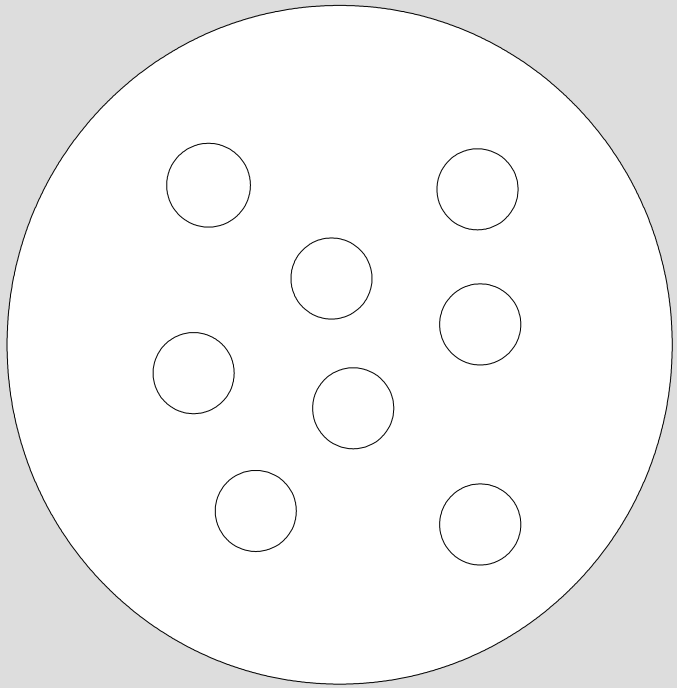
# Abstraction on Automata (2)

- Abstraction based on state collapsing:

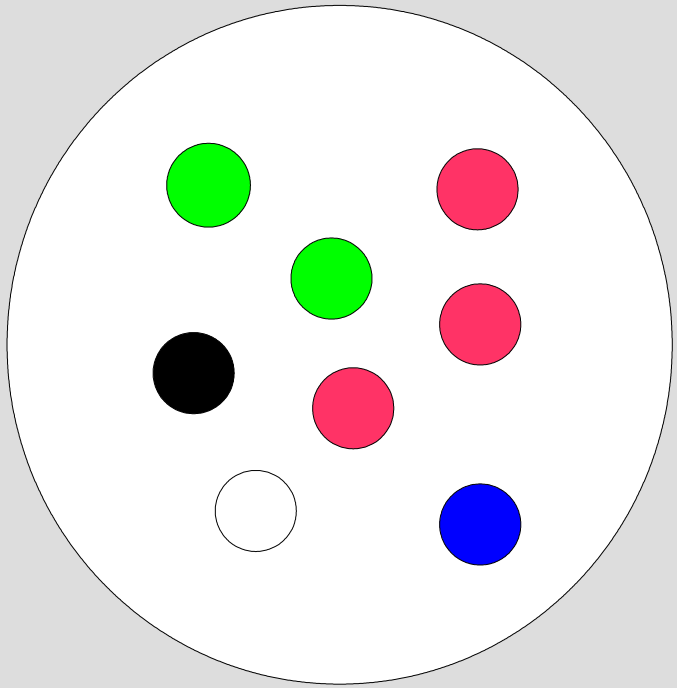
  several states -> one new state

- Equivalence relation on states

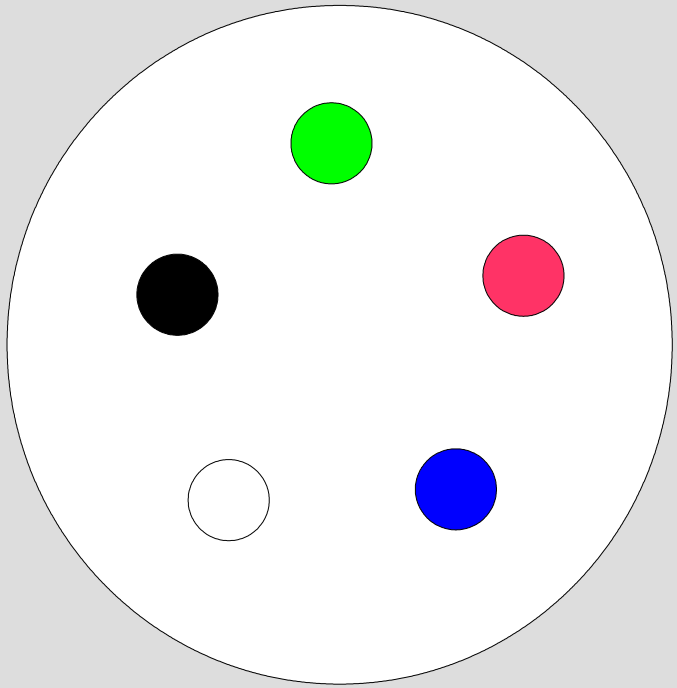
# Abstraction on Automata (3)



# Abstraction on Automata (3)



# Abstraction on Automata (3)



# Abstractions on Automata (4)

- Equivalence relation is based on languages accepted by states

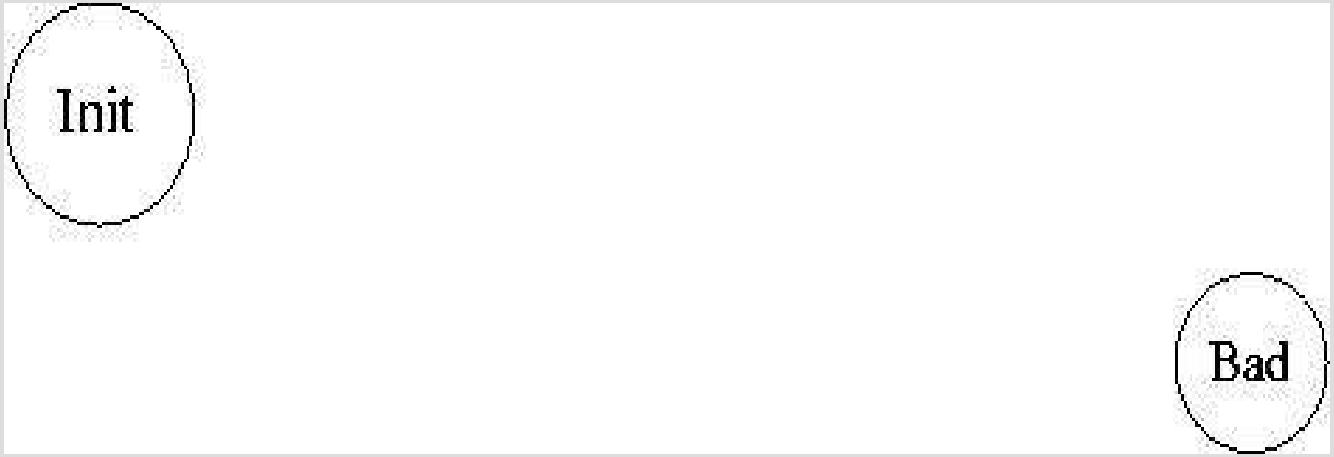
- Languages of finite depth
- Predicate languages

# Abstract Regular Tree MC

- Generalization of abstract regular MC [Bouajjani,...]
- Coputation of overapproximation of $\tau^*(\text{Init})$

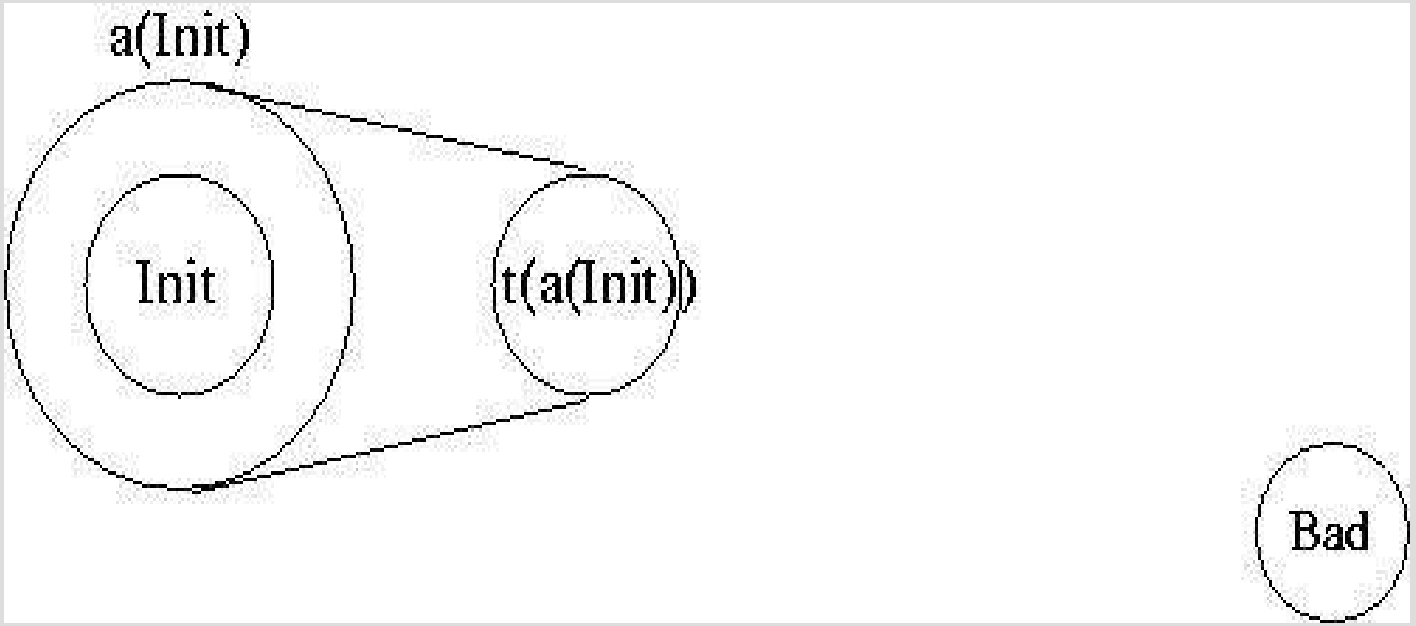
$$O(\tau^*(\text{Init})) \cap \text{bad} = \emptyset$$

# Abstract Regular Tree MC

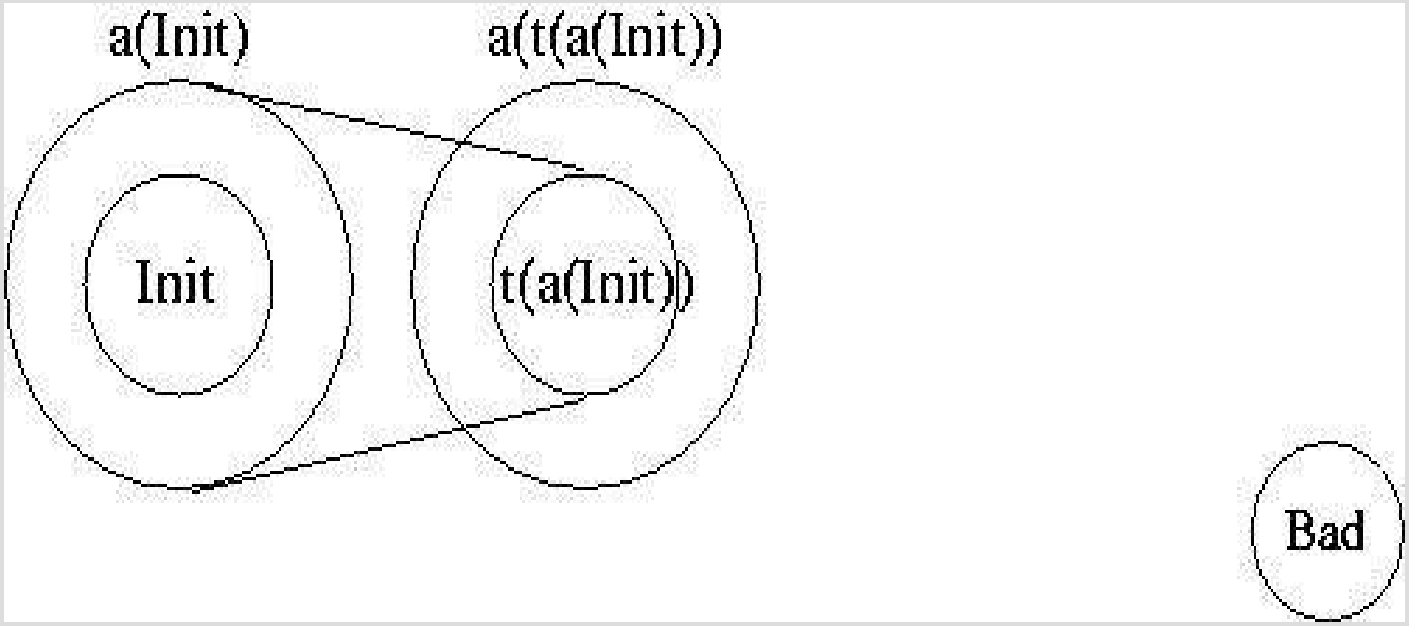


# Abstract Regular Tree MC

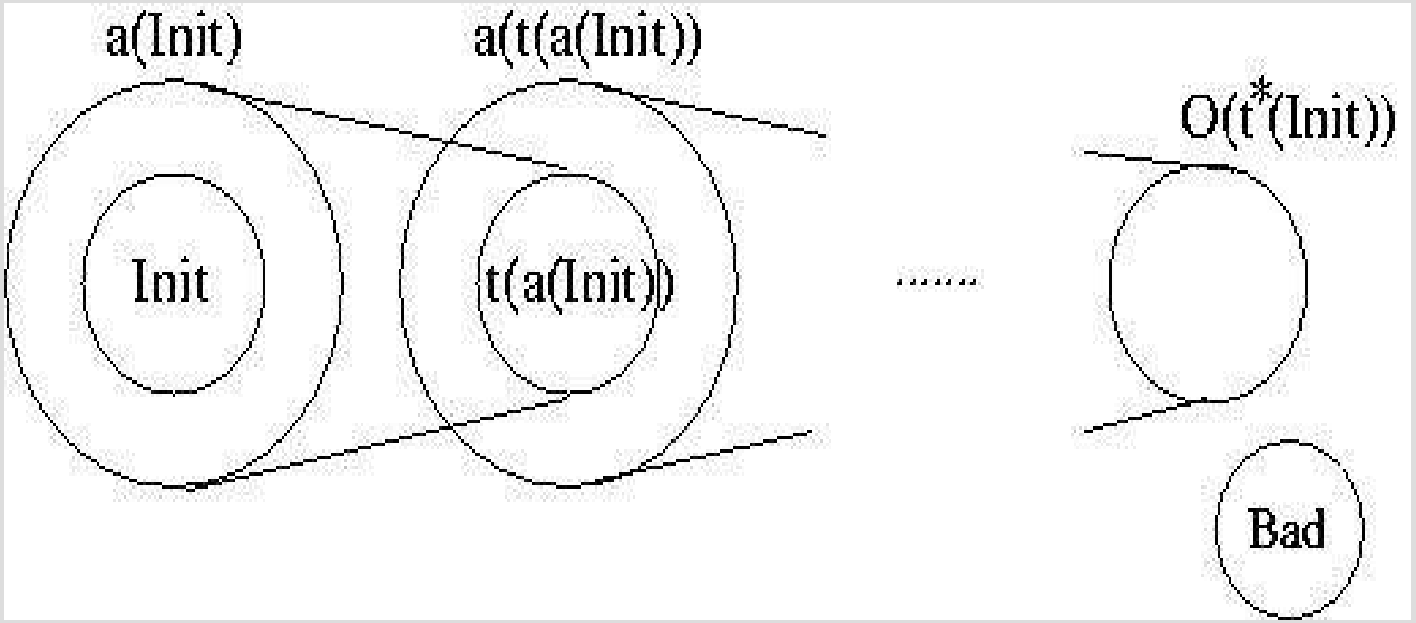
# Abstract Regular Tree MC



# Abstract Regular Tree MC



# Abstract Regular Tree MC



# Verified Protocols

| | Length abstractio | Predicate based abstractio |
|---|---|---|
| **Token passing** | backwards: 0.08s | forwards: 0.06s |
| **Two-way token passin** | backwards: 1.0s | forwards: 0.09s |
| **Percolate** | backwards: 20.8s | forwards: 2.4s |
| **Tree arbiter** | backwards: 0.31s | backwards: 0.34s |
| **Leader election** | backwards: 2.0s | forwards: 1.74s |
| **Broadcasting** | backwards: 9.1s | forwards: 1.0s |

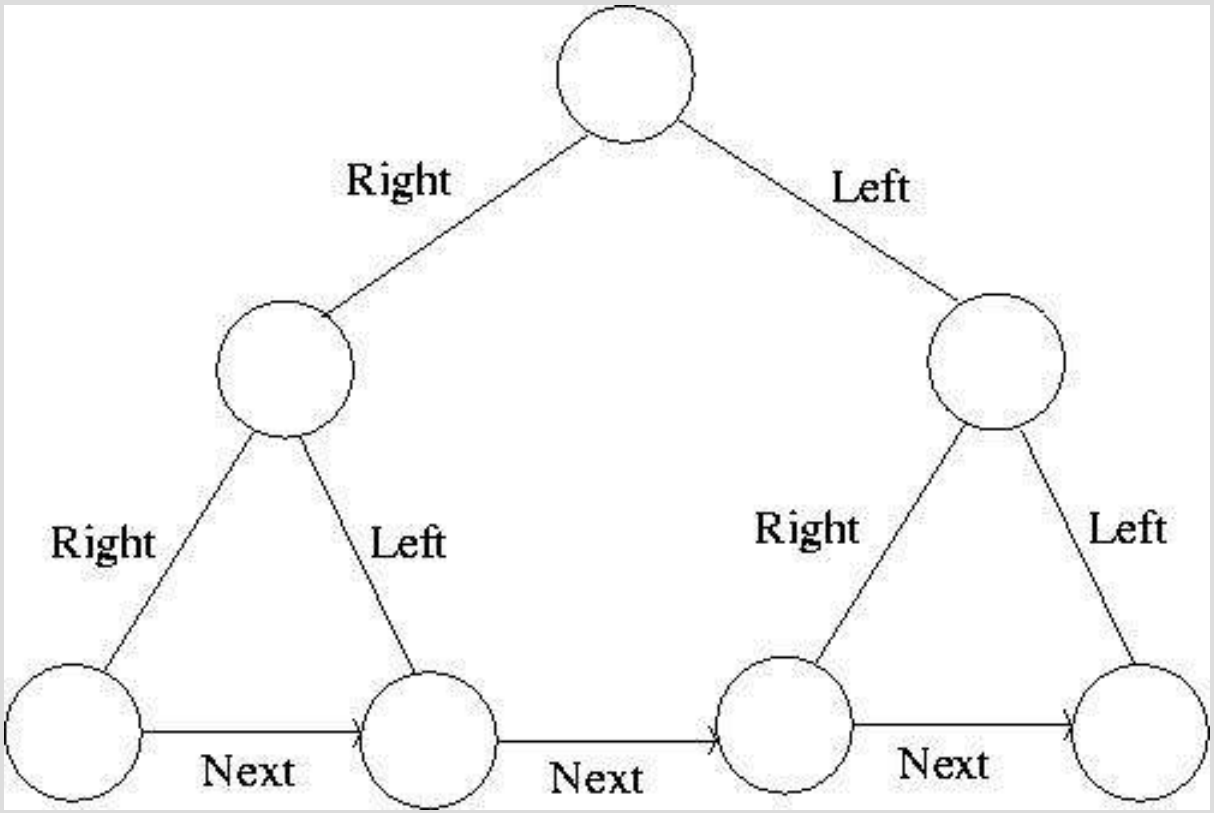
# Possible Use of ARTMC

- Programs with pointers
- XML manipulations
- Cryptographic protocols
- Network broadcasting
- Systems with dynamic process creation

# ARTMC and Programs with Pointers

- Inspirated by use of ARMC for programs with 1-selector linked lists [Bouajjani,...]
- In general, data structure is a directed graph
  => It is necessary to have unbounded number of "extra pointers"

# Programs with Pointers: Tree with Linked Lists



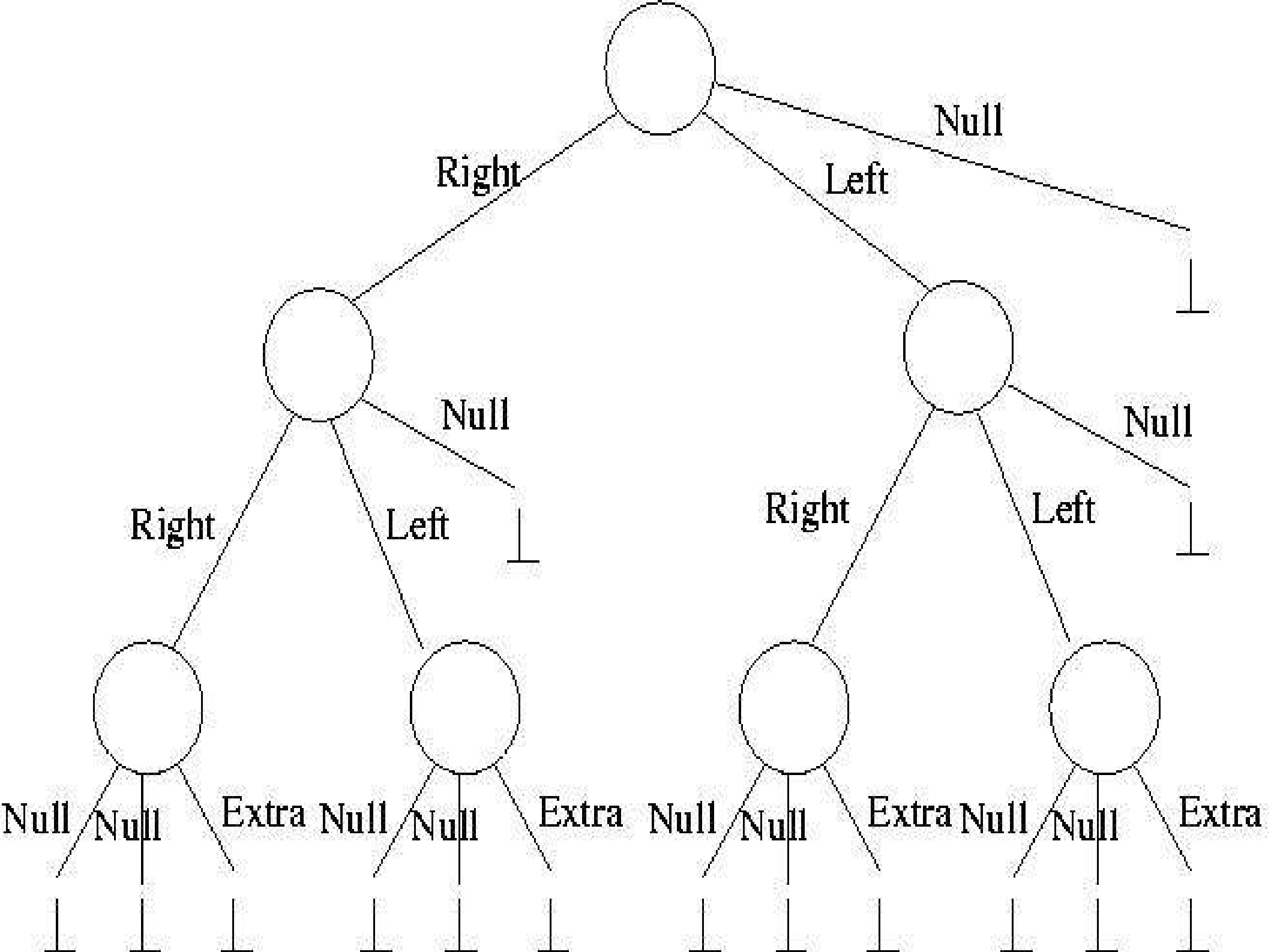
# Programs with Pointers

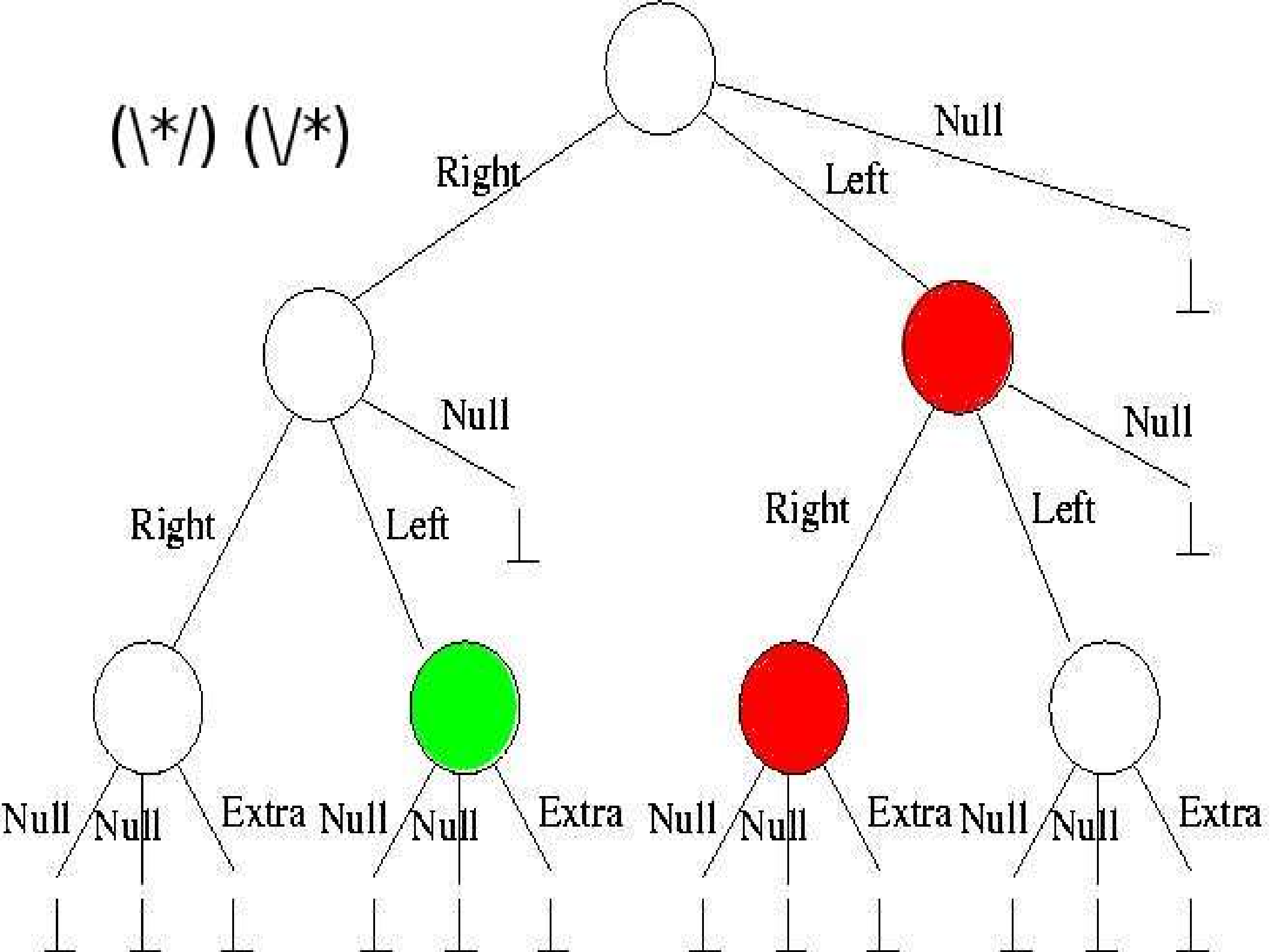
- Tree is used just like a backbone
- Pointer destination is encoded by so-called "pointer descriptors"
    - Pointer descriptor describes destination relatively to the tree shape.
    - Each pointer descriptor has an "UP" part, and "DOWN" part

# Programs with Pointers: Tree with Linked Lists

It is necessary to have 3 descriptors:

- left - “ε  /”
- right - “ε  \”
- extra - “(*/)  (\/*)”


Null
Right
Left
Null
Null
Right
Left
Null
Right
Left
Null
Null
Extra
Null
Null
Extra
Null
Null
Extra
Null
Null
Extra


(*/) (\/*)
Right
Left
Null
Null
Right
Left
Null
Right
Left
Null
Null
Null
Extra
Null
Null
Extra
Null
Null
Extra
Null
Null
Extra

# Programs with Pointers: Destination of the pointer

Descriptor “extra” in previews example is not deterministic -> there is more possible destinations

- Combination of descriptors and markers
  - Descriptor shows possible destinations
  - Marker restrict them

- Still not deterministic => We use all possibilities

# Programs with pointers – operations on the structure

- All operation except x.next=y can be performed by tree transducers
- representation is not closed for x.next=y – there in not guarantee of existence of suitable descriptor

# x.next=y

- Reuse of existing pointer descriptor (if exists)
  - can be performed by tree transducers
- Refine set of pointer descriptors
  - Add new one
  - Increase power of existing one
    - It is necessary to create new transducer

# Programs with pointers - state of research

- Ongoing implementation of convertor from programs to tree transducers in Mona GTA library
- PLAN: paper for TACAS 2006

# Publications

- Rogalewicz, Vonar: Tree Automata in Modelling and Verification of Concurrent Programs – ASIS 2004
- Rogalewicz: Towards Applying Mona In Abstract Regular Tree Model Checking – EEICT 2005

- Bouajjani, Habermehl, Rogalewicz, Vojnar: Abstract Regular Tree Model Checking – submitted to Infinity 2005