Higher-Order Nonemptiness Step by Step

Paweł Parys

University of Warsaw

RP 2022 (based on FST-TCS 2020 & ICALP 2021 papers)

What is it about?

Higher-Order = we consider higher-order recursion schemes

Nonemptiness = we solve the acceptance problem for alternating reachability automata (= language nonemptiness)

Step by Step = we give a new method, working in multiple simple steps

Higher-order recursion schemes – what is this?

Definition

Higher-order recursion schemes = a generalization of context-free grammars, where nonterminals can take arguments. We use them to generate trees.

Equivalent definition: simply-typed lambda-calculus + recursion

In other words:
- programs with recursion
- higher-order functions (i.e., functions taking other functions as parameters)
- every function/parameter has a fixed type
- no data values, only functions

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$S \rightarrow A\, b$$
$$A\, f \rightarrow a\, (A\, (D\, f))\, (f\, c)$$
$$D\, f\, x \rightarrow f\, (f\, x)$$

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

$$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$$

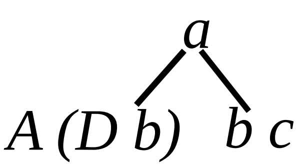
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D



Rules:

$$S \rightarrow A\, b$$
$$A\, f \rightarrow a\, (A\, (D\, f))\, (f\, c)$$
$$D\, f\, x \rightarrow f\, (f\, x)$$

$$S \rightarrow A\, b \rightarrow a\, (A\, (D\, b))\, (b\, c)$$

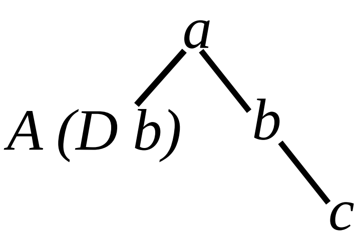
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D



Rules:

$S \rightarrow A\ b$

$A\ f \rightarrow a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \rightarrow f\ (f\ x)$

$S \rightarrow A\ b \rightarrow a\ (A\ (D\ b))\ (b\ c)$

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

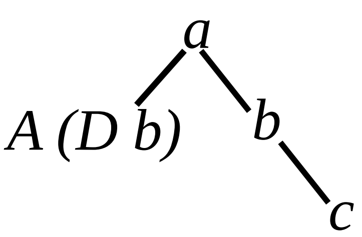
a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$



$$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$$
$$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$$

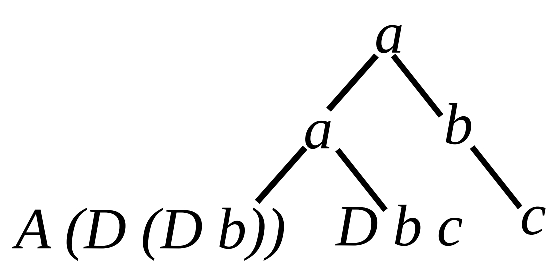
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D



Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

$$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$$
$$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$$

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

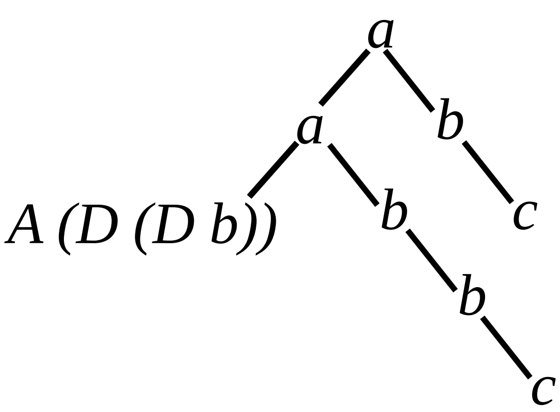
a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$



$$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$$
$$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$$
$$D\ b\ c \to b\ (b\ c)$$

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

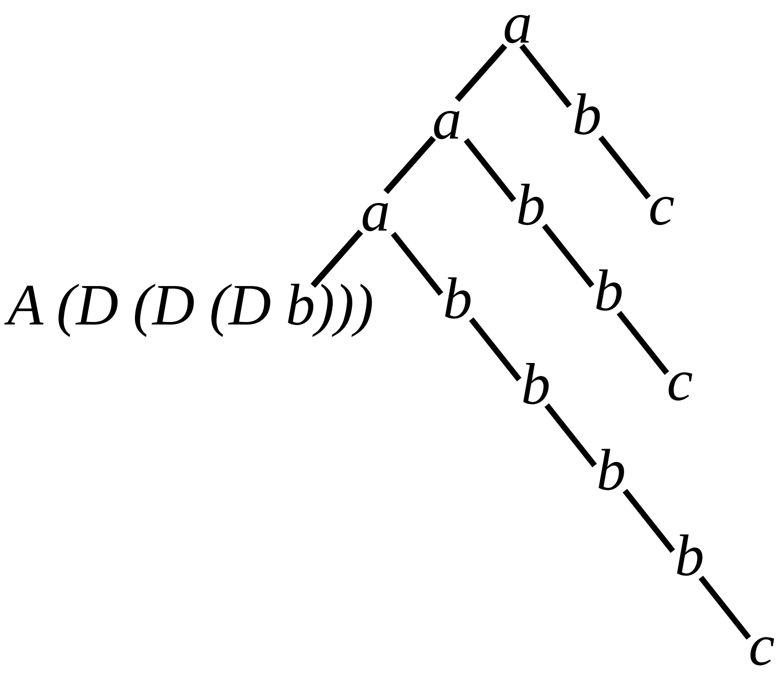
S (starting), A, D

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$



$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

$D\ b\ c \to b\ (b\ c)$

$A\ (D\ (D\ b)) \to a\ (A\ (D\ (D\ (D\ b))))\ (D\ (D\ b)\ c)$

$D\ (D\ b)\ c \to D\ b\ (D\ b\ c) \to b\ (b\ (D\ b\ c))$

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

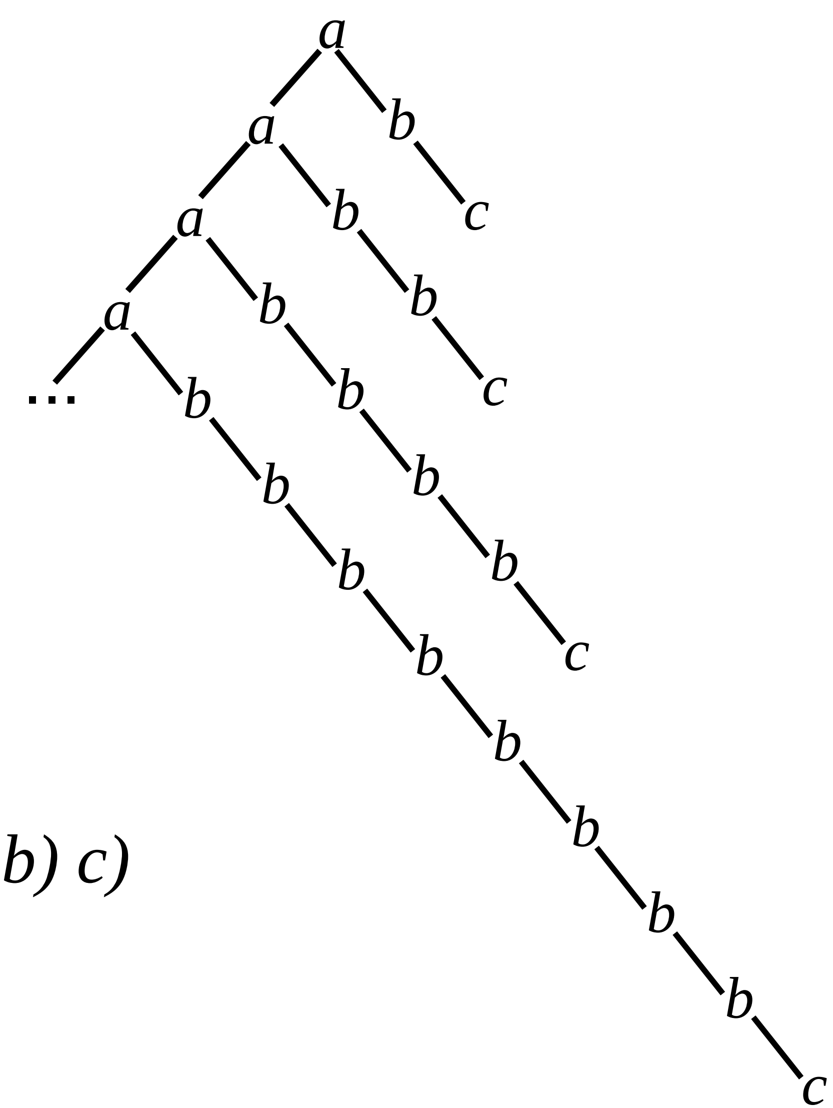
S (starting), A, D

Rules:

$S \to A\ b$

$A\ f \to a\ (A\ (D\ f))\ (f\ c)$

$D\ f\ x \to f\ (f\ x)$



$S \to A\ b \to a\ (A\ (D\ b))\ (b\ c)$

$A\ (D\ b) \to a\ (A\ (D\ (D\ b)))\ (D\ b\ c)$

$D\ b\ c \to b\ (b\ c)$

$A\ (D\ (D\ b)) \to a\ (A\ (D\ (D\ (D\ b))))\ (D\ (D\ b)\ c)$

$D\ (D\ b)\ c \to D\ b\ (D\ b\ c) \to b\ (b\ (D\ b\ c))$

Types

Ranked alphabet: (rank = number of children)

a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$S \to A\ b$$
$$A\ f \to a\ (A\ (D\ f))\ (f\ c)$$
$$D\ f\ x \to f\ (f\ x)$$

Every nonterminal (every argument) has assigned some type, for example:

- o – a tree
- $o \to o$ – a function that takes a tree, and produces a tree
- $o \to (o \to o) \to o$ – a function that takes a tree and a function of type $o \to o$, and produces a tree

Order of a type

$$\mathrm{ord}(o) = 0$$
$$\mathrm{ord}(\alpha_1 \to \ldots \to \alpha_k \to o) = 1+\max(\mathrm{ord}(\alpha_1), \ldots, \mathrm{ord}(\alpha_k))$$

For example:

- $\mathrm{ord}(o) = 0$,
- $\mathrm{ord}(o \to o) = \mathrm{ord}(o \to o \to o) = 1$,
- $\mathrm{ord}(o \to (o \to o) \to o) = 2$

Order of a recursion scheme
= maximal order of (a type of) its nonterminal

Model-checking for recursion schemes

General goal: verifying properties of trees generated by schemes

Why? Recursion schemes are decidable models (abstractions) of programs using higher-order recursion

Model-checking for recursion schemes

Input: alternating tree automaton (ATA) $\mathcal{A}$, recursion scheme $\mathcal{G}$

Question: does $\mathcal{A}$ accept the tree generated by $\mathcal{G}$?

Theorem [Ong 2006]

This problem is decidable for parity ATA (i.e., for MSO).

Model-checking for recursion schemes

Input: alternating tree automaton (ATA) $\mathcal{A}$, recursion scheme $\mathcal{G}$
Question: does $\mathcal{A}$ accept the tree generated by $\mathcal{G}$?

Theorem [Ong 2006]
This problem is decidable for parity ATA (i.e., for MSO).

Several proofs, using:
- game semantics
- collapsible pushdown automata
- intersection types
- Krivine machines

and several extensions.
Some proofs only for reachability ATA.

We show another, very simple algorithm for reachability ATA.

Model-checking for recursion schemes

Input: alternating tree automaton (ATA) $\mathcal{A}$, recursion scheme $\mathcal{G}$
Question: does $\mathcal{A}$ accept the tree generated by $\mathcal{G}$?

Theorem [Ong 2006]
This problem is decidable for parity ATA (i.e., for MSO).

Complexity:
- n-EXPTIME-complete for recursion schemes of order n,
- FTP: linear in the size of $\mathcal{G}$, when size of $\mathcal{A}$ and maximal arity of types in $\mathcal{G}$ are fixed,
- the same for parity ATA and for reachability ATA
- (algorithms based on intersection types perform relatively well in practice)

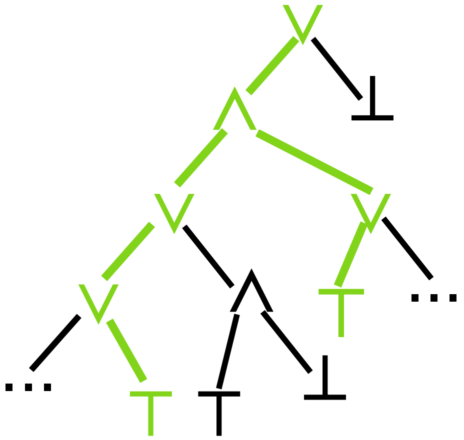
Our algorithm achieves the same complexity.

Preprocessing

We consider an (appropriately defined) product of $\mathcal{G}$ and $\mathcal{A}$.

It is a recursion scheme generating a tree labeled by:
$\wedge$ (AND),
$\vee$ (OR),
with $\top$ (empty AND), $\perp$ (empty OR) as special cases

We ask about alternating reachability.



General idea

We replace the recursion scheme $\mathcal{G}_n$ of order n by an equivalent recursion scheme $\mathcal{G}_{n-1}$ of order n-1. Size grows exponentially.

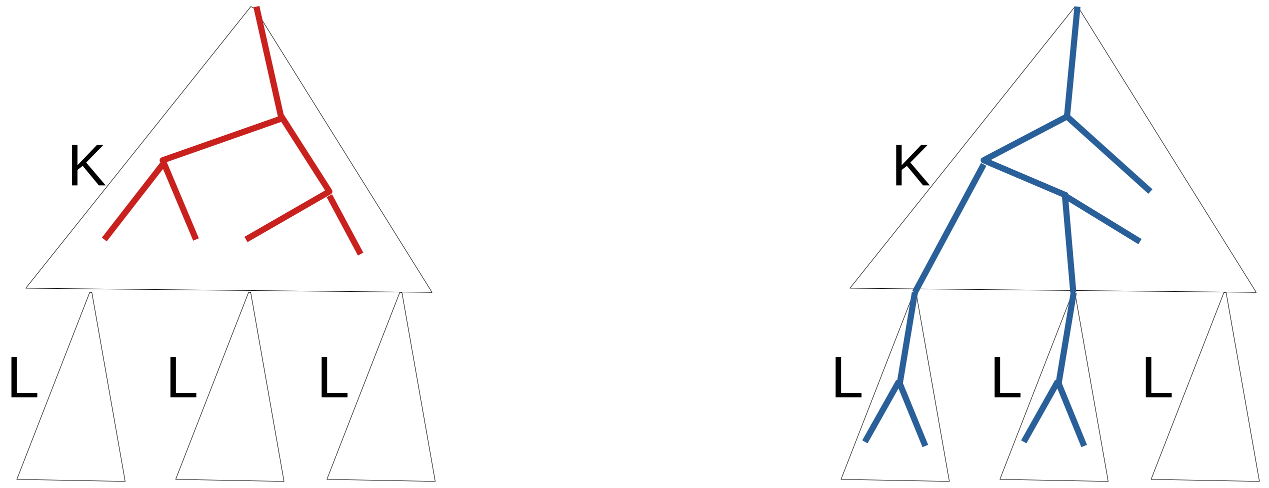
$$\mathcal{G}_n \Rightarrow \mathcal{G}_{n-1} \Rightarrow \mathcal{G}_{n-2} \Rightarrow \cdots \Rightarrow \mathcal{G}_1 \Rightarrow \mathcal{G}_0$$

For recursion schemes of order 0 the problem becomes trivial.

Transformation

Consider an application KL, where L is of order 0 (generates a tree). When is the tree generated by KL accepting?

- When K⊥ is accepting (i.e., K is accepting without using the argument)
- When both KT and L are accepting

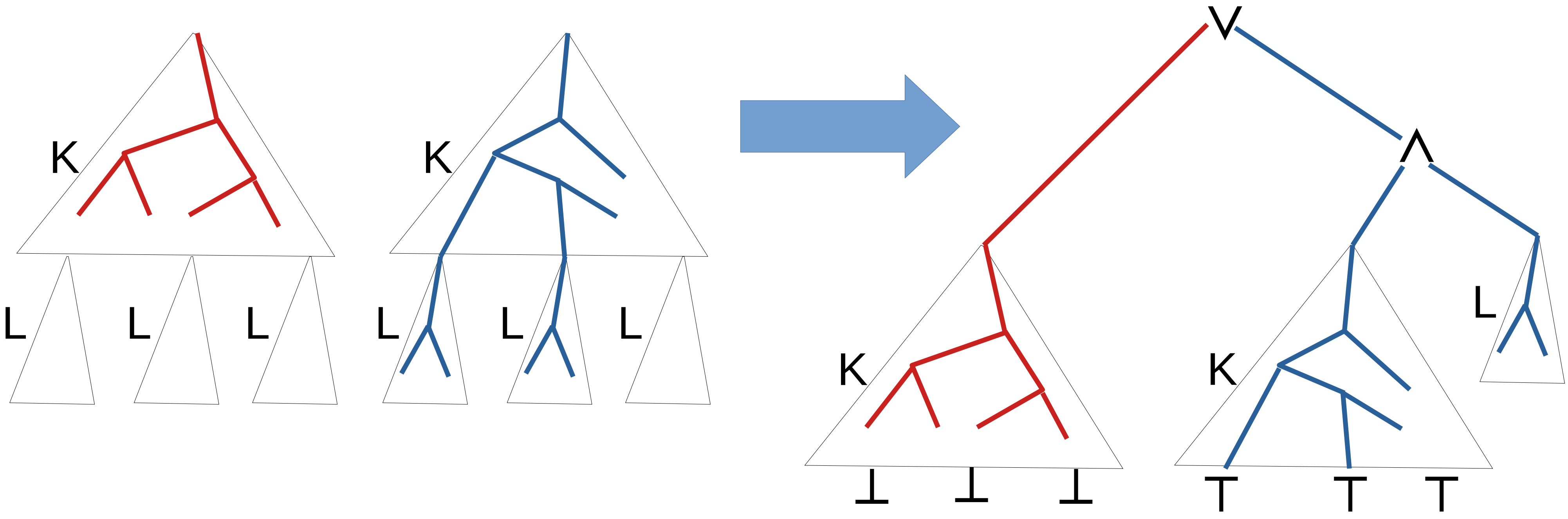


Transformation

Consider an application KL, where L is of order 0 (generates a tree).
When is the tree generated by KL accepting?

- When K⊥ is accepting (i.e., K is accepting without using the argument)
- When both KT and L are accepting

We change KL into ∨ (K⊥) (∧ (KT) L)



Complete example (order 1)

order-0 argument

$$X \rightarrow Y\ Z$$
$$Y\ x \rightarrow \vee\ T\ x$$
$$Z \rightarrow T$$

$$X \rightarrow \vee\ Y_0\ (\wedge\ Y_1\ Z)$$
$$Y_0 \rightarrow \vee\ T\ \bot \qquad Y_1 \rightarrow \vee\ T\ T$$
$$Z \rightarrow T$$

(k order-0 arguments $\Rightarrow$ 2^k variants of the nonterminal)

Complete example (order 2)

$X \rightarrow Z\ Y$

$Y\ x \rightarrow \vee\ T\ x$

$Z\ y \rightarrow y\ (y\ T)$

order-0 arguments

$X \rightarrow Z\ Y_0\ Y_1$

$Y_0 \rightarrow \vee\ T\ \bot \qquad Y_1 \rightarrow \vee\ T\ T$

$Z\ y_0\ y_1 \rightarrow \vee\ y_0\ (\wedge\ y_1\ (\vee\ y_0\ (\wedge\ y_1\ T)))$

Complete example (order 2)

$$X \rightarrow Z\ Y$$
$$Y\ x \rightarrow \vee\ \mathrm{T}\ x$$
$$Z\ y \rightarrow y\ (y\ \mathrm{T})$$

order-0 arguments

$$X \rightarrow Z\ Y_0\ Y_1$$
$$Y_0 \rightarrow \vee\ \mathrm{T}\ \bot \qquad Y_1 \rightarrow \vee\ \mathrm{T}\ \mathrm{T}$$
$$Z\ y_0\, y_1 \rightarrow \vee\ y_0\ (\wedge\ y_1\ (\vee\ y_0\ (\wedge\ y_1\ \mathrm{T})))$$

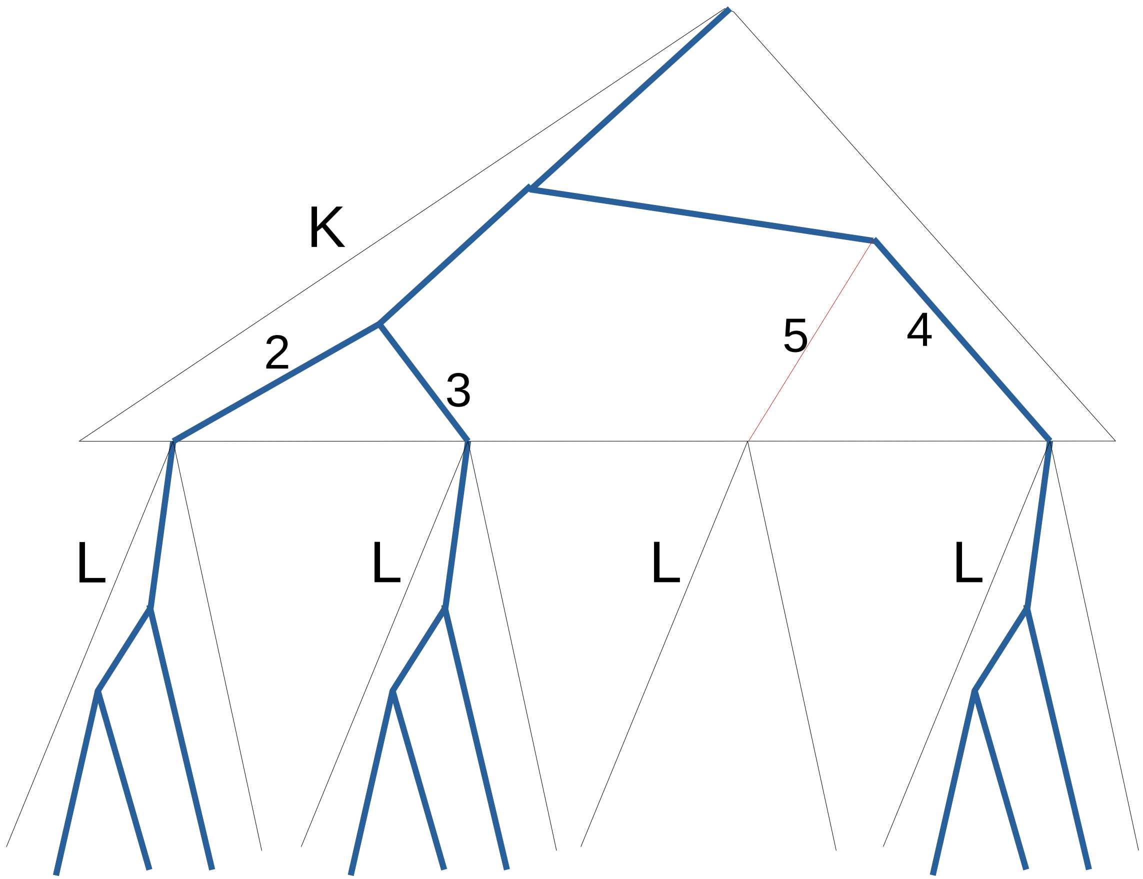
- easy to generalize
- easy (syntactical) correctness proof
- verified in Coq

Similar transformation works for parity automata

Consider an application KL, where L is of order 0 (generates a tree).

How can a winning strategy in KL look like?

- the greatest priority seen in K is p or better
  $\ldots \prec 7 \prec 5 \prec 3 \prec 1 \prec 2 \prec 4 \prec 6 \prec 8 \ldots$
- the strategy in every copy of L can be the same

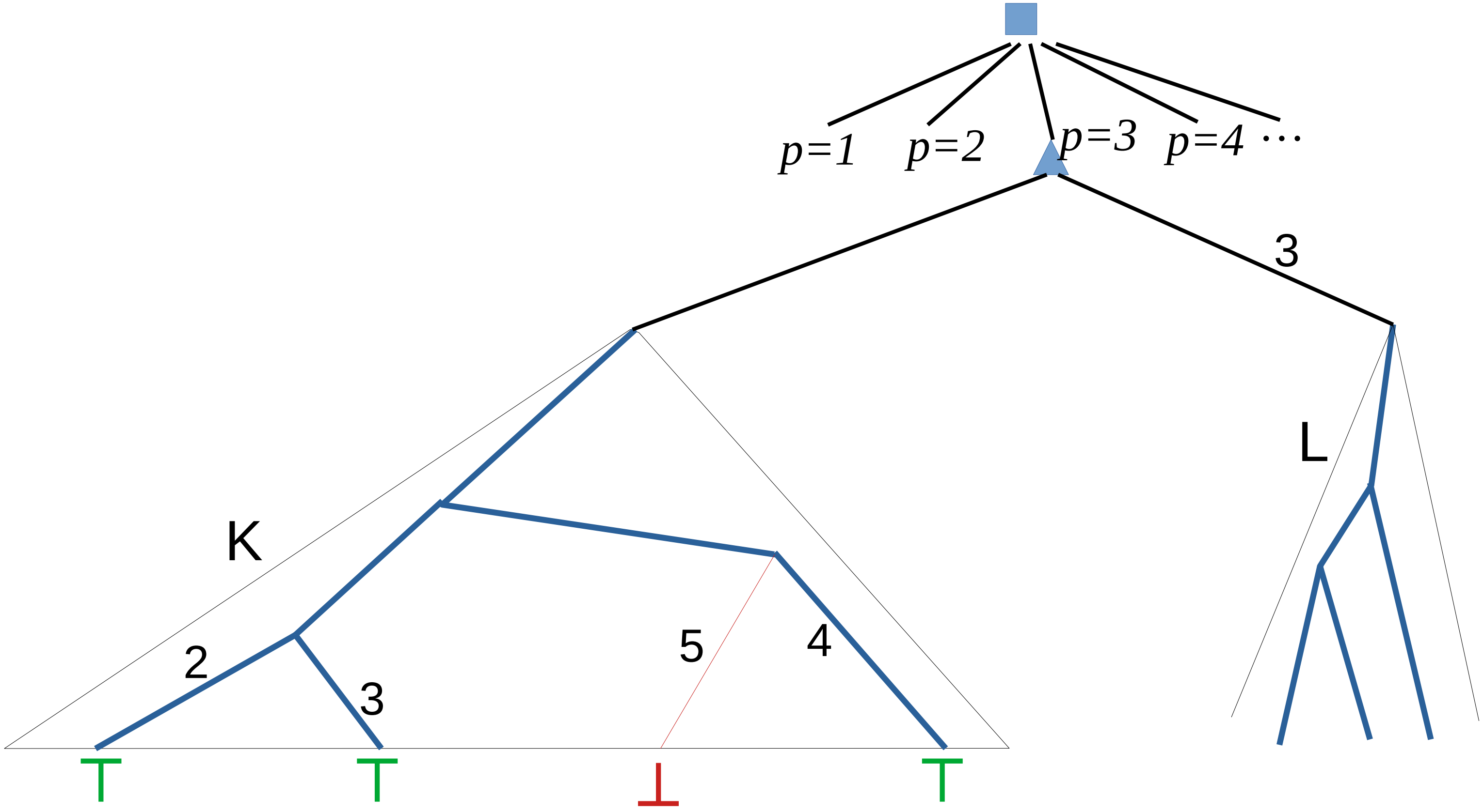


p=3

Similar transformation works for parity automata

After the transformation

- Even declares the priority p for K
- Odd can either check or accept this declaration
- If he checks, we play in K; reaching an argument ends the game
- If he accepts, we read p, and we continue in L



More details:

- Duplicate nonterminals – a copy for every value of p
- Duplicate arguments – a copy for every value of p
- Remove arguments of order 0 $\Rightarrow$ order decreases by 1

Conclusion

- We consider the model-checking problem for recursion schemes + reachability ATA / parity ATA
- We propose a new, simpler algorithm solving this problem: we repeatedly reduce the order of a recursion scheme by one, increasing its size exponentially
- We obtain optimal complexity

Thank you!