Higher-Order Model Checking Step by Step

Paweł Parys

University of Warsaw

ICALP 2021

What is it about?

Higher-Order = we consider higher-order recursion schemes

Model Checking = we solve the acceptance problem for alternating parity automata

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:

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

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

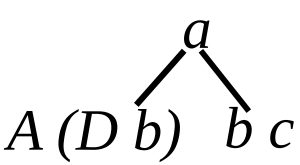
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

S (starting), A, D



Rules:

$S \to A\ b$

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

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

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

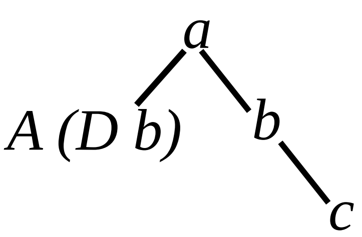
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

S (starting), A, D



Rules:

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

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

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:

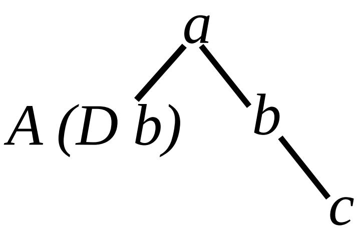
$S \rightarrow A\ b$

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

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

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

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



Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

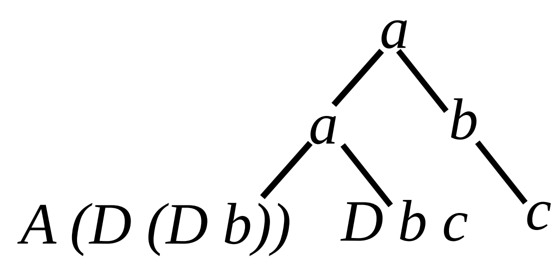
S (starting), A, D

Rules:

$S \rightarrow A\ b$

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

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



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

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

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

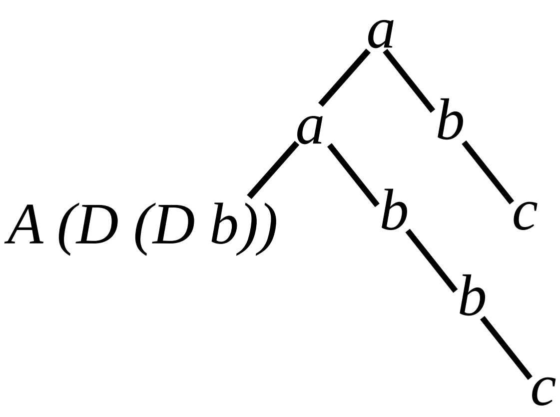
S (starting), A, D

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)

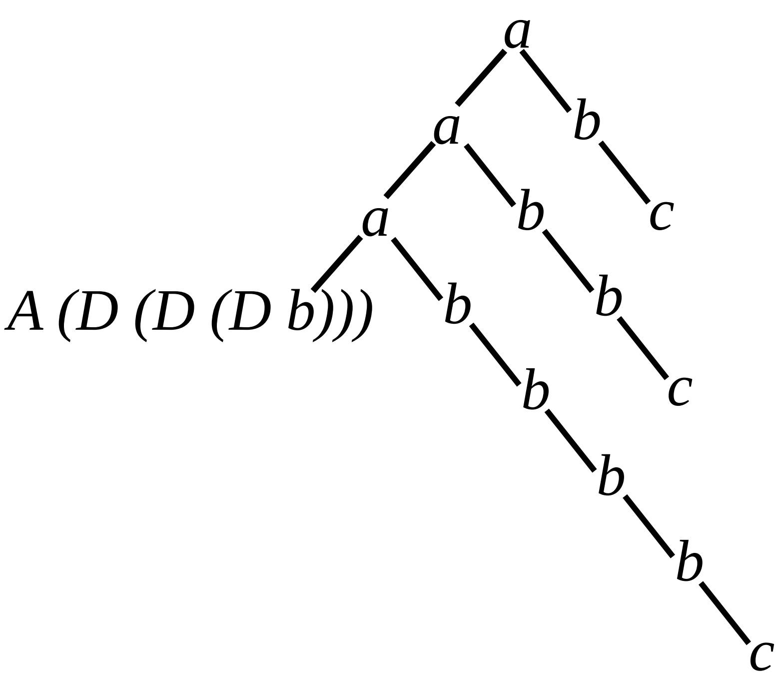
a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

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



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

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

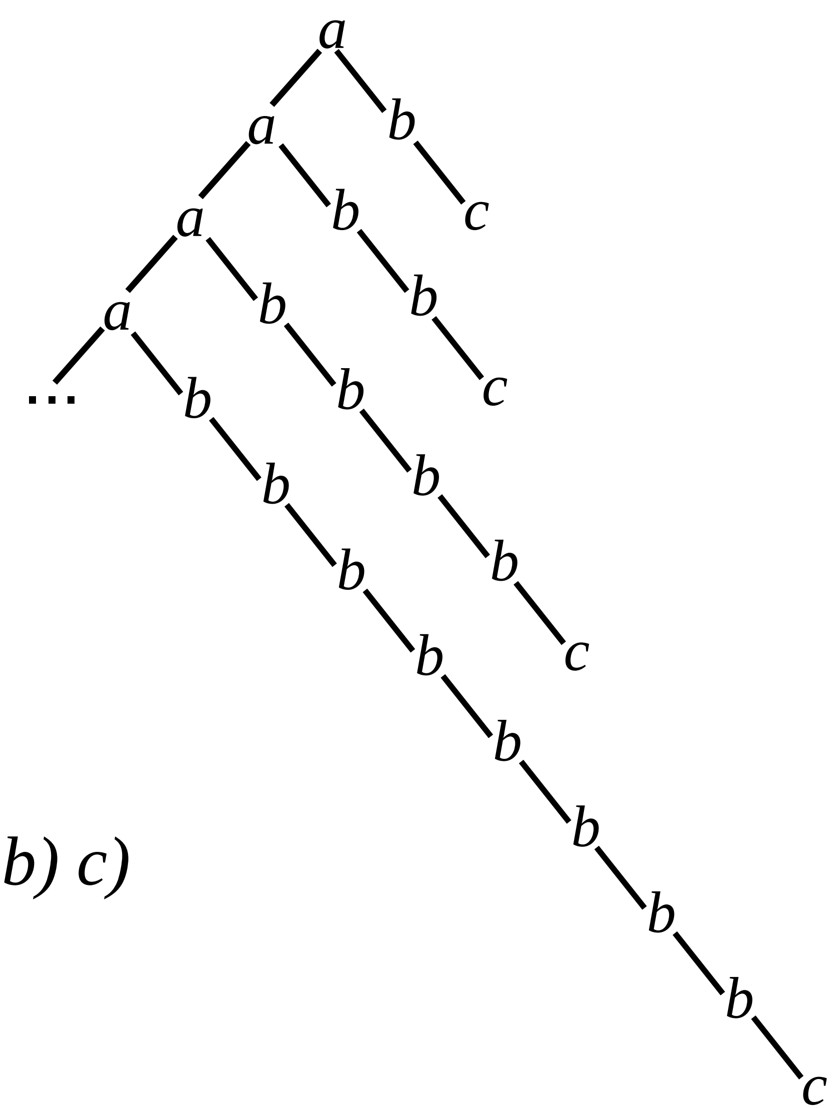
S (starting), A, D

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

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

For example:

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

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) $\mathcal{A}$ with parity condition,
recursion scheme $\mathcal{G}$

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

Theorem [Ong 2006]
This problem is decidable.

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

and several extensions.
Some proofs only for reachability ATA.

We show another, quite simple algorithm.

Model-checking for recursion schemes

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

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

Theorem [Ong 2006]
This problem is decidable.

Complexity:

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

Our algorithm achieves the same complexity.

Preprocessing

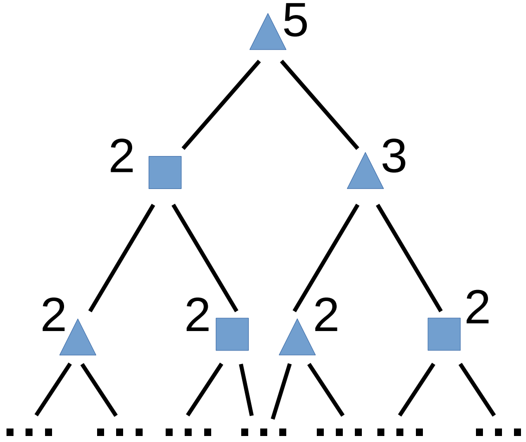
We consider an (appropriately defined) product of $\mathcal{G}$ and $\mathcal{A}$.

It generates a tree of "runs of $\mathcal{A}$ on $\mathcal{G}$" with nodes labeled by:

- player name,
- priority.

This tree is thus an infinite parity game.

We ask who wins this game.



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.

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

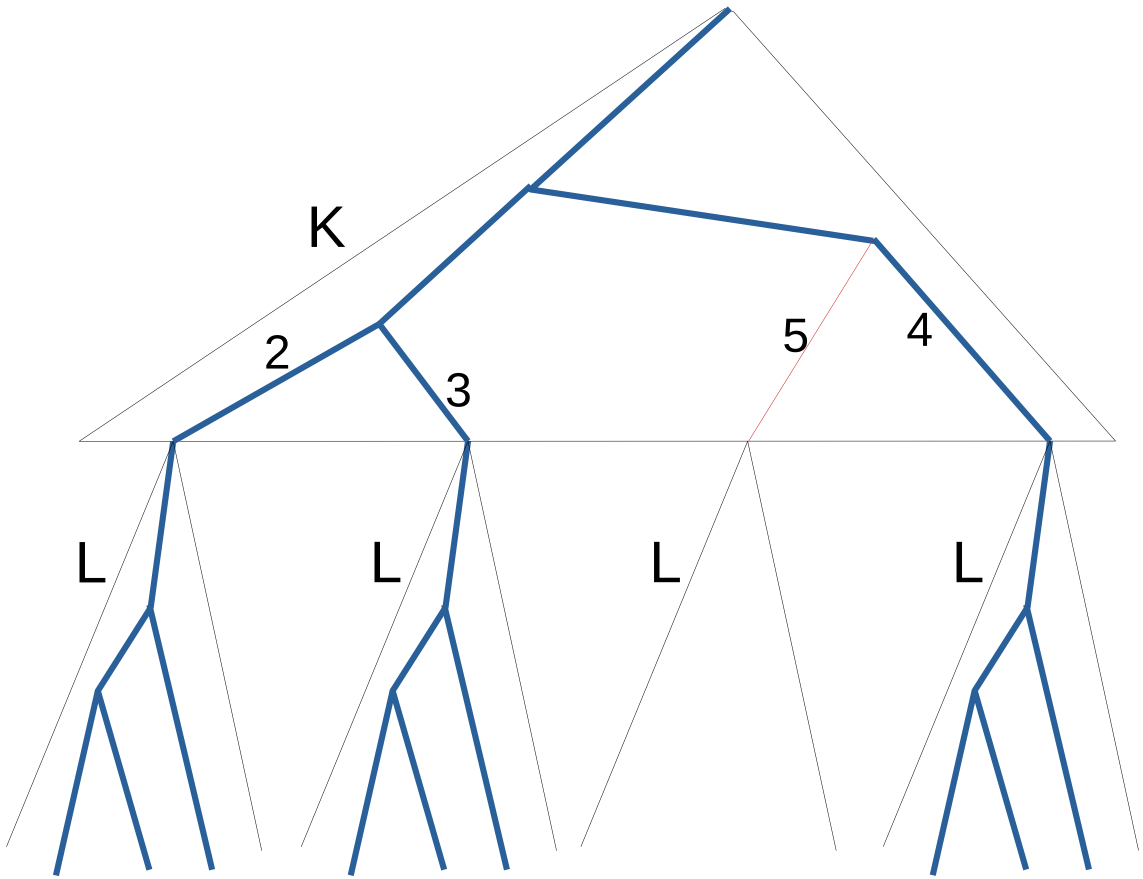
For recursion schemes of order 0 the problem becomes trivial.

Transformation

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

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

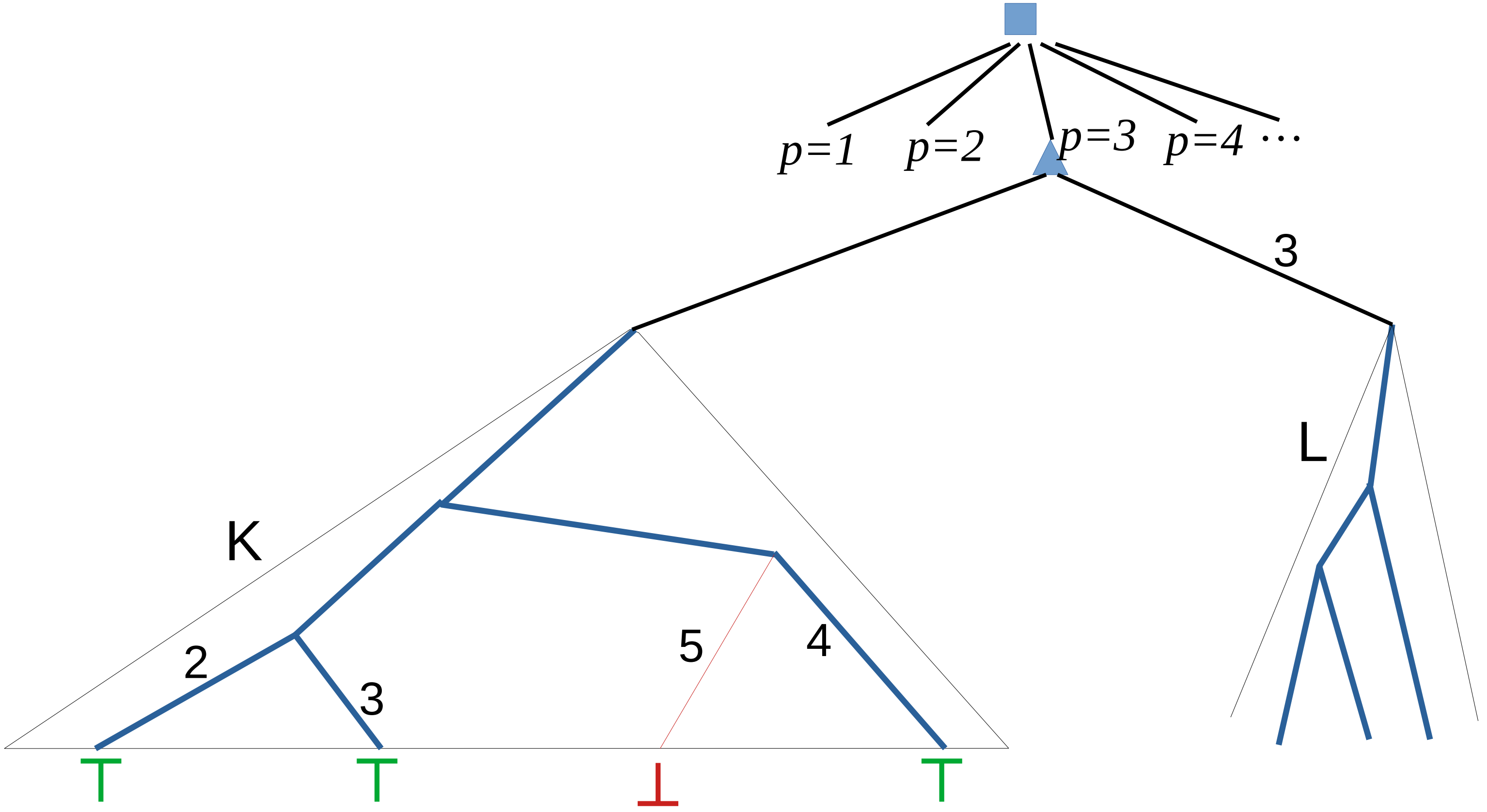


$p=3$

Transformation

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 + parity ATA
- We propose a new, simpler method 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!