Unboundedness for Recursion Schemes: A Simpler Type System

David Barozzini **Paweł Parys** Jan Wróblewski

University of Warsaw

ICALP 2022

What is it about?

Recursion schemes = we consider trees generated by higher-order recursion schemes

Unboundedness = we provide an algorithm checking whether some properties in these trees are unbounded

Simpler type system = we give a new, simpler method, leading to a practical algorithm

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:

$$
\begin{aligned}
S &\to A\ b \\
A\ f &\to a\ (A\ (D\ f))\ (f\ c) \\
D\ f\ x &\to f\ (f\ x)
\end{aligned}
$$

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:

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

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

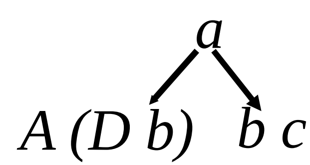
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

S (starting), A, D



Rules:

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

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

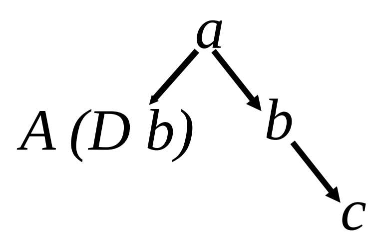
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

S (starting), A, D



Rules:

$$
\begin{aligned}
&S &&\to A\ b\\
&A\ f &&\to a\ (A\ (D\ f))\ (f\ c)\\
&D\ f\ x &&\to f\ (f\ x)
\end{aligned}
$$

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

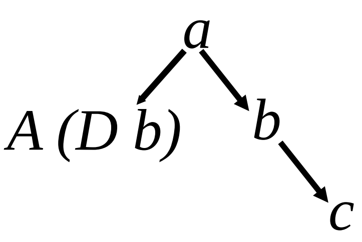
Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

S (starting), A, D



Rules:

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

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

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

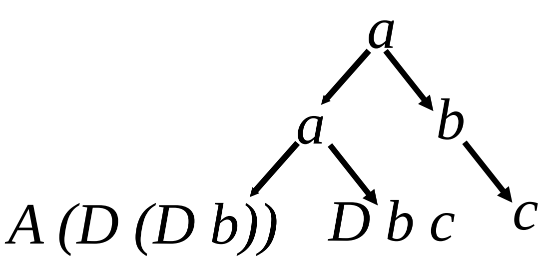
a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

$$
\begin{aligned}
S &\to A\ b \\
A\ f &\to a\ (A\ (D\ f))\ (f\ c) \\
D\ f\ x &\to f\ (f\ x)
\end{aligned}
$$



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

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

Higher-order recursion schemes – example

Ranked alphabet: (rank = number of children)

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

Nonterminals:

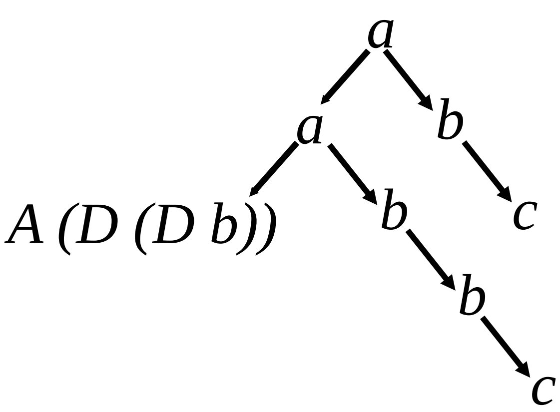
S (starting), A, D

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)

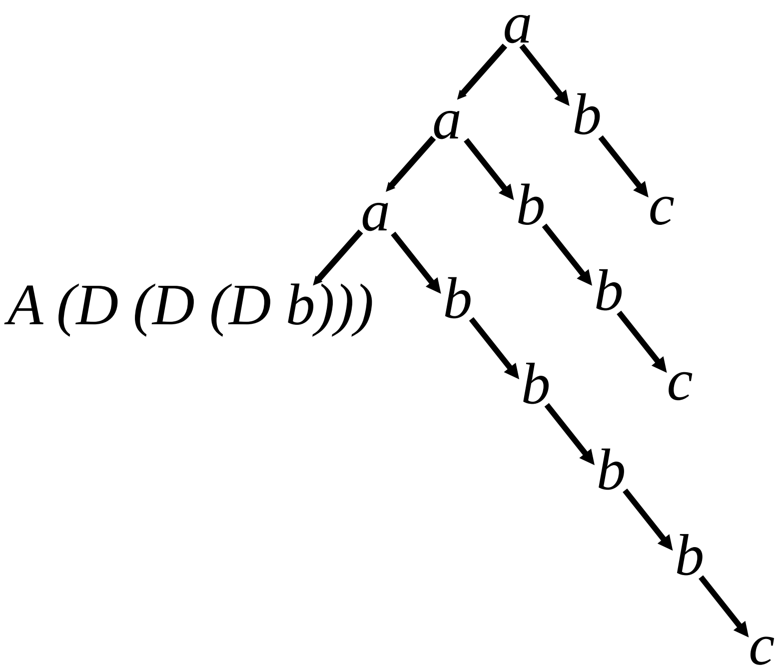
a of rank 2, b of rank 1, c of rank 0

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)

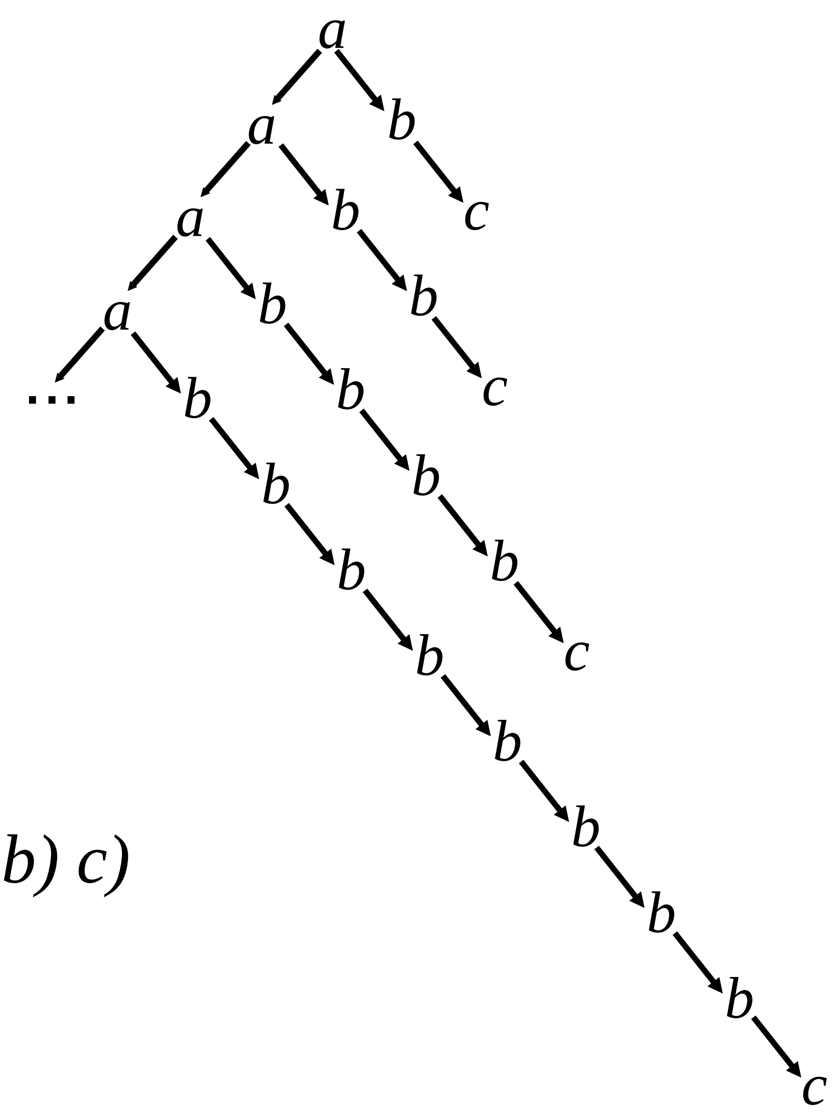
a of rank 2, b of rank 1, c of rank 0

Nonterminals:

S (starting), A, D

Rules:

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



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

Sorts (simple 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:

$$
\begin{aligned}
&S &&\rightarrow A\, b\\
&A\, f &&\rightarrow a\, (A\, (D\, f))\, (f\, c)\\
&D\, f\, x &&\rightarrow f\, (f\, x)
\end{aligned}
$$

Every nonterminal (every argument) has assigned some “sort”, for example:

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

Model-checking

Theorem [Ong 2006]
MSO model-checking on trees generated by recursion schemes is decidable.

Input: recursion scheme $\mathcal{G}$, MSO formula ϕ
Question: is ϕ true in the (infinite) tree generated by $\mathcal{G}$?

Model-checking

Theorem [Ong 2006]
MSO model-checking on trees generated by recursion schemes is decidable.

Input: recursion scheme $\mathcal{G}$, MSO formula ϕ
Question: is ϕ true in the (infinite) tree generated by $\mathcal{G}$?

This procedure can be used for model-checking programs written in functional programming languages:

Input: a program P, a property ψ
Question: does P satisfy ψ?

CEGAR loop, etc.
There exist tools that take (short) programs in Ocaml and can verify some useful properties.

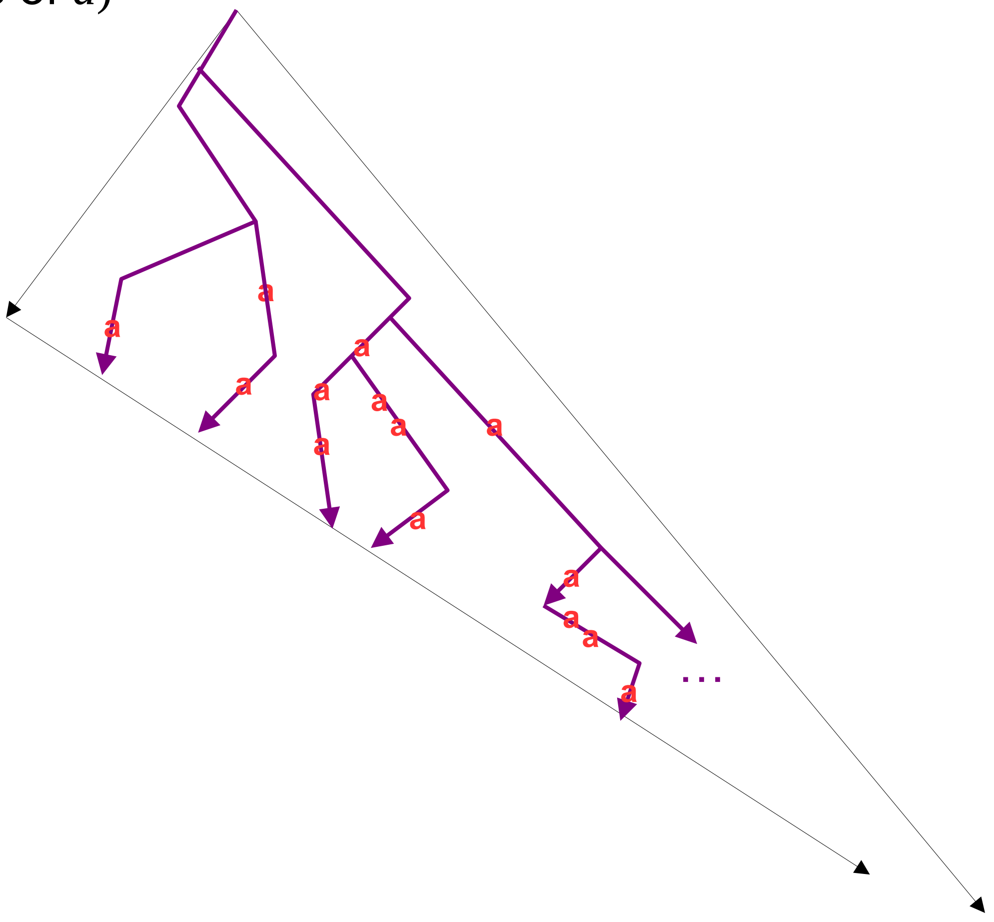
Several recent papers – can we go beyond MSO?

What about checking properties not expressible in MSO, e.g., talking about boundedness?

Unboundedness – basic problem

Input: recursion scheme G
Question: In the tree generated by G, are there (finite) branches with arbitrarily many occurrences of a symbol "*a*"?

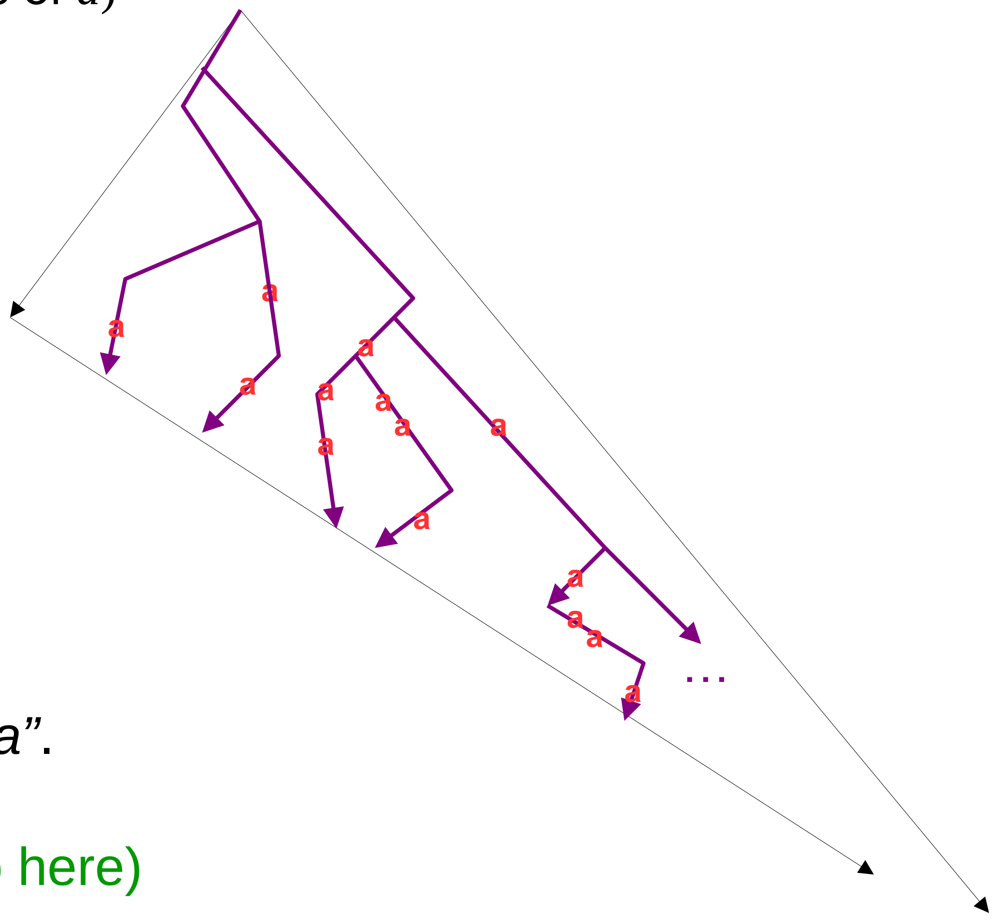
($\forall n$ $\exists$branch with $>n$ occurrences of a)



Unboundedness – basic problem

Input: recursion scheme G
Question: In the tree generated by G, are there (finite) branches with arbitrarily many occurrences of a symbol "*a*"?

($\forall n$ $\exists$branch with $>n$ occurrences of a)

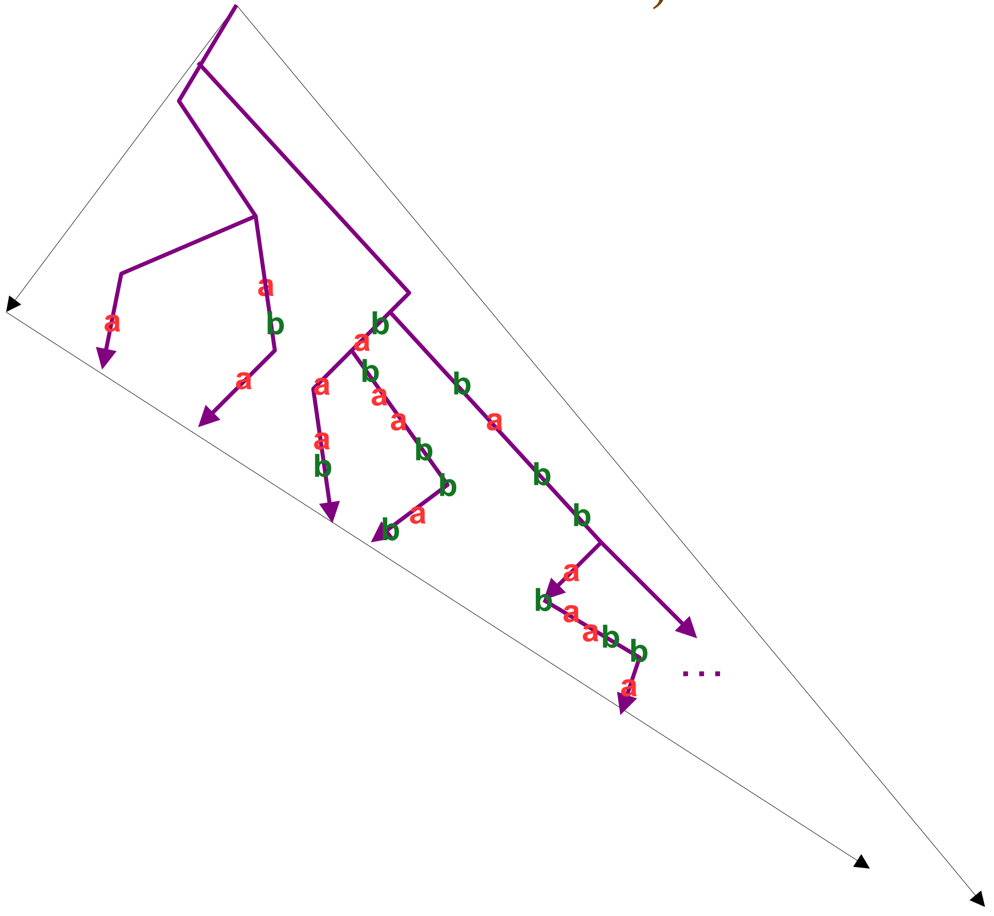


Notice:
There may be no path with infinitely many „*a*".
Our property **is not regular!!!**
(the result [Ong – LICS 2006] does not help here)

Simultaneous unboundedness

Input: recursion scheme G, set of symbols A
Question: In the tree generated by G, are there (finite) branches with arbitrarily many occurrences of every symbol from A?

($\forall n\ \exists$branch $\forall a \in A$ there are $>n$ occurrences of a on the branch)



Known results

Given a recursion scheme $\mathcal{G}$ generating a tree $\mathcal{T}$, the following problems are decidable:

- Does $\mathcal{T}$ satisfy $\phi \in$MSO? [Ong 2006]
  (equivalently: is $\mathcal{T}$ accepted by a parity automaton)?
- Simultaneous unboundedness for $\mathcal{T}$. [Clemente, P., Salvati, Walukiewicz 2016]
- Does $\mathcal{T}$ satisfy $\phi \in$WMSO+U? [P. 2018]
- Only if $\mathcal{G}$ is safe: Is $\mathcal{T}$ accepted by a B-automaton $\mathcal{A}$?
  [Barozzini, Clemente, Colcombet, P. 2020]

Known results

Given a recursion scheme $\mathcal{G}$ generating a tree $\mathcal{T}$, the following problems are decidable:

- Does $\mathcal{T}$ satisfy $\phi \in$MSO? [Ong 2006]
  (equivalently: is $\mathcal{T}$ accepted by a parity automaton)?
- Simultaneous unboundedness for $\mathcal{T}$. [Clemente, P., Salvati, Walukiewicz 2016]
- Does $\mathcal{T}$ satisfy $\phi \in$WMSO+U? [P. 2018]
- Only if $\mathcal{G}$ is safe: Is $\mathcal{T}$ accepted by a B-automaton $\mathcal{A}$?
  [Barozzini, Clemente, Colcombet, P. 2020]

- The problem is n-EXP complete for schemes of order n
- There exist tools solving this problem in practice:
  - ➔ TRecS, HorSat, … [Kobayashi, Broadbent, ...]
  - ➔ HORSC, TravMC2 [Neatherway, Ramsay, Ong, ...]
- The tools are based on intersection type systems

Known results

Given a recursion scheme $\mathcal{G}$ generating a tree $\mathcal{T}$, the following problems are decidable:

- Does $\mathcal{T}$ satisfy $\phi \in$MSO? [Ong 2006]
  (equivalently: is $\mathcal{T}$ accepted by a parity automaton)?
- Simultaneous unboundedness for $\mathcal{T}$. [Clemente, P., Salvati, Walukiewicz 2016]
- Does $\mathcal{T}$ satisfy $\phi \in$WMSO+U? [P. 2018]
- Only if $\mathcal{G}$ is safe: Is $\mathcal{T}$ accepted by a B-automaton $\mathcal{A}$?
  [Barozzini, Clemente, Colcombet, P. 2020]

  - solution for safe schemes [Hague, Kochems, Ong 2016]
  - solution for all schemes [Clemente, P., Salvati, Walukiewicz 2016]
  - can be solved in n-EXP for schemes of order n [P. 2017]
  - this paper: can be solved in practice (for safe schemes)

Order of a sort

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

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

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\,N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\,N_1 \ldots N_k$$

only if $ord(M\,N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\ N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\ N_1 \ldots N_k$$

only if $ord(M\ N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check safety for our example HORS:

$$\begin{aligned} S &\rightarrow A\ b \\ A\ f &\rightarrow a\ (A\ (D\ f))\ (f\ c) \\ D\ f\ x &\rightarrow f\ (f\ x) \end{aligned}$$

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\ N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\ N_1 \ldots N_k$$

only if $ord(M\ N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check safety for our example HORS:

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

$ord(D\ f) = 1 \leq 1 = ord(f)$ $\to$ OK

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\ N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\ N_1 \ldots N_k$$

only if $ord(M\ N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Let's check safety for our example HORS:

$$\begin{array}{lll} S & \to & A\ b \\ A\ f & \to & a\ (A\ (D\ f))\ (f\ c) \\ D\ f\ x & \to & f\ (f\ x) \end{array}$$

✓ safe

$ord(D\ f) = 1 \leq 1 = ord(f)$ → OK

All other subterms are of order 0 → OK

What is safety?

Restriction on terms appearing on right sides of rules:

- unrestricted terms:

$$M ::= a \mid x \mid A \mid M\, N$$

- safe terms:

$$M ::= a \mid x \mid A \mid M\, N_1 \ldots N_k$$

only if $ord(M\, N_1 \ldots N_k) \leq ord(N_i)$ for all i

In other words: if we apply an argument of some order k, then we have to apply also all arguments of order $\geq k$

Example: Unsafe HORS (generating ”Urzyczyn's tree” U):

Types: $a^{o\to o\to o}$, $b^{o\to o}$, $c^{o\to o}$, d^{o}, e^{o}, S^{o}, $F^{(o\to o)\to o\to o\to o}$

Rules: $S \to F\, b\, d\, e$

$$F\, f\, x\, y \to a\, (F\, (F\, f\, x)\, y\, (c\, y))\, (a\, (f\, y)\, x)$$

✘ unsafe

(and not equivalent to any safe HORS)

$$ord(F\, f\, x) = 1 > 0 = ord(x)$$

(F expects two order-0 arguments; we have applied one (x), but not the other)

Why safety helps?

Theorem [Knapik, Niwiński, Urzyczyn 2002; Blum, Ong 2007]
Substitution (hence β-reduction) in safe λ-calculus can be implemented without renaming bound variables.

Bad example: when you substitute $(\lambda x.y\ x)\ [a\ x\ x\ /\ y]$, it is necessary to change the first two x to some other variable name

Our solution to the unboundedness problem

We provide a system of intersection types.

A type derivation says:

- which arguments / free variables are used (and with which type)
- if the term is „productive”:
  - produces the letter „a”, or
  - used a productive argument twice

“Productive” places in a type derivation can be counted.

Our solution to the unboundedness problem

We provide a system of intersection types.
A type derivation says:

- which arguments / free variables are used (and with which type)
- if the term is „productive”:
  - produces the letter „a”, or
  - used a productive argument twice

“Productive” places in a type derivation can be counted.

Theorem

G has type derivations with arbitrarily many productive places $\Longleftrightarrow$ the tree generated by G has branches with arbitrarily many symbols „a”

$\Longrightarrow$ soundness - always

$\Longleftarrow$ completeness – proof only for safe schemes
– ??? for other schemes

Our solution to the unboundedness problem

Algorithm & implementation:

- based on HORSAT2
- tries to find all possible type derivations
- found a derivation with a “productive loop” → answer YES
- optimizations are necessary – mostly coming from HORSAT2 (which types for subterms may be useful)

Our solution to the unboundedness problem

Algorithm & implementation:

- based on HORSAT2
- tries to find all possible type derivations
- found a derivation with a “productive loop” → answer YES
- optimizations are necessary – mostly coming from HORSAT2 (which types for subterms may be useful)

Evaluation:

- tried on (adapted) benchmarks from HORSAT, coming from real verification problems + some new examples
- 24 inputs, only 2 timeouts (> 600s)
- on other inputs works in <60s, often <1s
- size (largest solved): 400 rules, order 8

Conclusion

- We consider the unboundedness problem for recursion schemes
- We propose a new, simpler type system for this problem
- Correctness proof for safe schemes
- Open question: does the type system work for all schemes?
- We implemented a tool working relatively well in practice

Thank you!