Complexity of the Diagonal Problem for Recursion Schemes

Paweł Parys

University of Warsaw

FSTTCS, 13.12.2017

Higher-order recursion schemes – what is this?

Definition

Recursion schemes = simply-typed lambda-calculus + recursion

or: a context-free grammar in which nonterminals can take arguments

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

We consider here nondeterministic schemes.

Higher-order recursion schemes – what is this?

Definition

Recursion schemes = simply-typed lambda-calculus + recursion

or: a context-free grammar in which nonterminals can take arguments

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

We consider here nondeterministic schemes.

actual objects		abstraction
sequential programs	⟹	finite-state systems
programs with H-O recursion	⟹	H-O recursion schemes

Higher-order recursion schemes – example

Rules:

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

Here: S,A,D are nonterminals (where S – starting nonterminal),
a,b are terminals,
f,x are variables (parameters)

Higher-order recursion schemes – example

Rules:

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

Here: S,A,D are nonterminals (where S – starting nonterminal),
a,b are terminals,
f,x are variables (parameters)

The same using another syntax:

fun S() = { A(a); }
fun A(f) = { if ? then A(λx.D(f,x)) else f(b); }
fun D(f,x) = { f(f(x)); }

Higher-order recursion schemes – example

Rules:

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

Here: S,A,D are nonterminals (where S – starting nonterminal),
a,b are terminals,
f,x are variables (parameters)

In our world, everything is required to have a fixed type:

$S : o, \quad A : (o \to o) \to o, \quad D : (o \to o) \to o \to o$

$a : o \to o, \quad b : o$

$f : o \to o, \quad x : o$

Higher-order recursion schemes – example

Rules:

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

Dynamics:

$S \to A\ a$

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Basic setting:

- Terminals are either of type $o \to o$ (one argument of ground type), or of type o (no arguments)
  Then a scheme generates a word (e.g., $a\ a\ a\ a\ b$)

Higher-order recursion schemes – example

Rules:

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

Dynamics:

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

Basic setting:

- Terminals are either of type $o \to o$ (one argument of ground type), or of type o (no arguments)
  Then a scheme generates a word (e.g., $a\ a\ a\ a\ b$)

Recognized language: $\{a^{2^n}b : n \in \mathbb{N}\}$

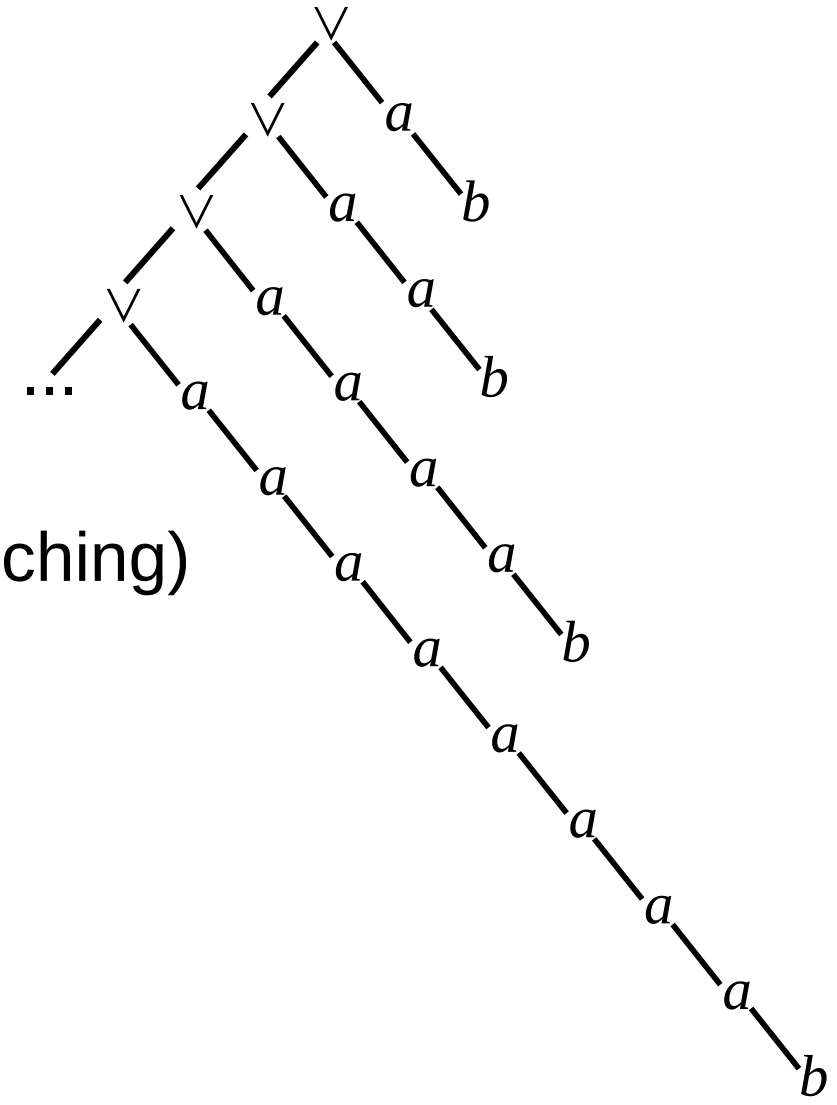
Higher-order recursion schemes – example

Rules:

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

Recognized language: $\{a^{2^n}b : n \in \mathbb{N}\}$

Another view: generating a single infinite tree, containing all possible derivations of the scheme ("$\vee$" = branching)



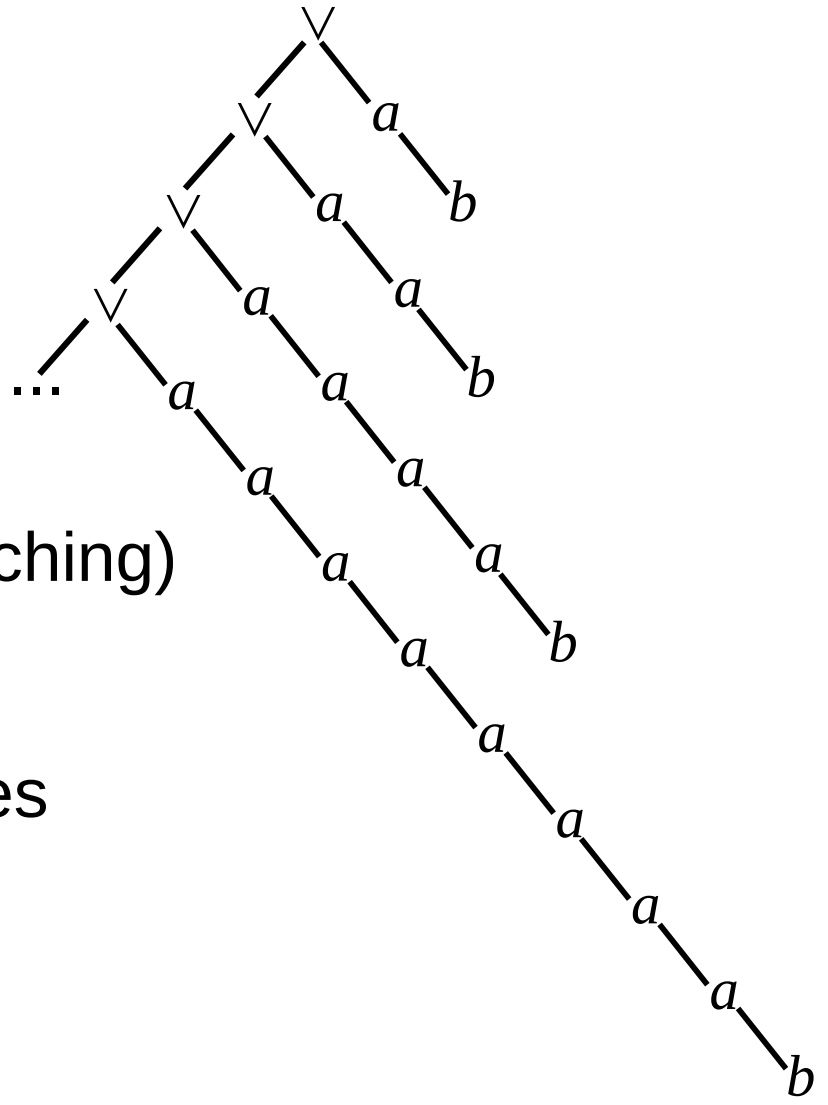
Higher-order recursion schemes – example

Rules:

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

Recognized language: $\{a^{2^n}b : n \in \mathbb{N}\}$

Another view: generating a single infinite tree, containing all possible derivations of the scheme ("$\vee$" = branching)



General goal: verifying properties of languages/trees generated by schemes

Model checking for schemes

General goal: verifying properties of languages/trees generated by schemes

Theorem [Ong 2006]
There is an algorithm, which given a recursion scheme G and an MSO formula ϕ checks whether ϕ holds in the tree generated by G.

Model checking for schemes

General goal: verifying properties of languages/trees generated by schemes

Theorem [Ong 2006]
There is an algorithm, which given a recursion scheme G and an MSO formula ϕ checks whether ϕ holds in the tree generated by G.

Moreover: there exist (experimental) tools that take as input a program in a functional language, and a property, and check whether the program satisfies this property. They approximate the program by a recursion scheme, and then they check the property in the tree generated by the scheme.

Model checking for schemes

General goal: verifying properties of languages/trees generated by schemes

Theorem [Ong 2006]
There is an algorithm, which given a recursion scheme G and an MSO formula ϕ checks whether ϕ holds in the tree generated by G.

Moreover: there exist (experimental) tools that take as input a program in a functional language, and a property, and check whether the program satisfies this property. They approximate the program by a recursion scheme, and then they check the property in the tree generated by the scheme.

Aim of this work: verify properties not expressible in MSO.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Example:

$A = \{a, b\}$

$L_1 = \{a^n b \mid n \in \mathbb{N}\} \cup \{ab^n \mid n \in \mathbb{N}\}$ – answer = NO

$L_2 = a^* b^*$ – answer = YES

$L_3 = \{a^n b^{n+3} \mid n \in \mathbb{N}\}$ – answer = YES

$L_4 = \{bbb(ab)^n \mid n \in \mathbb{N}\}$ – answer = YES

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Remark

Notice that this is not a regular property of the tree generated by S, i.e., the tree containing all words from L on particular branches.

We say here that “something is unbounded”.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Motivation?

1. Interesting on its own: sometimes we want to know whether some event may happen arbitrarily many times, or not.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Motivation?

1. Interesting on its own: sometimes we want to know whether some event may happen arbitrarily many times, or not.
2. It is easy to reduce the problem of computing the downward closure of the language recognized by a scheme to the diagonal problem
   The downward closure $L{\downarrow}$ of a language L contains all words that can be obtained from words in L by removing some letters [Zetzsche 2015].
   The downward closure is always a regular language.

examples:

$(a^*b^*){\downarrow} = a^*b^*$

$\{a^n b^{n+3} \mid n \in \mathbb{N}\}{\downarrow} = a^*b^*$

$\{bbb(ab)^n \mid n \in \mathbb{N}\}{\downarrow} = \{a,b\}^*$

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Motivation?

1. Interesting on its own: sometimes we want to know whether some event may happen arbitrarily many times, or not.
2. It is easy to reduce the problem of computing the downward closure of the language recognized by a scheme to the diagonal problem
   The downward closure $L{\downarrow}$ of a language L contains all words that can be obtained from words in L by removing some letters [Zetzsche 2015].

2'. For languages recognized by schemes, it is undecidable whether $L = A^*$, $L_1 = L_2$, etc. But we can check this approximately, by checking whether $L{\downarrow} = A^*$, $L_1{\downarrow} = L_2{\downarrow}$, etc.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Motivation?

1. Interesting on its own: sometimes we want to know whether some event may happen arbitrarily many times, or not.
2. It is easy to reduce the problem of computing the downward closure of the language recognized by a scheme to the diagonal problem
   The downward closure $L{\downarrow}$ of a language L contains all words that can be obtained from words in L by removing some letters [Zetzsche 2015].

2'. For languages recognized by schemes, it is undecidable whether $L = A^*$, $L_1 = L_2$, etc. But we can check this approximately, by checking whether $L{\downarrow} = A^*$, $L_1{\downarrow} = L_2{\downarrow}$, etc.

3. The problem "is there a piecewise testable language (i.e., boolean combination of downward closed languages) containing L_1 and not intersecting with L_2" reduces to the diagonal problem [Czerwiński, Martens, van Rooijen, Zeitoun 2015]. This gives a more refined approximation for disjointness of L_1 and L_2 than the test $L_1{\downarrow} \cap L_2{\downarrow} = \varnothing$.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

Theorem [Clemente, P., Walukiewicz, Salvati 2016]
The diagonal problem for recursion schemes is decidable.

This paper: further improvements

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch $\xrightarrow{\text{step 1}}$ a scheme S of order $m\text{-}1$ with this word written in leaves

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

Order of a type:

$ord(o) = 0$

$ord(\alpha_1 \to ... \to \alpha_k \to o) = 1+\max(ord(\alpha_1), \ldots, ord(\alpha_k))$

For example:

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

Order of a HORS = maximal order of (a type of) its nonterminal

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

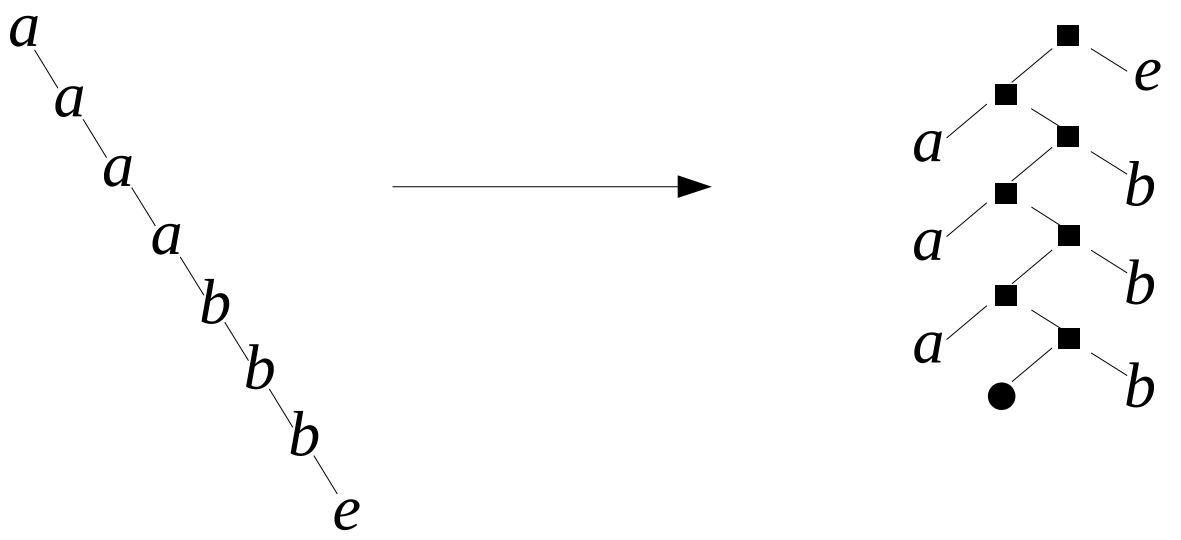
How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

Example:

$S \to A\ e$
$A\ x \to a\ (A\ (b\ x))$
$A\ x \to x$

$S \to \blacksquare\ A\ e$
$A \to \blacksquare\ a\ (\blacksquare\ A\ b))$
$A \to \bullet$



We have generalized the previous setting: our new scheme recognizes a language of trees, not a language of words

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order $m\text{-}1$ with this word written in leaves

Example:

$$S \to A\ e$$
$$A\ x \to a\ (A\ (b\ x))$$
$$A\ x \to x$$

$\longrightarrow$

$$S \to \blacksquare\ A\ e$$
$$A \to \blacksquare\ a\ (\blacksquare\ A\ b))$$
$$A \to \bullet$$

Idea: 1) Observe that an argument of type o can be used at most once.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

Example:

$$\begin{array}{l} S \to A\ e \\ A\ x \to a\ (A\ (b\ x)) \\ A\ x \to x \end{array} \longrightarrow \begin{array}{l} S \to \blacksquare\ A\ e \\ A \to \blacksquare\ a\ (\blacksquare\ A\ b)) \\ A \to \bullet \end{array}$$

Idea: 1) Observe that an argument of type o can be used at most once.

2) All arguments of type o are dropped ($\Rightarrow$ order decreases).

3) Every subterm $M\ N$ with N of type o can be replaced

a) either by $\blacksquare\ M\ N$ (when the argument is used in M),

b) or by M (when the argument is ignored in M).

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

Example:

$$\begin{array}{l} S \to A\ e \\ A\ x \to a\ (A\ (b\ x)) \\ A\ x \to x \end{array} \longrightarrow \begin{array}{l} S \to \blacksquare\ A\ e \\ A \to \blacksquare\ a\ (\blacksquare\ A\ b)) \\ A \to \bullet \end{array}$$

Idea: 1) Observe that an argument of type o can be used at most once.

2) All arguments of type o are dropped ($\Rightarrow$ order decreases).

3) Every subterm $M\ N$ with N of type o can be replaced

a) either by $\blacksquare\ M\ N$ (when the argument is used in M),

b) or by M (when the argument is ignored in M).

4) Additional work is required to choose correctly a) or b).

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

—step 2→ a scheme S of order m-1 with a *similar* word written on a branch

The diagonal problem

Input: a scheme S recognizing L, set of letters A

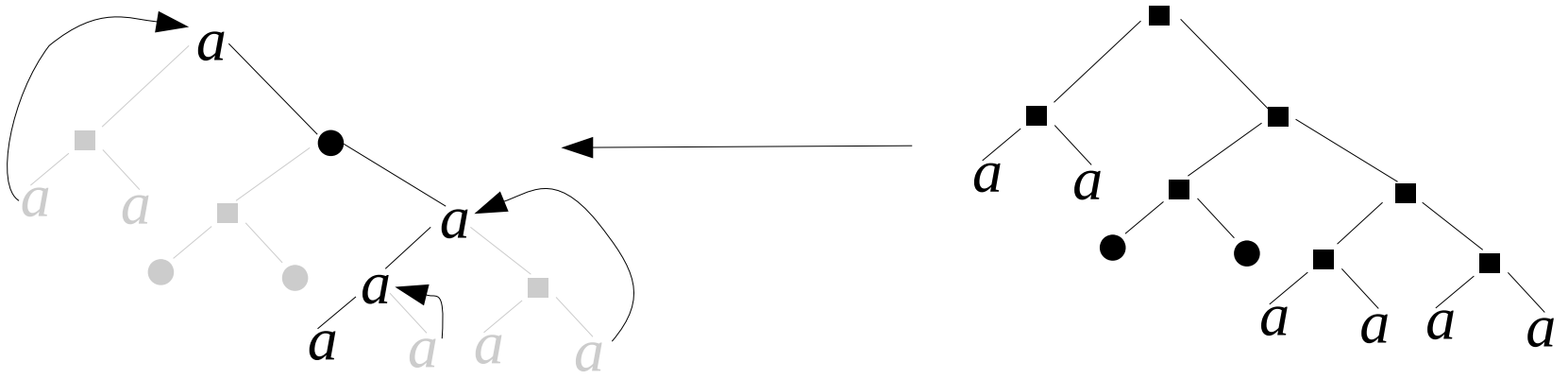
Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

—step 2→ a scheme S of order m-1 with a *similar* word written on a branch

Example:



Idea:

1) Choose (nondeterministically) only one branch.
2) For every removed subtree with a, write a new a just above.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

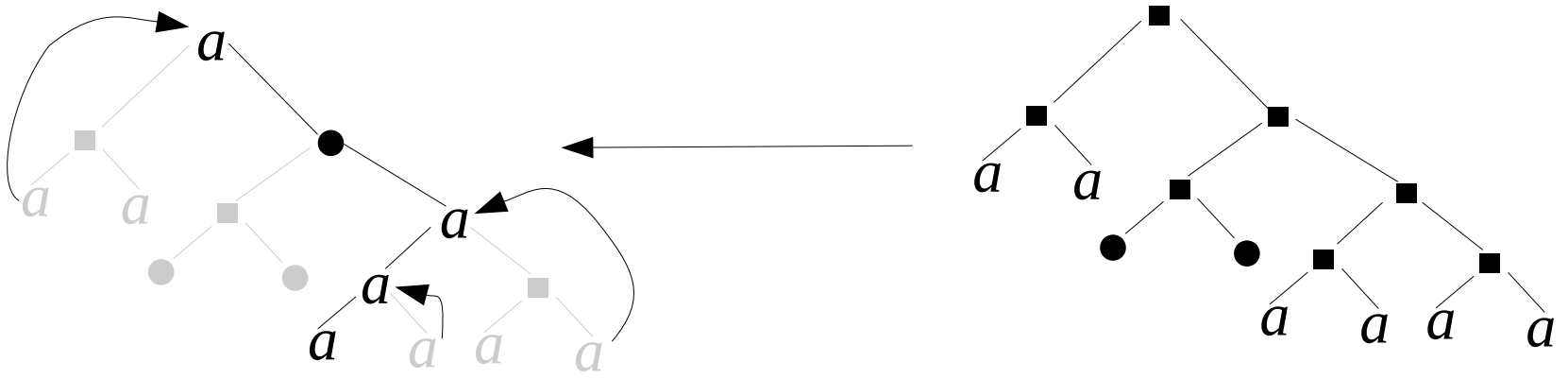
Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

—step 2→ a scheme S of order m-1 with a *similar* word written on a branch

Example:



Idea:

1) Choose (nondeterministically) only one branch.
2) For every removed subtree with a, write a new a just above.
3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

The diagonal problem

Input: a scheme S recognizing L, set of letters A

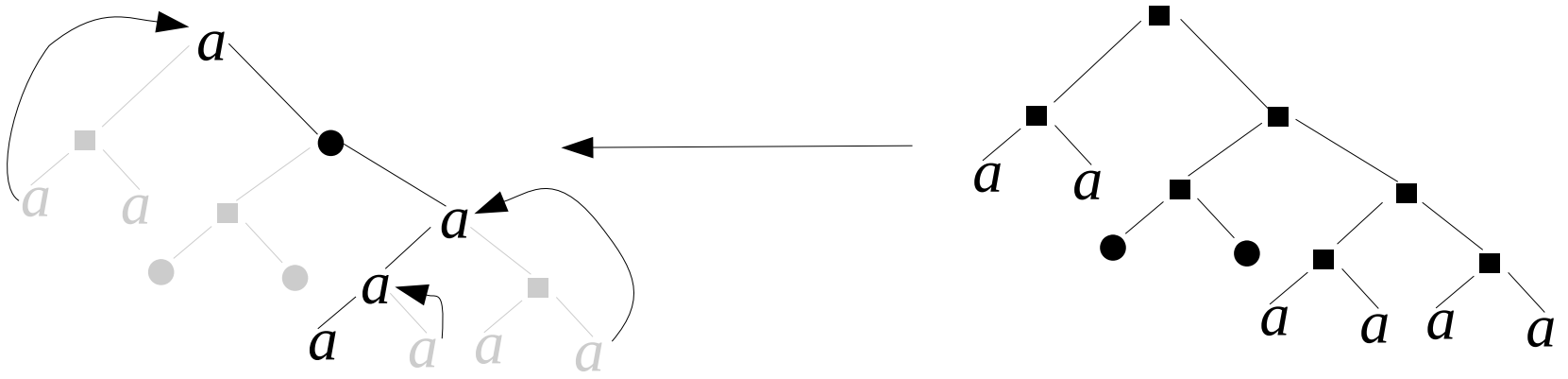
Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a word written on a branch —step 1→ a scheme S of order m-1 with this word written in leaves

—step 2→ a scheme S of order m-1 with a similar "word" written on $|A|$ branches

Example:



Idea:

1) Choose (nondeterministically) only one branch.
2) For every removed subtree with a, write a new a just above.
3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

We have to this for every letter ⇒ $|A|$ branches

The diagonal problem

Input: a scheme S recognizing L, set of letters A

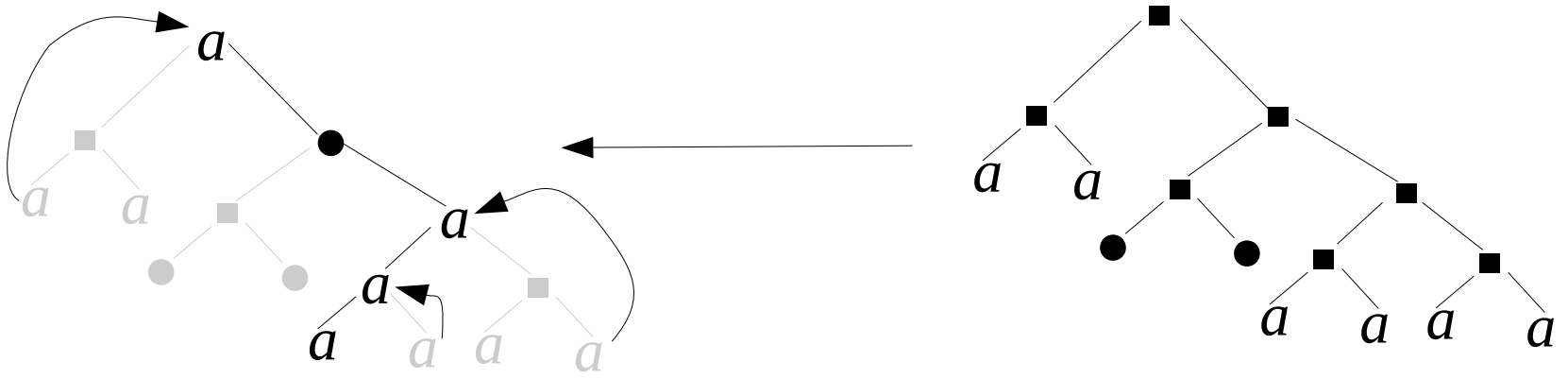
Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a "word" written on $|A|$ branches —step 1→ a scheme S of order m-1 with this "word" written in leaves

a scheme S of order m-1 with a similar "word" written on $|A|$ branches ←step 2—

Example:



Idea:

1) Choose (nondeterministically) only one branch.
2) For every removed subtree with a, write a new a just above.
3) The number of a's decreases at most logarithmically,
   if the branch is chosen correctly (always go to the subtree with more a's).

We have to this for every letter ⇒ $|A|$ branches

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a "word" written on $|A|$ branches

step 1 →

a scheme S of order m-1 with this "word" written in leaves

step 2 →

a scheme S of order m-1 with a similar "word" written on $|A|$ branches

Repeat these steps until the order drops down to 0,
and solve the diagonal problem for a regular language.

The diagonal problem

Input: a scheme S recognizing L, set of letters A

Question: is it the case that for every $n \in \mathbb{N}$ there is a word $w \in L$ containing every letter from A at least n times?

How to solve it?

a scheme S of order m with a "word" written on $|A|$ branches —step 1→ a scheme S of order m-1 with this "word" written in leaves

—step 2→ a scheme S of order m-1 with a similar "word" written on $|A|$ branches

Repeat these steps until the order drops down to 0, and solve the diagonal problem for a regular language.

Complexity of this solution?

- Every step applied to a scheme of order m increases its size m-fold exponentially.
- In total the complexity becomes about m^2-EXPTIME

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

- Basic idea: we give a type system that “simulates” all the steps of the previous algorithm at once. While deriving a type in this type system, we recover the solution to the diagonal problem.

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

- Basic idea: we give a type system that “simulates” all the steps of the previous algorithm at once. While deriving a type in this type system, we recover the solution to the diagonal problem.

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

- Basic idea: we give a type system that "simulates" all the steps of the previous algorithm at once. While deriving a type in this type system, we recover the solution to the diagonal problem.

Complexity:

- For schemes of order m (recognizing languages of words), the number of types is $(m\text{-}1)$-fold exponential.
  Thus the problem is in $(m\text{-}1)$-EXPTIME (or in NP for $m \in \{0,1\}$).

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

- Basic idea: we give a type system that "simulates" all the steps of the previous algorithm at once. While deriving a type in this type system, we recover the solution to the diagonal problem.

Complexity:

- For schemes of order m (recognizing languages of words), the number of types is $(m\text{-}1)$-fold exponential.
  Thus the problem is in $(m\text{-}1)$-EXPTIME (or in NP for $m \in \{0,1\}$).
- We also prove a corresponding lower bound: the problem is $(m\text{-}1)$-EXPTIME-complete.
- For schemes generating languages of trees, the problem is m-EXPTIME-complete (or NP-complete for $m=0$).

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

Complexity:

- For schemes of order m (recognizing languages of words), the number of types is $(m\text{-}1)$-fold exponential.
  Thus the problem is in $(m\text{-}1)$-EXPTIME (or in NP for $m \in \{0,1\}$).
- We also prove a corresponding lower bound: the problem is $(m\text{-}1)$-EXPTIME-complete.
- For schemes generating languages of trees, the problem is m-EXPTIME-complete (or NP-complete for $m=0$).
- This complexity is high. But all known problems for schemes of order m are at least $(m\text{-}1)$-EXPTIME-hard, even the question whether the recognized language is nonempty. Beside of that, for some problems there exist tools that work in practice.
- All algorithms for schemes that were implemented are based on type systems, so also our algorithm have the potential of working in practice (typing is complicated in the worst-case, but usually it is much easier).

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

Advantages of our algorithm:

- Faster
- Based on a type system
- It gives also the reflection property:
  Recall that we can see a recursion scheme as a generator of an infinite tree, containing all words of the recognized language on branches. Given a scheme G we can construct a scheme H which generates the same tree as G, but enriched with additional labels: the label of every node of the tree says what is the answer to the diagonal problem for the subtree starting in this node (thus we have not only the answer for the whole tree, but also for every subtree).

Contribution

In this paper we give a new algorithm solving the diagonal problem for recursion schemes.

Further work:

- The algorithm for the diagonal can be used for model-checking trees generated by schemes against formulas of the WMSO+U logic [P., submitted to STACS 2018]
- This logic extends the MSO logic (weak variant – only quantification over finite sets) with the U quantifier:
  U$X.\phi$ means that ϕ holds for some arbitrarily large finite sets X

Thank you!