Intersection Types for Unboundedness Problems

Paweł Parys

University of Warsaw

ITRS 2018

Our setting

Intersection types can be used as:

- an extension of simple types (mostly undecidable)
- a refinement of simple types (mostly decidable)

this talk

Our setting

We consider infinitary, simply-typed λ-calculus and simply-typed λY-calculus.

Our setting

We consider infinitary, simply-typed λ-calculus
and simply-typed λY-calculus.

Simple types (sorts): o, $o\to(o\to o)$, $(o\to o)\to o$, $(o\to o)\to(((o\to o)\to o)\to o)$

Our setting

We consider infinitary, simply-typed λ-calculus
and simply-typed λY-calculus.

Simple types (sorts): $\underset{0}{o}$, $\underbrace{o \to o \to o}_{1}$, $\underbrace{(o \to o) \to o}_{2}$, $\underbrace{(o \to o) \to ((o \to o) \to o) \to o}_{3}$

Order: $\mathrm{ord}(o)=0$, $\mathrm{ord}(\alpha \to \beta)=\max(\mathrm{ord}(\alpha)+1, \mathrm{ord}(\beta))$

Our setting

We consider infinitary, simply-typed λ-calculus
and simply-typed λY-calculus.

Simple types (sorts): $\underset{0}{o}$, $\underbrace{o\to o\to o}_{1}$, $\underbrace{(o\to o)\to o}_{2}$, $\underbrace{(o\to o)\to((o\to o)\to o)\to o}_{3}$

Order: $ord(o)=0$, $ord(\alpha\to\beta)=\max(ord(\alpha)+1, ord(\beta))$

λ-terms:

- variables: x^α, y^β, ...
- constants: a^α, b^β, … – only for sorts of order ≤ 1
- λ-abstraction: $(\lambda x^\alpha.K^\beta)^{\alpha\to\beta}$
- application: $(K^{\alpha\to\beta}\, L^\alpha)^\beta$
  + coinduction

Every term has a particular sort.

We assume that all arguments of constants are already applied:
$a^{o\to o\to o}\, K^o\, L^o$ is allowed, but $a^{o\to o\to o}\, K^o$ is not allowed

Our setting – λY-calculus

λY-term is a finite representation of an infinite λ-term:

- In a λY-term we may use a binder "Y"
- Meaning:

  $(Yx^{\alpha}.M^{\alpha})^{\alpha}$ - this is the unique (infinite) λ-term such that

  $Yx.M = M[Yx.M/x]$

Example:

the λY-term: Yx.((λy.ay) x)

represents the λ-term: ((λy.ay) ((λy.ay) ((λy.ay) ((λy.ay) ...))))

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

$\lambda x_1.\cdots.\lambda x_n.y\, M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\, M_1 \ldots M_k$

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

  $\lambda x_1.\cdots.\lambda x_n.y\,M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\,M_1 \ldots M_k$
- We may reduce each $M_1, \ldots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

  $\lambda x_1.\cdots.\lambda x_n.y\, M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\, M_1 \ldots M_k$
- We may reduce each $M_1, \ldots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Suppose that:

- ➔ K is of sort o
- ➔ K has no free variables
- ➔ we only use constants of order ≤ 1.

Then the Böhm tree is a tree built out of constants.

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

$\lambda x_1.\cdots.\lambda x_n.y\,M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\,M_1 \ldots M_k$

- We may reduce each $M_1, \ldots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Suppose that:

- K is of sort o
- K has no free variables
- we only use constants of order ≤ 1.

Then the Böhm tree is a tree built out of constants.

Example:

$$Yx.((\lambda y.ay)\ x) = ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ...))))$$

$$\downarrow$$

$$(a\ ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ...))))$$

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

  $\lambda x_1.\cdots.\lambda x_n.y\, M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\, M_1 \ldots M_k$
- We may reduce each $M_1, \ldots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Suppose that:
- ➔ K is of sort o
- ➔ K has no free variables
- ➔ we only use constants of order ≤ 1.

Then the Böhm tree is a tree built out of constants.

Example:

$$Yx.((\lambda y.ay)\ x) = ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ \ldots))))$$
$$\downarrow$$
$$(a\ (a\ ((\lambda y.ay)\ ((\lambda y.ay)\ \ldots))))$$

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

  $\lambda x_1.\cdots.\lambda x_n.y\, M_1 \ldots M_k$ or $\lambda x_1.\cdots.\lambda x_n.a\, M_1 \ldots M_k$
- We may reduce each $M_1, \ldots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Suppose that:

- ➔ K is of sort o
- ➔ K has no free variables
- ➔ we only use constants of order ≤ 1.

Then the Böhm tree is a tree built out of constants.

Example:

$$Yx.((\lambda y.ay)\ x) = ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ \ldots))))$$

$$\downarrow$$

$$(a\ (a\ (a\ ((\lambda y.ay)\ \ldots))))$$

Our setting – Böhm trees

- Every finite λ-term K reduces to a term in β-normal form.
- Every (infinite) λ-term K reduces to term in head-β-normal form, i.e.:

$$\lambda x_1.\cdots.\lambda x_n.y\, M_1 \dots M_k \quad \text{or} \quad \lambda x_1.\cdots.\lambda x_n.a\, M_1 \dots M_k$$

- We may reduce each $M_1, \dots, M_k$ to head-β-normal form, etc.
- The limit is called the *Böhm tree* of K.

Suppose that:

- K is of sort o
- K has no free variables
- we only use constants of order ≤ 1.

Then the Böhm tree is a tree built out of constants.

Example:

$$Yx.((\lambda y.ay)\ x) = ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ ((\lambda y.ay)\ \dots))))$$

$$\downarrow$$

$$(a\ (a\ (a\ (a\ \dots))))$$

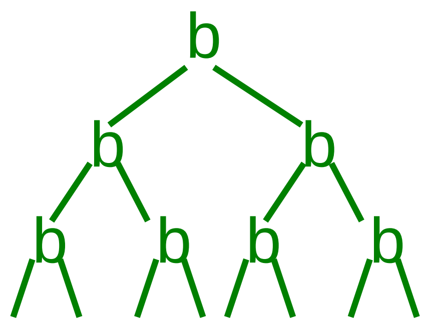
a
|
a
|
a
|
a
|
...

Our setting – Böhm trees

Example:

$$Yx.((\lambda y.byy)\ x) = ((\lambda y.byy)\ ((\lambda y.byy)\ ((\lambda y.byy)\ ((\lambda y.byy)\ \ldots))))$$

$$(b\,(b\ (b\ \ldots)\ (b\ \ldots))\,(b\ (b\ \ldots)\ (b\ \ldots)))$$



Equivalent formalisms - trees generated by:

- higher-order recursion schemes (HORSes)
- collapsible pushdown automata
- ordered tree-pushdown automata

Intersection types for λY-calculus – general setting

In the context of λY-calculus (recursion schemes), intersection types were used for:

- $\boxed{\text{model checking}}$
- $\boxed{\text{transformation of schemes}}$
- pumping

this talk

Plan

1) model checking for co-trivial tree automata (via intersection types)

2) transformation “words → trees” (via intersection types)
   + how to use it to solve unboundedness problems

3) unboundedness problems (directly via intersection types)

Motivation: from program verification to recursion schemes

Example

```
open(x, “foo”)
a := 0
while a<100 do
     read(x)
     a := a+1
close(x)
```

is the file “foo” accessed according to open,read*,close?

Motivation: from program verification to recursion schemes

Example

Step 1: information about infinite data domains is approximated.

```
open(x, "foo")
a := 0
while a<100 do
    read(x)
    a := a+1
close(x)
```

```
open(x, "foo")

while * do
    read(x)

close(x)
```

is the file "foo" accessed according to open,read*,close?

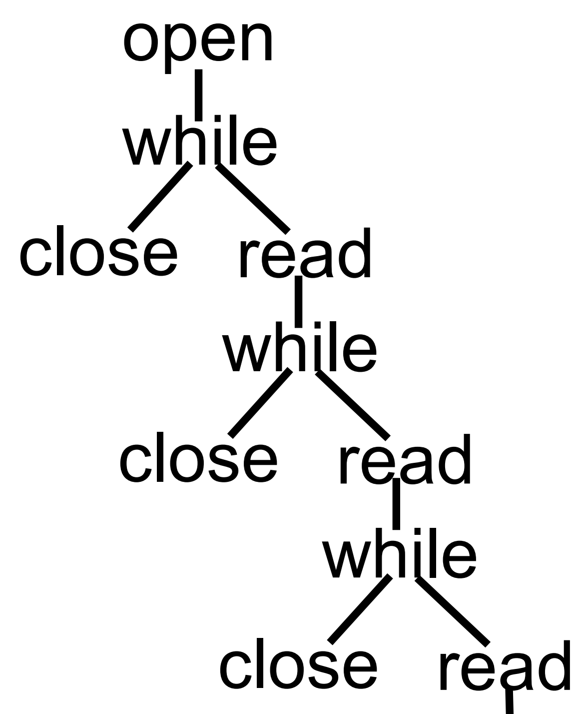
is the file "foo" accessed according to open,read*,close?

Motivation: from program verification to recursion schemes

Example

Step 2: consider the tree of possible control flows.

```
open(x, “foo”)
while * do
    read(x)
close(x)
```



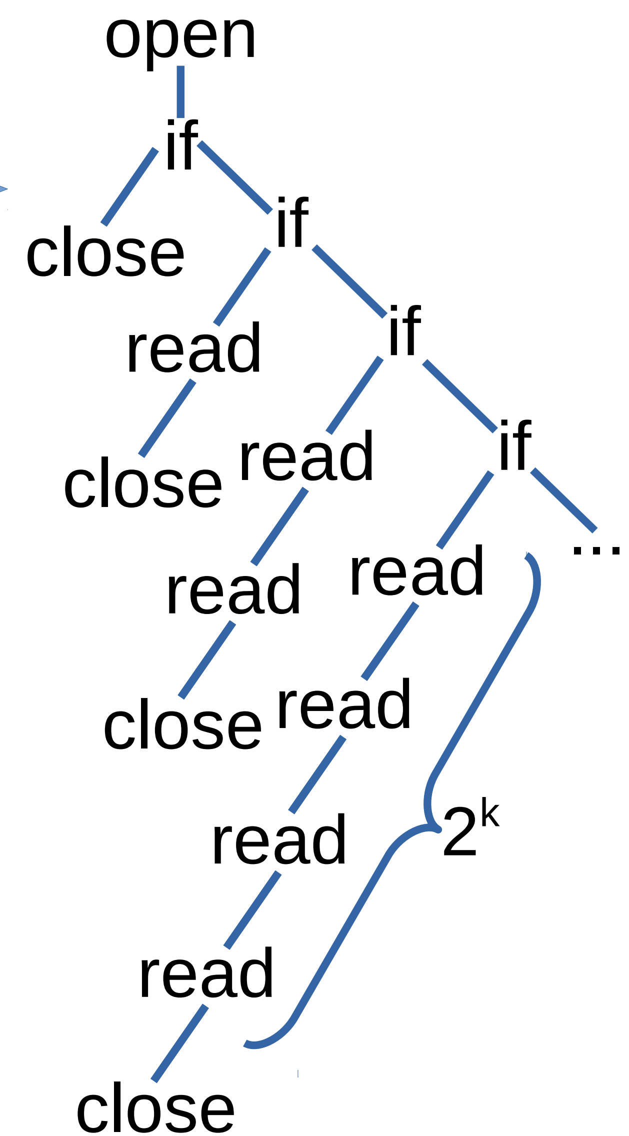
is the file “foo” accessed according to open,read*,close?

is each path labelled by open,read*,close?

Motivation: from program verification to recursion schemes

What about higher order programs?

```
let f(x, g) =
    if * then g(x)
    else f(x, fun h x -> h(x); h(x))
open(x)
f(x, read)
close(x)
```



- For programs without recursion, each path of the tree is a regular language.
- Programs with (higher-order) recursion can be approximated by recursion schemes

questions about programs → questions about Bohm trees

Intersection types describing co-trivial ATA

We fix some alternating tree automaton:

Q – set of states

Δ – set of transitions of the form $(q,a) \to (Q_1, ..., Q_r)$ where $r=arity(a)$

q_0 – initial state (for the root of the tree)

Intersection types describing co-trivial ATA

We fix some alternating tree automaton:

Q – set of states

Δ – set of transitions of the form $(q,a) \to (Q_1, \ldots, Q_r)$ where $r=arity(a)$

q_0 – initial state (for the root of the tree)

Run on a tree t = labeling of nodes of t by sets of states

- if a node v is labeled by S, and its children by $S_1, \ldots, S_r$, then for every $q \in S$ there is a transition $(q,a) \to (Q_1, \ldots, Q_r)$ with $Q_1 \subseteq S_1, \ldots, Q_r \subseteq S_r$

Intersection types describing co-trivial ATA

We fix some alternating tree automaton:
Q – set of states
Δ – set of transitions of the form $(q,a) \to (Q_1, ..., Q_r)$ where $r=arity(a)$
q_0 – initial state (for the root of the tree)

Run on a tree t = labeling of nodes of t by sets of states

- if a node v is labeled by S, and its children by $S_1, ..., S_r$, then for every $q \in S$ there is a transition $(q,a) \to (Q_1, ..., Q_r)$ with $Q_1 \subseteq S_1, ..., Q_r \subseteq S_r$
- co-trivial accepting condition: only finitely many nodes are labeled by nonempty sets

Intersection types describing co-trivial ATA

We fix some alternating tree automaton:
Q – set of states
Δ – set of transitions of the form $(q,a) \to (Q_1, ..., Q_r)$ where $r=arity(a)$
q_0 – initial state (for the root of the tree)

Run on a tree t = labeling of nodes of t by sets of states

- if a node v is labeled by S, and its children by $S_1, ..., S_r$, then for every $q \in S$ there is a transition $(q,a) \to (Q_1, ..., Q_r)$ with $Q_1 \subseteq S_1, ..., Q_r \subseteq S_r$
- co-trivial accepting condition: only finitely many nodes are labeled by nonempty sets

Goal: given an automaton A and a term K, decide whether A accepts the Böhm tree of K.
We can achieve this goal using a type system of intersection types:
a type τ can be derived for $K \Leftrightarrow A$ accepts $BT(K)$
[Broadbent, Kobayashi – CSL 2013]

Intersection types describing co-trivial ATA

Intersection types:

- describe behavior of the automaton
- refine simple types (sorts): for every sort α we have a set Types^{α} of types refining sort α

Type judgments:

$$\Gamma \vdash K : \tau$$

Intersection types describing co-trivial ATA

Intersection types:

- describe behavior of the automaton
- refine simple types (sorts): for every sort α we have a set $Types^{\alpha}$ of types refining sort α

Type judgments:

$$\Gamma \vdash K : \tau$$

$Types^{o}=Q$

A term of sort o (a tree, or a term that generates a tree) has type q (where $q \in Q$) if the tree can be accepted from state q

Intersection types describing co-trivial ATA

Intersection types:

- describe behavior of the automaton
- refine simple types (sorts): for every sort α we have a set Types^{α} of types refining sort α

Type judgments:

$$\Gamma \vdash K : \tau$$

$\mathit{Types}^{o} = Q$

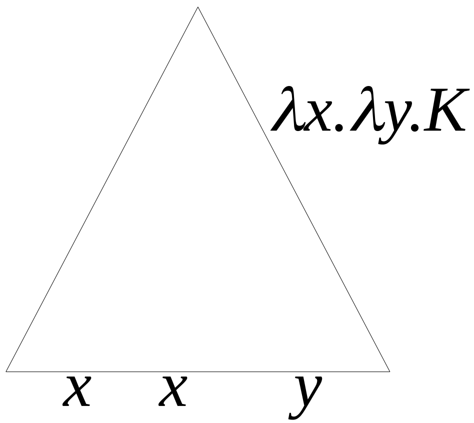
A term of sort o (a tree, or a term that generates a tree) has type q (where $q \in Q$) if the tree can be accepted from state q

For each transition $(q,a) \rightarrow (Q_1, ..., Q_r)$ of A we have a typing rule:

$$\frac{\Gamma \vdash K_i : p \quad \text{for each } i \in \{1,\ldots,r\} \text{ and each } p \in Q_i}{\Gamma \vdash a\, K_1 \ldots K_r : q}$$

Intersection types describing co-trivial ATA

Terms of order 1 describe fragments of trees:



$Types^{o\to o\to o} = P(Q)\times P(Q)\times Q$

such a type is of the form $Q_x \to Q_y \to q$

(it says that if the subtree given as the first argument is accepted from all states in Q_x, and the subtree given as the second argument is accepted from all states in Q_y, then the whole tree can be accepted from q)

Remark: Q_x has to be a set of states, not a single state, even if we consider nondeterministic automata instead of alternating automata, because x can appear multiple times in K.

Intersection types describing co-trivial ATA

In general:

$$Types^{o} = Q$$

$$Types^{\alpha \to \beta} = P(Types^{\alpha}) \times Types^{\beta}$$

Elements of $Types^{\alpha \to \beta}$ are written as $\Psi \to \tau$

Intersection types describing co-trivial ATA

In general:

$Types^{o}=Q$
$Types^{\alpha\to\beta}=P(Types^{\alpha})\times Types^{\beta}$

Elements of $Types^{\alpha\to\beta}$ are written as $\Psi\to\tau$

Typing rules:

$$\frac{\Gamma\vdash K_i : p \quad \text{for each } i\in\{1,\dots,r\} \text{ and each } p\in Q_i}{\Gamma\vdash a\, K_1 \dots K_r : q}$$

$$\frac{\Gamma\vdash K : \Psi\to\tau \qquad \Gamma\vdash L : \sigma \quad \text{for each } \sigma\in\Psi}{\Gamma\vdash K\, L : \tau}$$

$$\frac{\tau\in\Gamma(x)}{\Gamma\vdash x : \tau} \qquad \frac{\Gamma[x\to\Psi]\vdash K : \tau}{\Gamma\vdash \lambda x.K : \Psi\to\tau}$$

Intersection types describing co-trivial ATA

Typing rules:

$$\frac{\Gamma \vdash K_i : p \quad \text{for each } i \in \{1,\ldots,r\} \text{ and each } p \in Q_i}{\Gamma \vdash a\, K_1 \ldots K_r : q}$$

$$\frac{\Gamma \vdash K : \Psi \to \tau \qquad \Gamma \vdash L : \sigma \quad \text{for each } \sigma \in \Psi}{\Gamma \vdash K\, L : \tau}$$

$$\frac{\tau \in \Gamma(x)}{\Gamma \vdash x : \tau} \qquad\qquad \frac{\Gamma[x \to \Psi] \vdash K : \tau}{\Gamma \vdash \lambda x.K : \Psi \to \tau}$$

K may be infinite

Lemma: For a closed term K of sort o, and for a state q,

there is a finite derivation of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Intersection types describing co-trivial ATA

Typing rules:

$$\frac{\Gamma \vdash K_i : p \quad \text{for each } i \in \{1,\ldots,r\} \text{ and each } p \in Q_i}{\Gamma \vdash a\, K_1 \ldots K_r : q}$$

$$\frac{\Gamma \vdash K : \Psi \to \tau \qquad \Gamma \vdash L : \sigma \quad \text{for each } \sigma \in \Psi}{\Gamma \vdash K\, L : \tau}$$

$$\frac{\tau \in \Gamma(x)}{\Gamma \vdash x : \tau} \qquad\qquad \frac{\Gamma[x \to \Psi] \vdash K : \tau}{\Gamma \vdash \lambda x.K : \Psi \to \tau}$$

K may be infinite

Lemma: For a closed term K of sort o, and for a state q,

there is a finite derivation of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Lemma 2: If we consider A with trivial accepting condition, instead of co-trivial (if we allow infinite runs of A), we have the equivalence

there is a derivation (arbitrary – possibly infinite) of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Intersection types describing co-trivial ATA

Lemma: For a closed term K of sort o, and for a state q,

there is a finite derivation of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Proof sketch:

1) If $M \rightarrow_\beta N$, then $\Gamma \vdash M : \tau \Leftrightarrow \Gamma \vdash N : \tau$

2) For $K = BT(K)$ the lemma is trivial (only rules for a constant are used)

3) Both sides of the lemma talk only about finite prefixes of the term, so we can assume that K is finite. Then $K \rightarrow_\beta^* BT(K)$.

Intersection types describing co-trivial ATA

Lemma: For a closed λ-term K of sort o, and for a state q,

there is a finite derivation of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Goal: given an automaton A and a finite λY-term K', decide whether A accepts $BT(K')$.

Intersection types describing co-trivial ATA

Lemma: For a closed λ-term K of sort o, and for a state q,

there is a finite derivation of $\varepsilon \vdash K : q$ $\Leftrightarrow$ A accepts $BT(K)$ from state q

Goal: given an automaton A and a finite λY-term K', decide whether A accepts $BT(K')$.

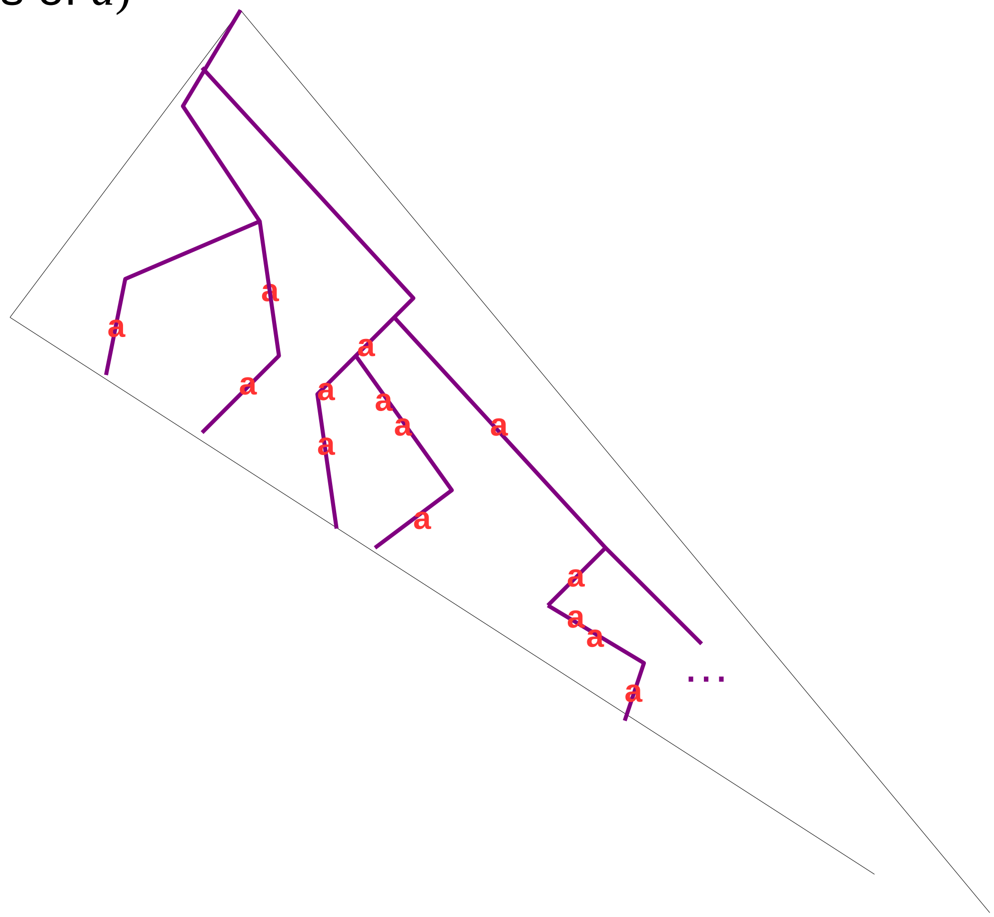
- Recall that K' is a finite representation of an infinite λY-term K.
- Seeing K' we have to check whether a type judgment can be derived for K.
- I.e., seeing $Yx.M$, we have to check which type judgments can be derived for $M[M[M[M[M[...]/x]/x]/x]/x]$.
- This is an easy fixpoint computation.

Unboundedness – basic problem

Input: closed λY-term K of sort o (i.e. infinite λ-term represented in a finite way)

Question: In the Böhm tree of K, are there (finite) branches with arbitrarily many symbols "*a*"?

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

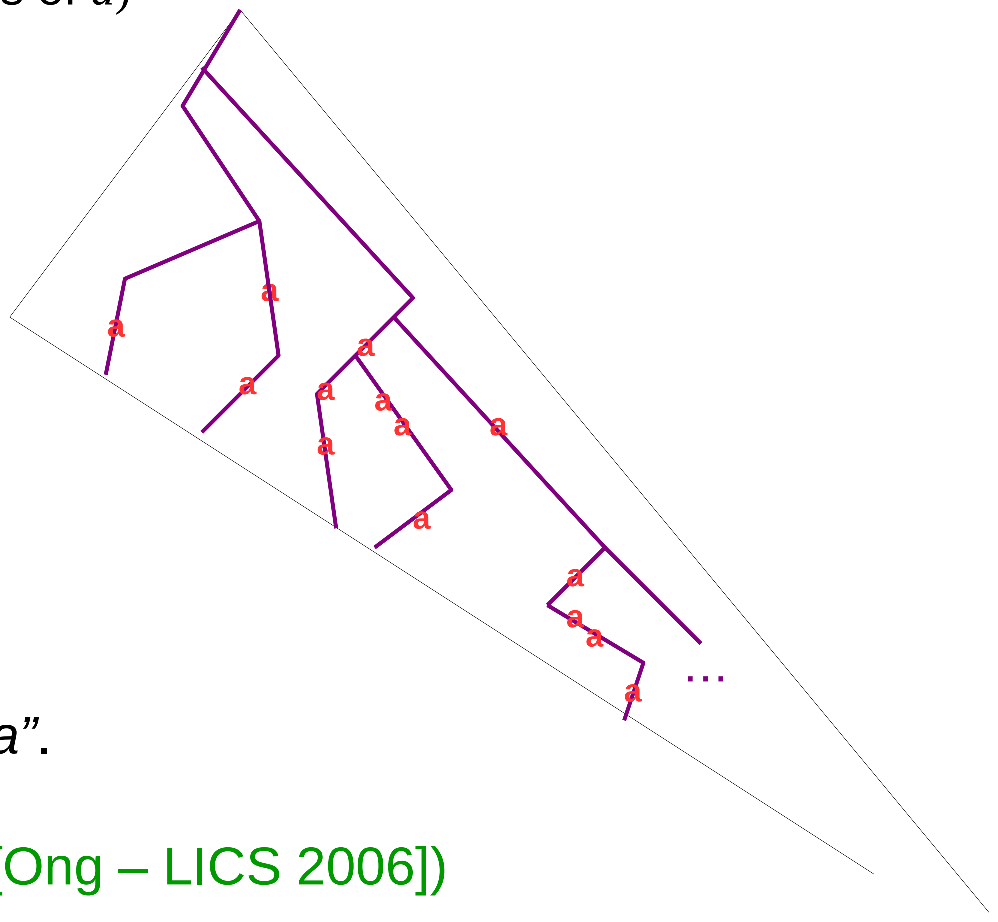


Unboundedness – basic problem

Input: closed λY-term K of sort o (i.e. infinite λ-term represented in a finite way)

Question: In the Böhm tree of K, are there (finite) branches with arbitrarily many symbols “*a*”?

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



Notice:

There may be no path with infinitely many „*a*”.

Our property **is not regular!!!**

(regular properties can be checked e.g. by [Ong – LICS 2006])

Unboundedness – basic problem

Input: closed λY-term K of sort o (i.e. infinite λ-term represented in a finite way)

Question: In the Böhm tree of K, are there (finite) branches
with arbitrarily many symbols "*a*"?
($\forall n\ \exists$branch with $>n$ appearances of a)

This is an instance of a more general problem, called **diagonal problem** or **simultaneous unboundedness problem (SUP)**:

Input: closed λY-term K of sort o, set A of symbols

Question: In the Böhm tree of K, are there (finite) branches with
arbitrarily many appearances of every symbol from A?
($\forall n\ \exists$branch $\forall a \in A$ there are $>n$ appearances of a on the branch)

This problem is decidable
[Hague, Kochems, Ong – POPL 2016],
[Clemente, P., Salvati, Walukiewicz – LICS 2016]

Unboundedness – basic problem

Input: closed λY-term K of sort o (i.e. infinite λ-term represented in a finite way)

Question 1: In the Böhm tree of K, are there finite branches with arbitrarily many symbols "*a*"?

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

Solution – preparation:

We generalize the problem to nondeterministic terms (aka nondeterministic recursion schemes).

- We add a new construct: $nd\ K^{\alpha}\ L^{\alpha}$
- We add reduction rules: $nd\ K\ L \to K$ and $nd\ K\ L \to L$
- Now there is no one unique Bohm tree
  Instead, we have a set of finite trees (normal forms) of a (closed, of sort o, potentially infinite) lambda-term K; we denote this set $\mathcal{L}(K)$

Unboundedness – basic problem

Input: closed λY-term K of sort o (i.e. infinite λ-term represented in a finite way)

Question 1: In the Böhm tree of K, are there finite branches with arbitrarily many symbols "*a*"?

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

Solution – preparation:

We generalize the problem to nondeterministic terms (aka nondeterministic recursion schemes).

- We add a new construct: $nd\ K^{\alpha}\ L^{\alpha}$
- We add reduction rules: $nd\ K\ L \to K$ and $nd\ K\ L \to L$
- Now there is no one unique Bohm tree
  Instead, we have a set of finite trees (normal forms) of a (closed, of sort o, potentially infinite) lambda-term K; we denote this set $\mathcal{L}(K)$
- New question: are there trees in $\mathcal{L}(K)$ with arbitrarily many symbols "*a*"?
- Easy reduction from question 1 to the new question:
  replace every appearance of $a\ M\ N$ by $a\ (nd\ M\ N)$;
  then $\mathcal{L}(K')$ is the set of branches $BT(K)$
- In particular all symbols in K' are of arity 0 and 1

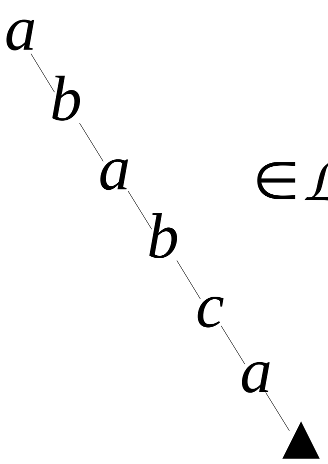
Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

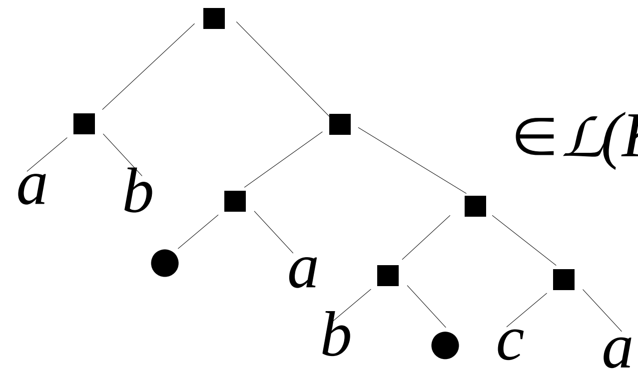
Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

How to solve it?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches $\xrightarrow{\text{step 1}}$ a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves



$\in \mathcal{L}(K)$



$\in \mathcal{L}(K')$

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols “*a*”?

How to solve it?

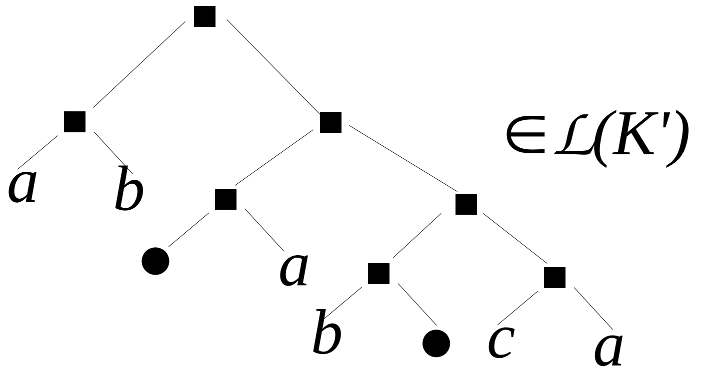
a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches

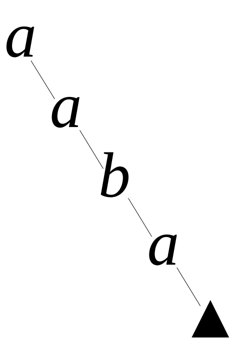
step 1

a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

step 2

a term K'' of order m-1, where $\mathcal{L}(K'')$ has *similar* words written on branches





$\in \mathcal{L}(K'')$

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

How to solve it?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches

step 1

a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

step 2

a term K'' of order m-1, where $\mathcal{L}(K'')$ has *similar* words written on branches

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

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

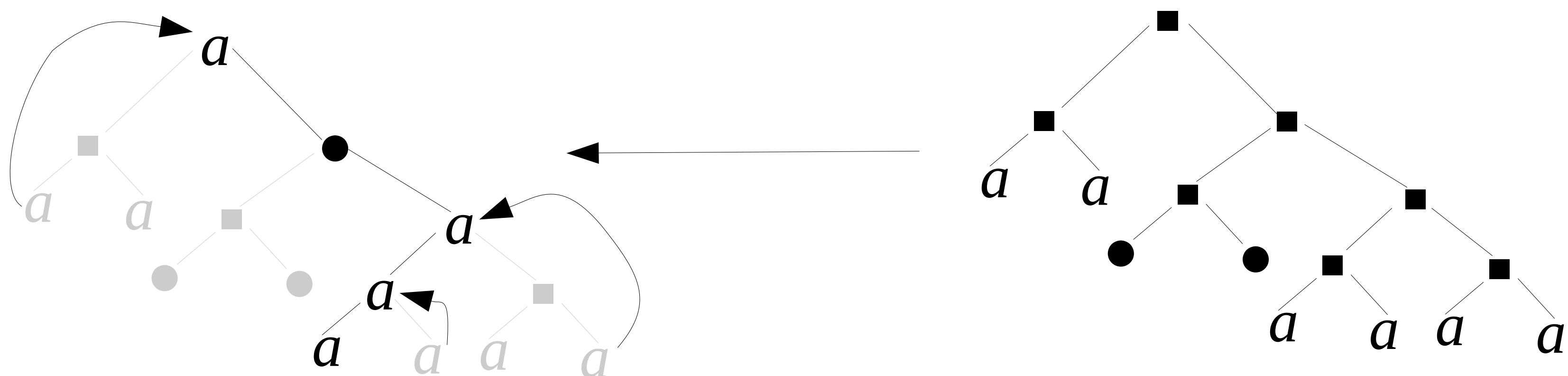
Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

step 2

a term K'' of order m-1, where $\mathcal{L}(K'')$ has *similar* words written on branches

Example:



Idea:

1) Choose (nondeterministically) only one branch.

2) For every removed subtree with a, write a new a just above.

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

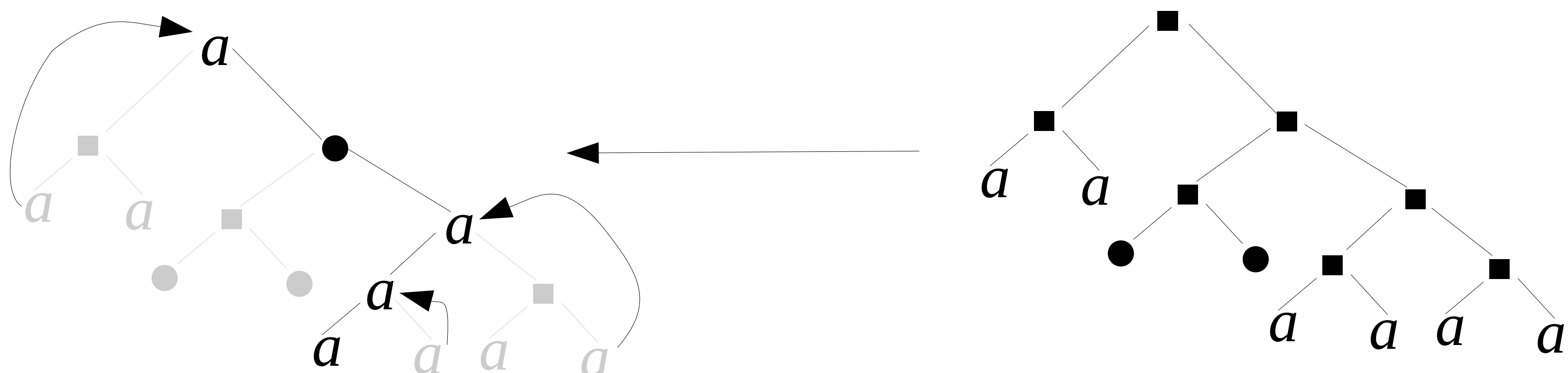
Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

step 2

a term K'' of order m-1, where $\mathcal{L}(K'')$ has *similar* words written on 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 skip the details.

Unboundedness – basic problem

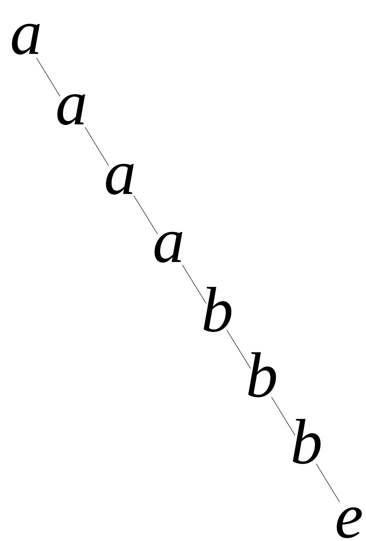
Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

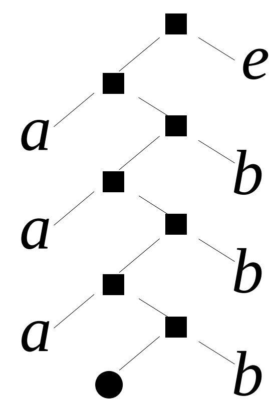
Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches $\xrightarrow{\text{step 1}}$ a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

Example:

$$\begin{aligned} S &\to A\; e\; c \\ A\; x\; y &\to a\; (A\; (b\; x)\; (d\; x)) \\ A\; x\; y &\to x \end{aligned} \quad\longrightarrow\quad \begin{aligned} S &\to \blacksquare\; A\; e \\ A &\to \blacksquare\; a\; (\blacksquare\; A\; b)) \\ A &\to \bullet \end{aligned}$$





[Asada, Kobayashi – ICALP 2016]

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches $\xrightarrow{\text{step 1}}$ a term K' of order $m\text{-}1$, where in $\mathcal{L}(K')$ these words are written in leaves

Example:

$$\begin{aligned} S &\to A\,e\,c \\ A\,x\,y &\to a\,(A\,(b\,x)\,(d\,x)) \\ A\,x\,y &\to x \end{aligned} \quad\longrightarrow\quad \begin{aligned} S &\to \blacksquare\ A\,e \\ A &\to \blacksquare\ a\,(\blacksquare\ A\,b)) \\ A &\to \bullet \end{aligned}$$

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

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "*a*"?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches $\xrightarrow{\text{step 1}}$ a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

Example:

$$\begin{array}{lcl} S & \to & A\; e\; c \\ A\; x\; y & \to & a\; (A\; (b\; x)\; (d\; x)) \\ A\; x\; y & \to & x \end{array} \longrightarrow \begin{array}{lcl} S & \to & \blacksquare\; A\; e \\ A & \to & \blacksquare\; a\; (\blacksquare\; A\; b)) \\ A & \to & \bullet \end{array}$$

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

2) All arguments of type o are dropped (⇒ 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).

Unboundedness – basic problem

Input: nondeterministic closed λY-term K of sort o (symbols of arity 0 & 1)

Question: are there trees (paths) in $\mathcal{L}(K)$ with arb. many symbols "a"?

a term K of order m, where $\mathcal{L}(K)$ is a set of words written on branches $\xrightarrow{\text{step 1}}$ a term K' of order m-1, where in $\mathcal{L}(K')$ these words are written in leaves

Example:

$$S \to A\, e\, c \qquad\qquad S \to \blacksquare\, A\, e$$
$$A\, x\, y \to a\, (A\, (b\, x)\, (d\, x)) \quad\longrightarrow\quad A \to \blacksquare\, a\, (\blacksquare\, A\, b))$$
$$A\, x\, y \to x \qquad\qquad A \to \bullet$$

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

2) All arguments of type o are dropped (⇒ order decreases).

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

a) either by ▪ $M\ N$ (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). We use intersection types here.

Type-guided transformation

Difficulty to overcome: given a nondeterministic closed λY-term K of sort o, with symbols of arity 0 & 1 only, we want to say for every its subterm M of order 0 whether M

- is "used in the generated tree", or (equivalently)
- is "responsible for creating the leaf of the generated tree"

We use intersection types to achieve this goal!

Type-guided transformation

Difficulty to overcome: given a nondeterministic closed λY-term K of sort o, with symbols of arity 0 & 1 only, we want to say for every its subterm M of order 0 whether M

- is “used in the generated tree”, or (equivalently)
- is “responsible for creating the leaf of the generated tree”

We use intersection types to achieve this goal!

Before we start:

- Notice that the considered property depends of the choice of the generated tree: maybe one tree uses M to generate the leaf, and another tree does not.
- Thus, we first guess whether M generates the leaf (nondeterministic choice), and then we make sure that the choice is respected.

Type-guided transformation

Difficulty to overcome: given a nondeterministic closed λY-term K of sort o, with symbols of arity 0 & 1 only, we want to say for every its subterm M of order 0 whether M

- is “used in the generated tree”, or (equivalently)
- is “responsible for creating the leaf of the generated tree”

We use intersection types to achieve this goal!

Before we start:

- Notice that the considered property depends of the choice of the generated tree: maybe one tree uses M to generate the leaf, and another tree does not.
- Thus, we first guess whether M generates the leaf (nondeterministic choice), and then we make sure that the choice is respected.

Let us first present the type system itself;
then, we present the transformation.

Type-guided transformation

For terms of sort o we need two types:

- this term is responsible for creating the leaf – denoted $(1,o)$;
- this term is not responsible for creating the leaf – denoted $(0,o)$.

$Types^o=\{0,1\}\times\{o\}$

Type-guided transformation

For terms of sort o we need two types:

- this term is responsible for creating the leaf – denoted $(1,o)$;
- this term is not responsible for creating the leaf – denoted $(0,o)$.

$Types^o = \{0,1\} \times \{o\}$

Rules:

$$\frac{}{\Gamma \vdash e : (1,o)} \qquad \frac{\Gamma \vdash K : (s,o)}{\Gamma \vdash a\,K : (s,o)}$$

Type-guided transformation

In general, for terms of sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

- a type is of the form $(s, \Psi_1 \to \ldots \to \Psi_k \to o)$,

  where $s \in \{0,1\}$, and $\Psi_i \subseteq Types^{\alpha_i}$
- In other words: $Types^{\alpha} = \{0,1\} \times P(Types^{\alpha_1}) \times \ldots \times P(Types^{\alpha_1}) \times \{o\}$
- s says whether the term is responsible for creating the leaf
- Ψ_i is the set of types needed for the i-th argument

Type-guided transformation

In general, for terms of sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

- a type is of the form $(s, \Psi_1 \to \ldots \to \Psi_k \to o)$,

  where $s \in \{0,1\}$, and $\Psi_i \subseteq Types^{\alpha_i}$
- In other words: $Types^{\alpha} = \{0,1\} \times P(Types^{\alpha_1}) \times \ldots \times P(Types^{\alpha_1}) \times \{o\}$
- s says whether the term is responsible for creating the leaf
- Ψ_i is the set of types needed for the i-th argument

Rules:

$$\frac{}{\Gamma \vdash e : (1,o)} \qquad \frac{\Gamma \vdash K : (s,o)}{\Gamma \vdash a\, K : (s,o)}$$

$$\frac{\Gamma \vdash K : \tau}{\Gamma \vdash nd\, K\, L : \tau} \qquad \frac{\Gamma \vdash L : \tau}{\Gamma \vdash nd\, K\, L : \tau}$$

Type-guided transformation

In general, for terms of sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

- a type is of the form $(s, \Psi_1 \to \ldots \to \Psi_k \to o)$,

  where $s \in \{0,1\}$, and $\Psi_i \subseteq Types^{\alpha_i}$
- In other words: $Types^{\alpha} = \{0,1\} \times P(Types^{\alpha_1}) \times \ldots \times P(Types^{\alpha_1}) \times \{o\}$
- s says whether the term is responsible for creating the leaf
- Ψ_i is the set of types needed for the i-th argument

Rules:

$$\frac{}{\Gamma \vdash e : (1,o)} \qquad \frac{\Gamma \vdash K : (s,o)}{\Gamma \vdash a\, K : (s,o)}$$

$$\frac{\Gamma \vdash K : \tau}{\Gamma \vdash nd\; K\; L : \tau} \qquad \frac{\Gamma \vdash L : \tau}{\Gamma \vdash nd\; K\; L : \tau}$$

$$\frac{\tau \in \Gamma(x)}{\Gamma \vdash x : \tau}$$

Type-guided transformation

In general, for terms of sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

- a type is of the form $(s, \Psi_1 \to \ldots \to \Psi_k \to o)$,

  where $s \in \{0,1\}$, and $\Psi_i \subseteq Types^{\alpha_i}$
- In other words: $Types^{\alpha} = \{0,1\} \times P(Types^{\alpha_1}) \times \ldots \times P(Types^{\alpha_1}) \times \{o\}$
- s says whether the term is responsible for creating the leaf
- Ψ_i is the set of types needed for the i-th argument

Rules:

$$\frac{}{\Gamma \vdash e : (1,o)} \qquad \frac{\Gamma \vdash K : (s,o)}{\Gamma \vdash a\, K : (s,o)}$$

$$\frac{\Gamma \vdash K : \tau}{\Gamma \vdash nd\, K\, L : \tau} \qquad \frac{\Gamma \vdash L : \tau}{\Gamma \vdash nd\, K\, L : \tau}$$

$$\frac{\tau \in \Gamma(x)}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma[x \to \Psi] \vdash K : (s,\sigma)}{\Gamma \vdash \lambda x.K : (s', \Psi \to \sigma)}$$

where $s'=0$, $s=1$ if Ψ contains a pair $(1,?)$ and $s'=s$ otherwise

Type-guided transformation

Rules:

$$\frac{}{\Gamma \vdash e : (1,o)}$$

$$\frac{\Gamma \vdash K : (s,o)}{\Gamma \vdash a\ K : (s,o)}$$

$$\frac{\Gamma \vdash K : \tau}{\Gamma \vdash nd\ K\ L : \tau}$$

$$\frac{\Gamma \vdash L : \tau}{\Gamma \vdash nd\ K\ L : \tau}$$

$$\frac{\tau \in \Gamma(x)}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma[x \to \Psi] \vdash K : (s,\sigma)}{\Gamma \vdash \lambda x.K : (s',\Psi \to \sigma)}$$

where $s'=0$, $s=1$ if Ψ contains a pair $(1,?)$ and $s'=s$ otherwise

$$\frac{\Gamma \vdash K : (s_K,\{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \qquad \Gamma \vdash L : (s_i,\sigma_i) \text{ for each } i}{\Gamma \vdash K\ L : (s_K+s_1+\ldots+s_n,\sigma)}$$

Type-guided transformation

It is not enough to derive types; we need to transform terms (basing on derived types)

We enrich type judgments:

$$\Gamma \vdash M : \tau \Rightarrow N$$

In environment Γ the term M can have type τ and then it should be transformed to term N.

Type-guided transformation

Transformation:

$$\frac{}{\Gamma \vdash e : (1,o) \Rightarrow e} \qquad \frac{\Gamma \vdash K : (s,o) \Rightarrow N}{\Gamma \vdash a\ K : (s,o) \Rightarrow \blacksquare\ a\ N}$$

$$\frac{\Gamma \vdash K : \tau \Rightarrow N}{\Gamma \vdash nd\ K\ L : \tau \Rightarrow N} \qquad \frac{\Gamma \vdash L : \tau \Rightarrow N}{\Gamma \vdash nd\ K\ L : \tau \Rightarrow N}$$

Type-guided transformation

Transformation:

$$\frac{}{\Gamma \vdash e : (1,o) \Rightarrow e} \qquad \frac{\Gamma \vdash K : (s,o) \Rightarrow N}{\Gamma \vdash a\, K : (s,o) \Rightarrow \blacksquare\ a\, N}$$

$$\frac{\Gamma \vdash K : \tau \Rightarrow N}{\Gamma \vdash nd\, K\, L : \tau \Rightarrow N} \qquad \frac{\Gamma \vdash L : \tau \Rightarrow N}{\Gamma \vdash nd\, K\, L : \tau \Rightarrow N}$$

$$\frac{\tau \in \Gamma(x) \qquad ord(x)=0}{\Gamma \vdash x : \tau \Rightarrow \bullet} \qquad \frac{\tau \in \Gamma(x) \qquad ord(x)>0}{\Gamma \vdash x : \tau \Rightarrow x_\tau}$$

Arguments of order 0 disappear!

Type-guided transformation

Transformation:

$$\frac{}{\Gamma \vdash e : (1,o) \Rightarrow e} \qquad \frac{\Gamma \vdash K : (s,o) \Rightarrow N}{\Gamma \vdash a\, K : (s,o) \Rightarrow \blacksquare\; a\, N}$$

$$\frac{\Gamma \vdash K : \tau \Rightarrow N}{\Gamma \vdash nd\, K\, L : \tau \Rightarrow N} \qquad \frac{\Gamma \vdash L : \tau \Rightarrow N}{\Gamma \vdash nd\, K\, L : \tau \Rightarrow N}$$

$$\frac{\tau \in \Gamma(x) \quad ord(x)=0}{\Gamma \vdash x : \tau \Rightarrow \bullet} \qquad \frac{\tau \in \Gamma(x) \quad ord(x)>0}{\Gamma \vdash x : \tau \Rightarrow x_\tau}$$

$$\frac{\Gamma[x \to \Psi] \vdash K : (s,\sigma) \Rightarrow N \quad ord(x)=0}{\Gamma \vdash \lambda x.K : (s',\Psi \to \sigma) \Rightarrow N} \qquad \frac{\Gamma[x \to \Psi] \vdash K : (s,\sigma) \Rightarrow N \quad ord(x)>0}{\Gamma \vdash \lambda x.K : (s',\Psi \to \sigma) \Rightarrow \lambda x_{\tau_1}.\cdots.\lambda x_{\tau_n}.N}$$

where $\Psi=\{\tau_1,\ldots,\tau_n\}$ and $s'=0$, $s=1$ if $\tau_i=(1,?)$ for some i, and $s'=s$ otherwise

Arguments of order 0 disappear!

Type-guided transformation

Transformation:

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \rightarrow \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\,L : (s_K + s_1 + \ldots + s_n, \sigma) \Rightarrow N\, M_1 \ldots M_n}\; ord(x)>0$$

Type-guided transformation

Transformation:

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\, L : (s_K + s_1 + \ldots + s_n, \sigma) \Rightarrow N\, M_1 \ldots M_n}\; ord(x)>0$$

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\, L : (s_K + s_1 + \ldots + s_n, \sigma) \Rightarrow N}\; s_1+\ldots+s_n=0,\ ord(x)=0$$

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\, L : (s_K + s_1 + \ldots + s_n, \sigma) \Rightarrow N\, M_j}\; s_j=1,\ ord(x)=0$$

Type-guided transformation

Transformation:

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\,L : (s_K+s_1+\ldots+s_n,\sigma) \Rightarrow N\, M_1 \ldots M_n}\ ord(x)>0$$

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\,L : (s_K+s_1+\ldots+s_n,\sigma) \Rightarrow N}\ s_1+\ldots+s_n=0,\ ord(x)=0$$

$$\frac{\Gamma \vdash K : (s_K, \{(s_1,\sigma_1),\ldots,(s_n,\sigma_n)\} \to \sigma) \Rightarrow N \qquad \Gamma \vdash L : (s_i,\sigma_i) \Rightarrow M_i \text{ for each } i}{\Gamma \vdash K\,L : (s_K+s_1+\ldots+s_n,\sigma) \Rightarrow N\, M_j}\ s_j=1,\ ord(x)=0$$

$$\frac{M_1, \ldots, M_n \text{ are all terms such that } \Gamma \vdash K : \tau \Rightarrow M_i}{\Gamma \vdash K : \tau \Rightarrow nd\ M_1\ (\ldots (nd\ M_{n-1}\ M_n) \ldots)}$$

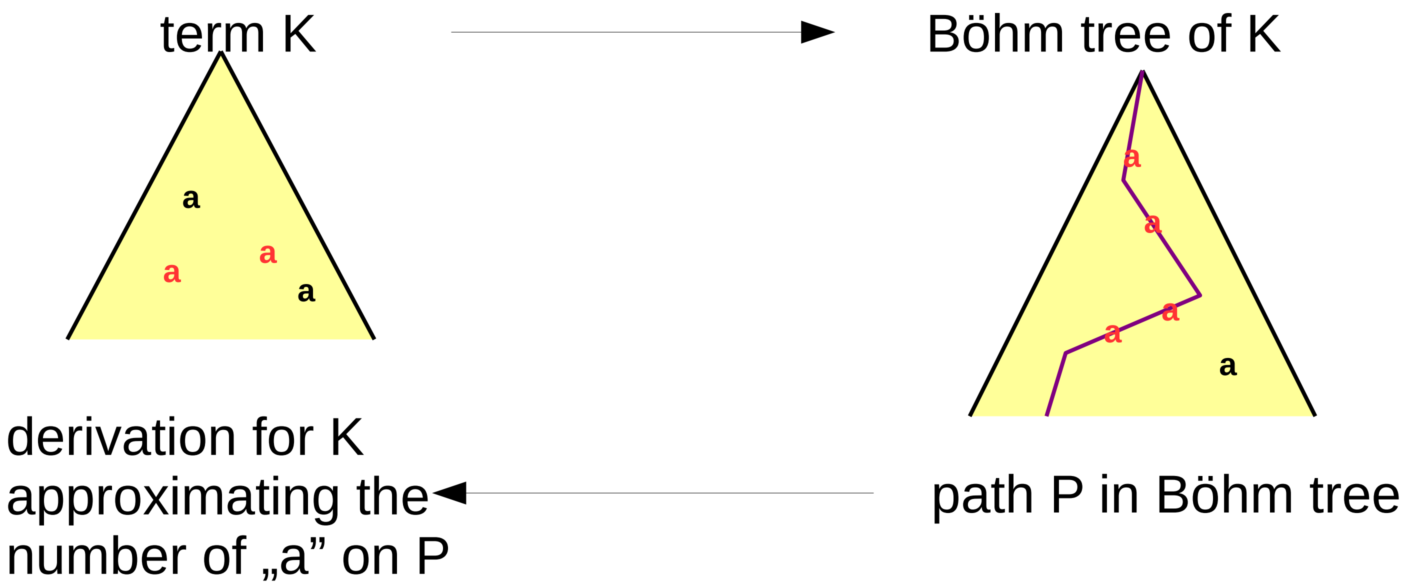
We have seen so far:

- A type system describing behavior of a (co-trivial) alternating tree automaton
- A type system that helps in transforming path-generating lambda-terms to tree-generating lambda-terms of order lower by one.
  - ➔ This allows to solve the unboundedness problem

Next:

- A type system that solves the unboundedness problem directly.

Unboundedness directly via intersection types - idea

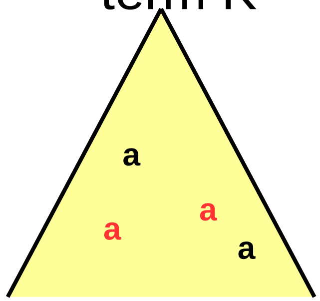


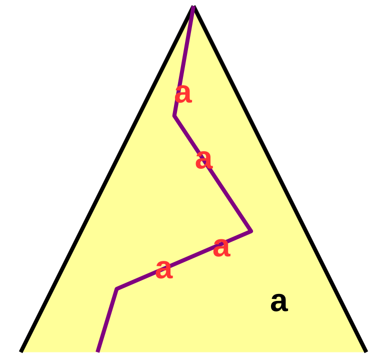
single letter: [P. – ITRS 2016]
multiple letters: [P. – FSTTCS 2017]

Property to describe (unboundedness): In the Böhm tree of K, are there finite paths with arbitrarily many symbols “*a*”?

Unboundedness directly via intersection types - idea

term K → Böhm tree of K



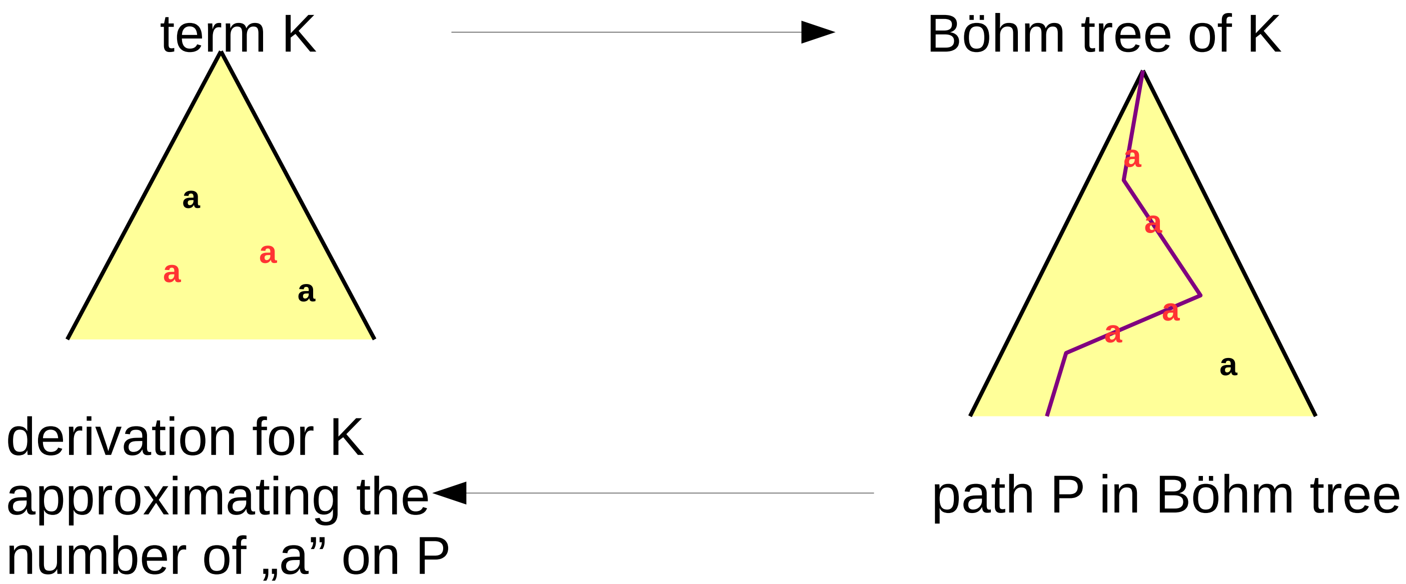


derivation for K approximating the number of „a” on P ← path P in Böhm tree

Easy to say using intersection types:

- which „a” of K will appear in the Böhm tree

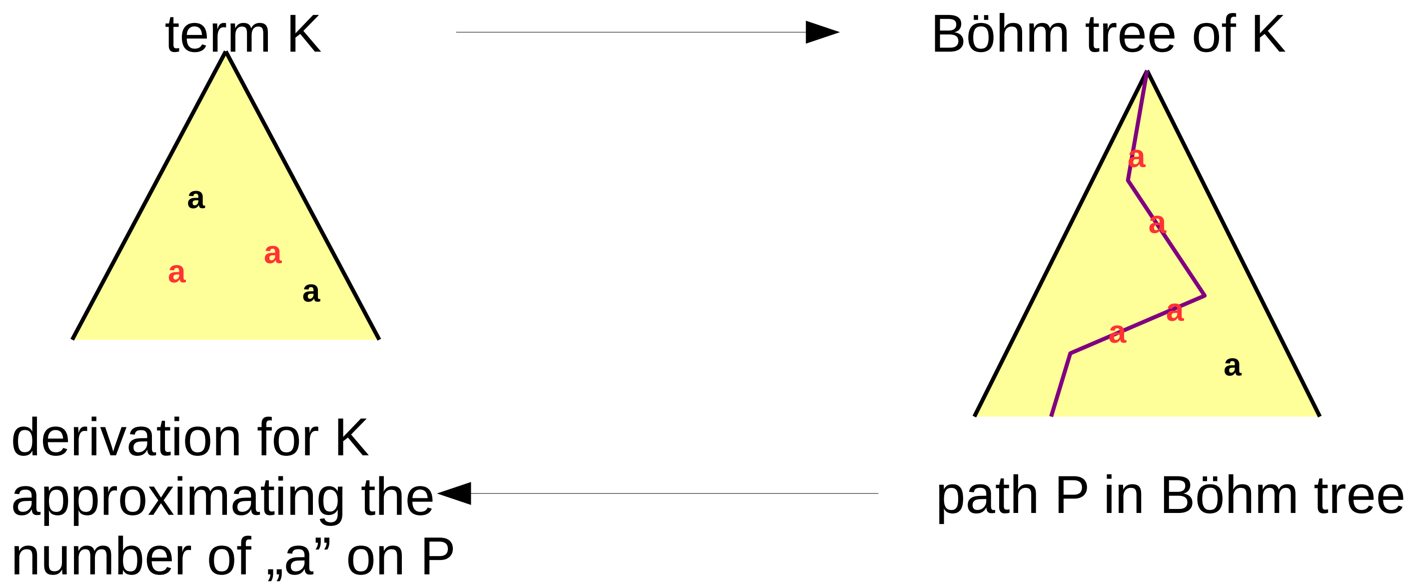
Unboundedness directly via intersection types - idea



Quite easy to say using intersection types:

- which „a” of K will appear on P in the Böhm tree

Unboundedness directly via intersection types - idea



Quite easy to say using intersection types:

- which „a” of K will appear on P in the Böhm tree

Difficulty:

- single „a” of K may result in many „a” on P

$(\lambda\, y.\, y\, (y\, b^{o})).a^{o\to o}$

Idea of solution:

- detect (and count) places where variable containing „a” is duplicated

Intersection types

Solution: type derivations are labeled by flags and markers.

Intersection types refining sort o:

$$\mathcal{T}^{o} = \{ (F, M, o) \}$$

flags used in the derivation

markers used in the derivation

(for each order m we have flags of order m,
and a marker of order m)

Intersection types

Solution: type derivations are labeled by flags and markers.

Intersection types refining sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

$$\mathcal{T}^{\alpha} = \{ (F, M, T_1 \to \ldots \to T_k \to o) \}$$

flags used in the derivation

sets of types refining $\alpha_1, \ldots, \alpha_k$

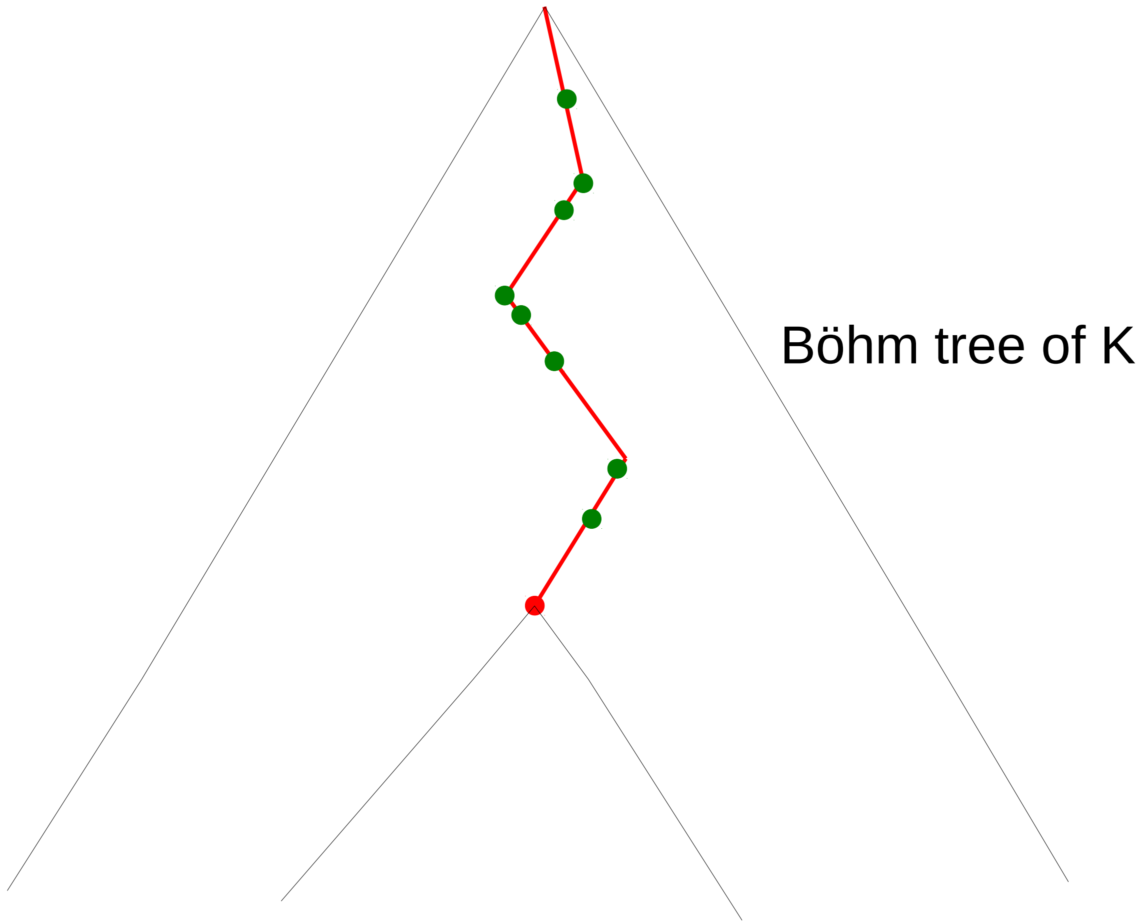
markers used in the derivation

(for each order m we have flags of order m,
and a marker of order m)

Only finite derivations!
No weakening! (every type for an argument have to be used)
For every sort there are only finitely many types refining it!

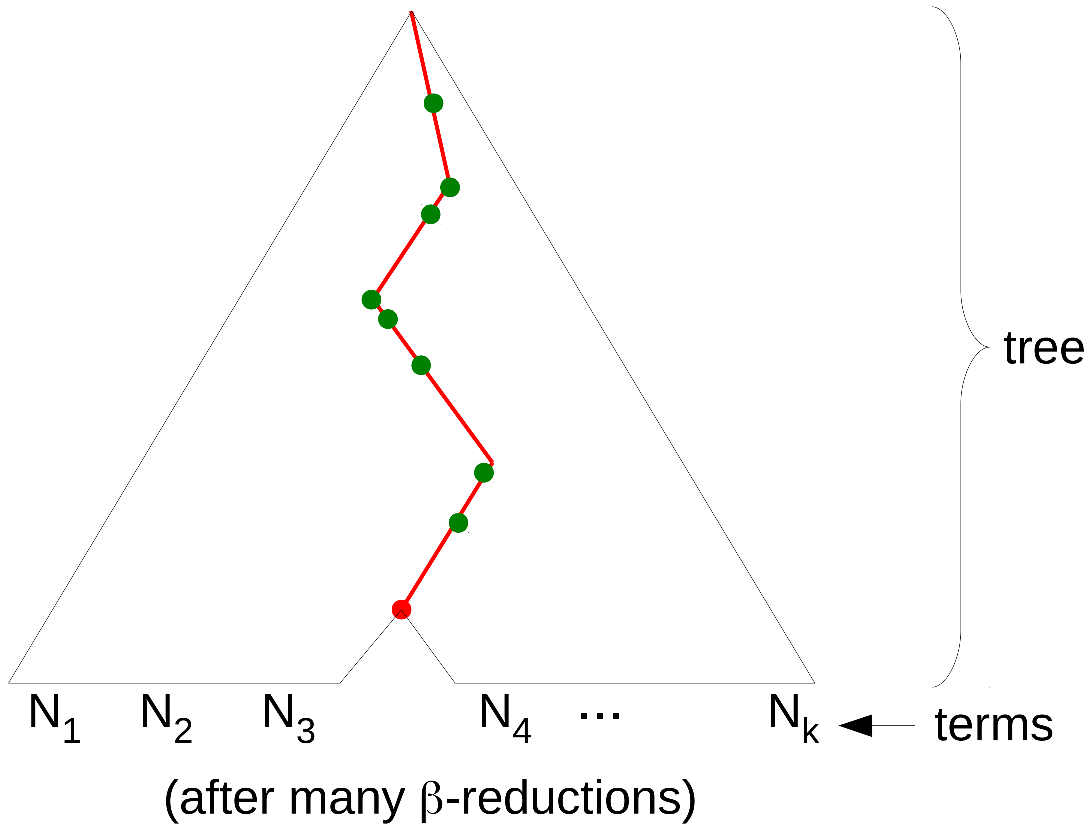
Flags & markers

one marker of order 0 (= end of path)
flags of order 1 (= „a” on the path)



Flags & markers

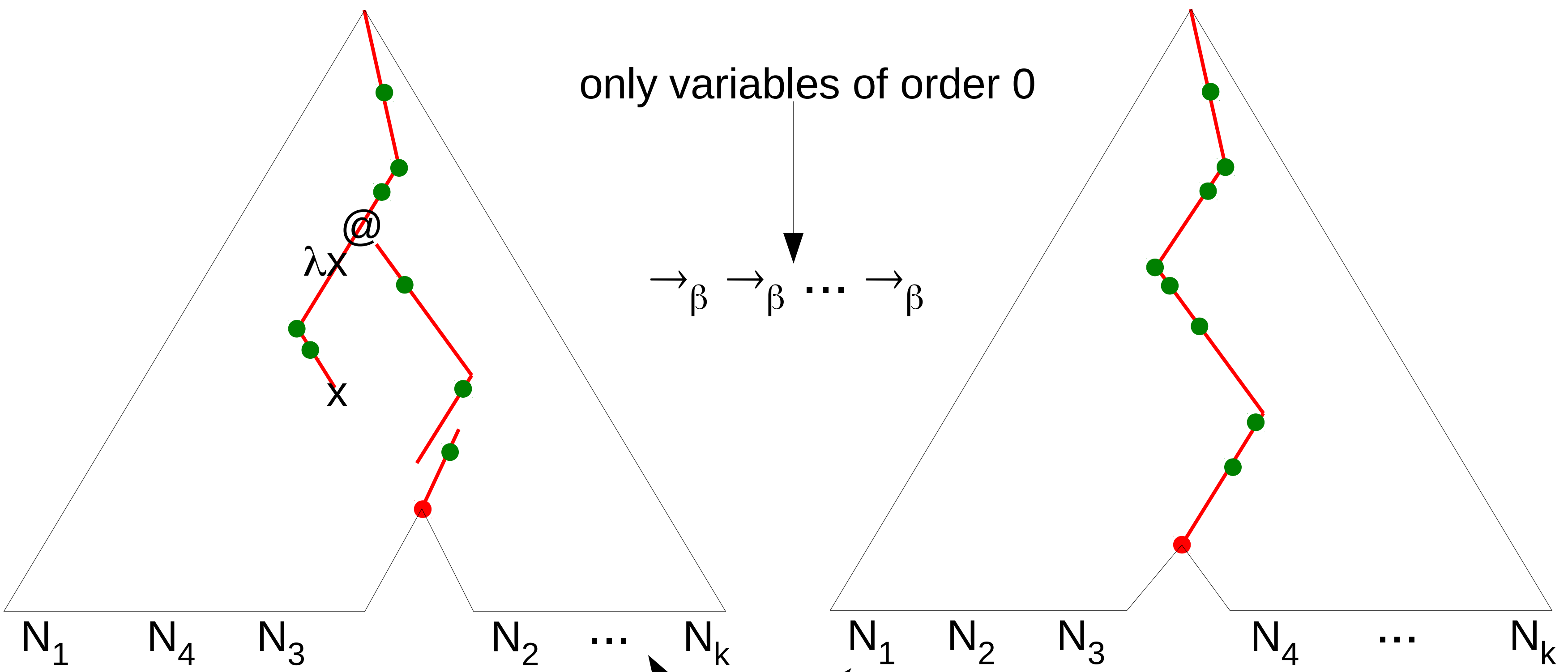
one marker of order 0 (= end of path)
flags of order 1 (= „a” on the path)



Flags & markers

one marker of order 0
flags of order 1

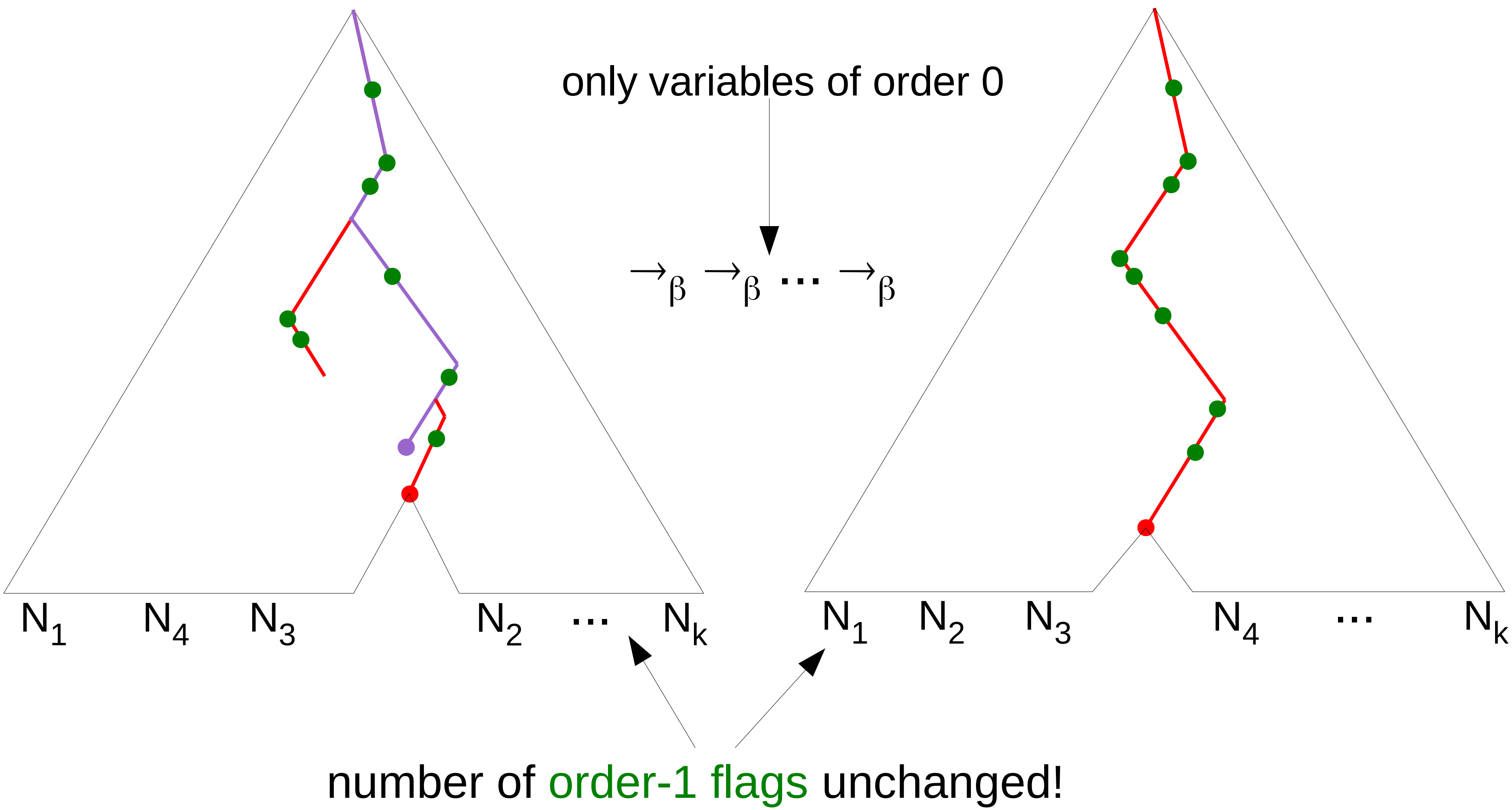
the type system ensures that a variable with marker is used exactly once!



number of order-1 flags unchanged!

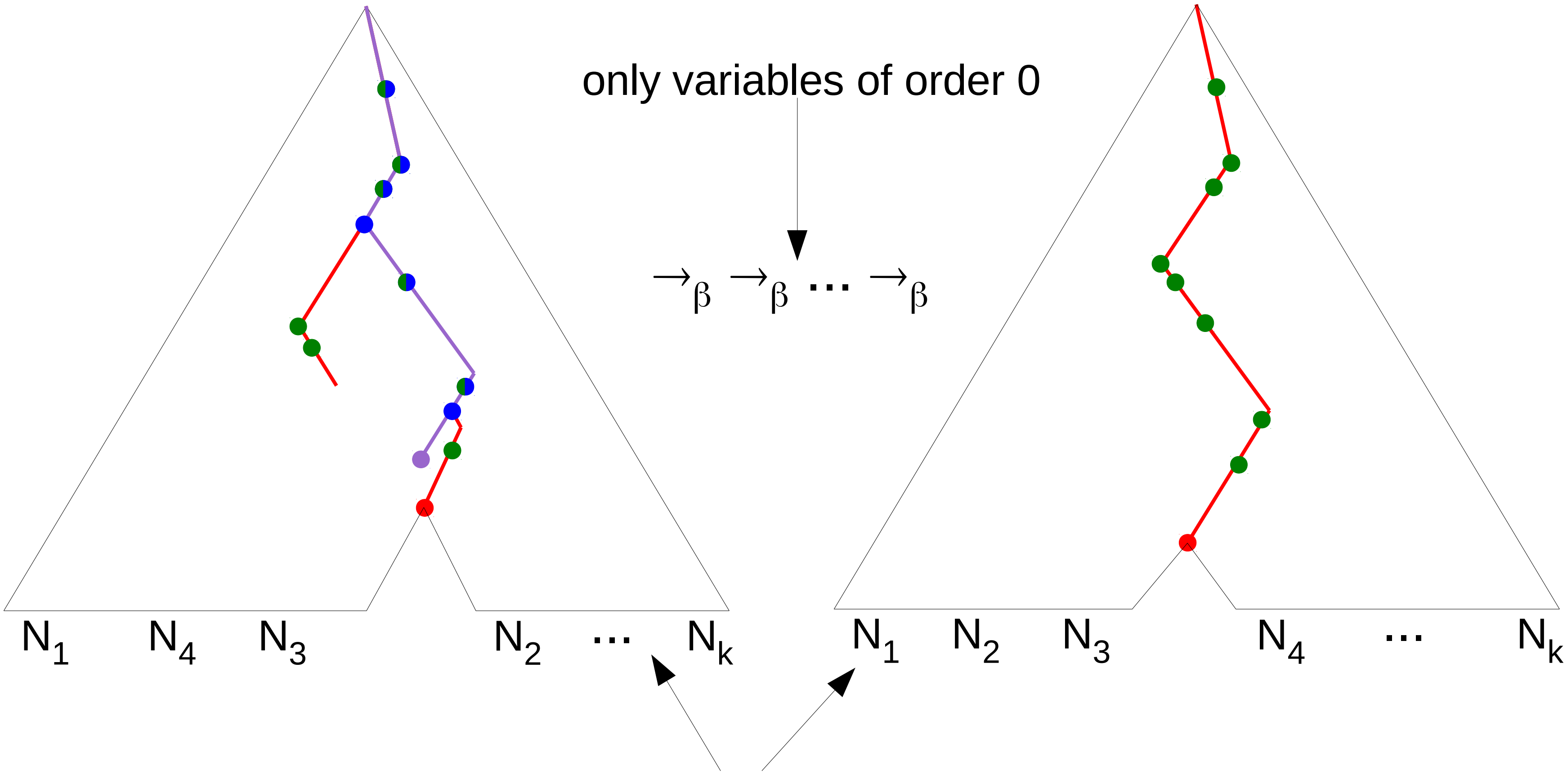
Flags & markers

one marker of order 0
flags of order 1
one marker of order 1



Flags & markers

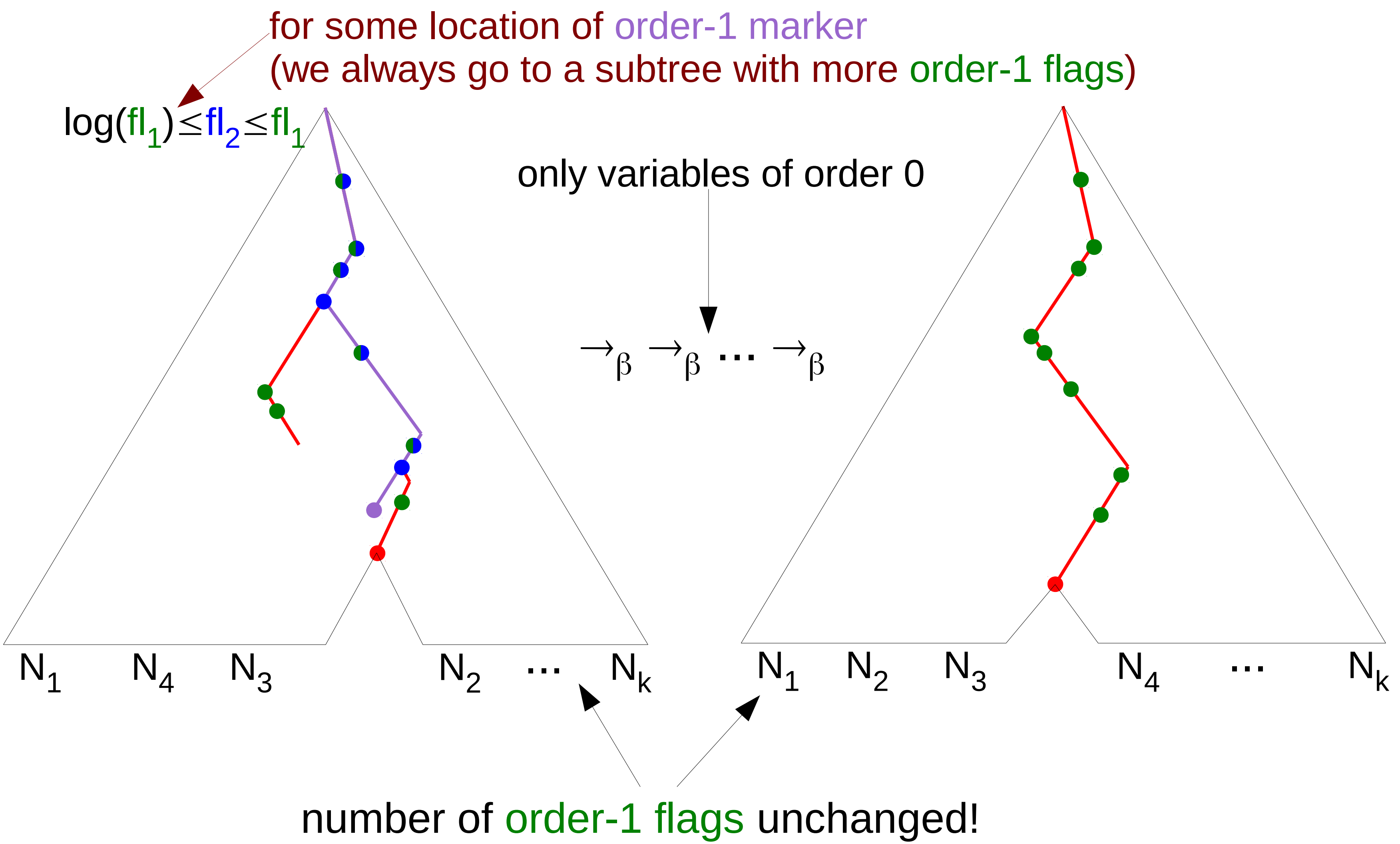
one marker of order 0
flags of order 1
one marker of order 1
flags of order 2 – places on the path to order-1 marker having a descendant with order-1 flag



number of order-1 flags unchanged!

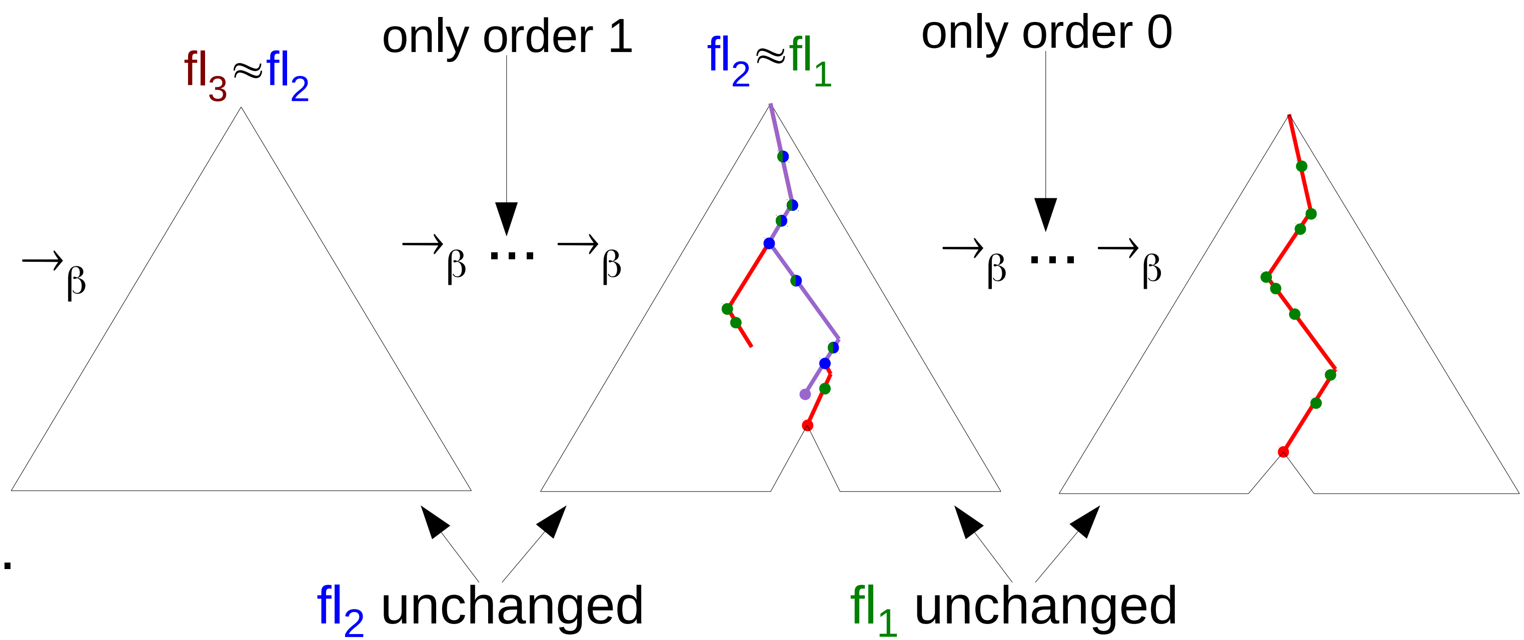
Flags & markers

one marker of order 0
flags of order 1
one marker of order 1
flags of order 2 – places on the path to order-1 marker having a descendant with order-1 flag



Flags & markers

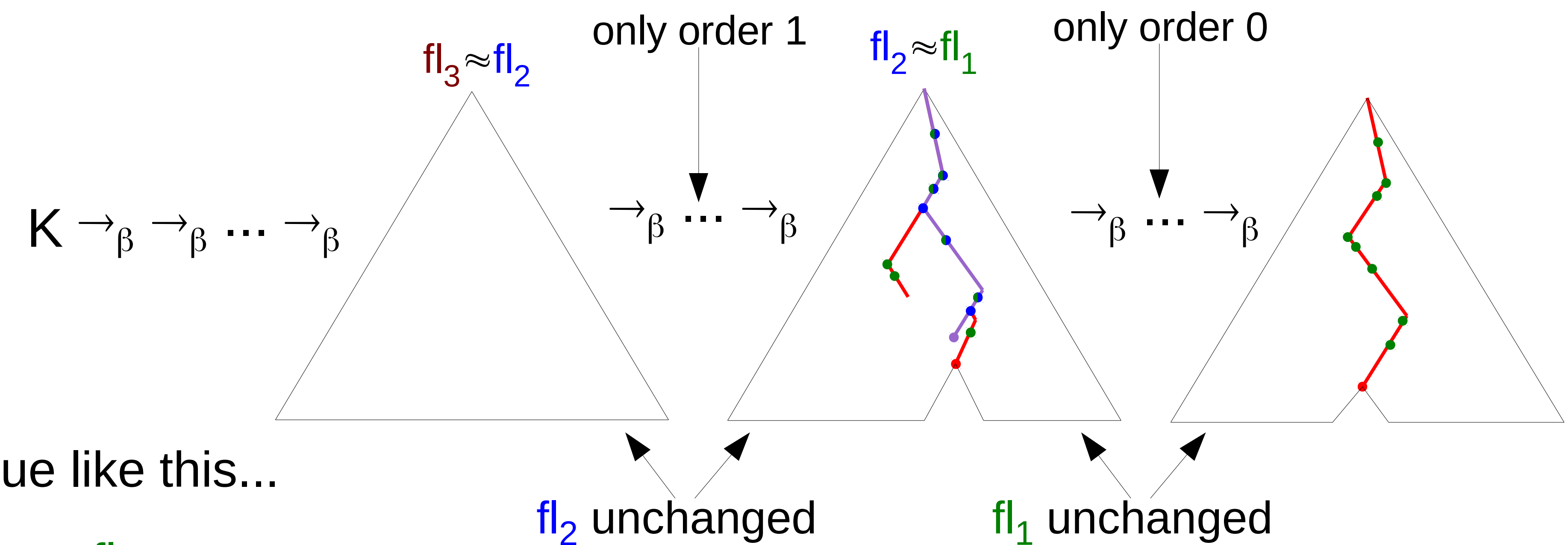
$K \to_\beta \to_\beta \cdots \to_\beta$



continue like this...

$fl_n \approx fl_1$

Flags & markers



continue like this...

$fl_n \approx fl_1$

We put all the flags & markers in derivations for K.
The number of order-n flags approximates the number of „a” on some path in the Böhm tree of K.

there exist derivations for K with arbitrarily many order-n flags $\Longleftrightarrow$ in the Böhm tree of K there exist paths with arbitrarily many „a”

easy to decide

Intersection types

Intersection types refining sort $\alpha = \alpha_1 \to \ldots \to \alpha_k \to o$:

$$\mathcal{T}^{\alpha} = \{ (F, M, T_1 \to \ldots \to T_k \to o) \}$$

flags used in the derivation

sets of types refining $\alpha_1, \ldots, \alpha_k$

markers used in the derivation

(for each order m we have flags of order m,
and a marker of order m)

Type judgments: $\Gamma \vdash^{c} K : \tau$, where c is the number of flag of order n

- Only finite derivations!
- No weakening! (every type for an argument have to be used)
- Some types are idemponent (when no marker is present) – can be used arbitrarily many times
- Some types are not idempotent (when a marker is present) – can be used only once
- But every kind of marker can be placed only in one place.
- In effect, for every sort there are only finitely many types refining it!

Advantages of the approach via intersection types:

- Better complexity
- We can obtain a reflection property:

  Given a lambda-term K (closed, of sort o), we can compute a lambda-term K' such that BT(K') is an enriched version of BT(K) – namely, for every node v of BT(K') we have an additional bit saying whether there are finite paths with arbitrarily many symbols “*a*” starting in v.

Advantages of the approach via intersection types:

- Better complexity
- We can obtain a reflection property:

  Given a lambda-term K (closed, of sort o), we can compute a lambda-term K' such that BT(K') is an enriched version of BT(K) – namely, for every node v of BT(K') we have an additional bit saying whether there are finite paths with arbitrarily many symbols "*a*" starting in v.

- Consequence: We can decide the WMSO+U logic on those Bohm trees – given a sentence ϕ of WMSO+U, and a lambda-term K (closed, of sort o), we can decide whether ϕ holds in BT(K).

  [P. – STACS 2018]

MSO+U logic (introduced by Bojańczyk in 2004)

MSO+U extends MSO by the following „U” quantifier:

$$UX.\phi(X)$$

$\phi(X)$ holds for sets of arbitrarily large size

$$\forall n \in \mathbb{N}\ \exists X\ (\ n < |X| < \infty\ \wedge\ \phi(X)\ )$$

This construction may be nested inside other quantifiers,
and ϕ may have free variables other than X.

WMSO+U logic (introduced by Bojańczyk in 2004)

MSO+U extends MSO by the following „U” quantifier:

$$UX.\phi(X)$$

$\phi(X)$ holds for sets of arbitrarily large size

$$\forall n \in \mathbb{N}\ \exists X\ (\ n<|X|<\infty\ \wedge\ \phi(X)\ )$$

This construction may be nested inside other quantifiers,
and ϕ may have free variables other than X.

We consider Weak MSO+U (quantification over finite sets only):

$$\exists X \rightarrow \exists_{fin} X$$

e.g. we can express that there exist paths with arbitrarily many „a”

Thank you!