# Homogeneity without Loss of Generality

**Paweł Parys**

University of Warsaw

FSCD 2018

# Higher-order recursion schemes – what is this?

Definition

Recursion schemes = simply-typed lambda-calculus + recursion

In other words:

Recursion schemes = context-free grammars, in which nontermi-
nals can have (typed) arguments

We use them to generate (infinite) trees

# Higher-order recursion schemes – definition

Types:

$$\alpha ::= o \mid \alpha \to \beta$$

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

abbreviation of $o \to ((o \to o) \to o)$

# Higher-order recursion schemes – definition

Types:

$$\alpha ::= o \mid \alpha \to \beta$$

Order:

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

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

# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

order 0 order 2 order 2

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

# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$

Rules:

$S \to A\, b$

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

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

It is required that:

1) types are respected

e.g. $D$ of type $(o\to o)\to o\to o$ is applied to $f$ of type $o\to o$,

$A$ of type $(o\to o)\to o$ is applied to $D\, f$ of type $o\to o$, etc.

2) right side of every rule is of type $o$

# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$

Rules:

$S \quad \to A\ b$

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

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

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

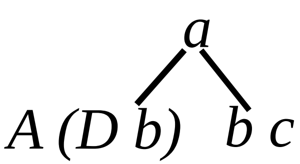
# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$




Rules:

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

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

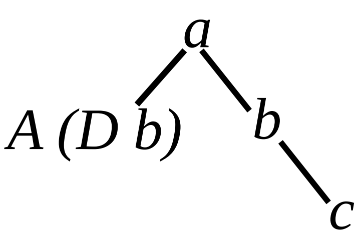
# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$




Rules:

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

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

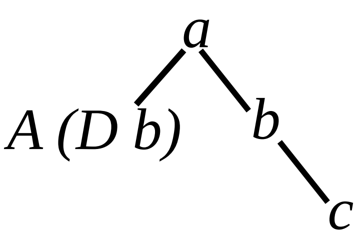
# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$




Rules:

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

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

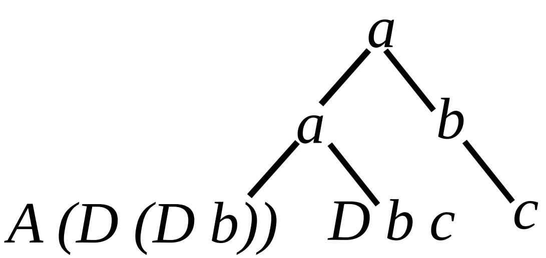
# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o\to o\to o}$ of rank 2, $b^{o\to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o\to o)\to o}$, $D^{(o\to o)\to o\to o}$




Rules:

$S \quad \to A\ b$

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

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

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

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

# Higher-order recursion schemes – definition by example

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

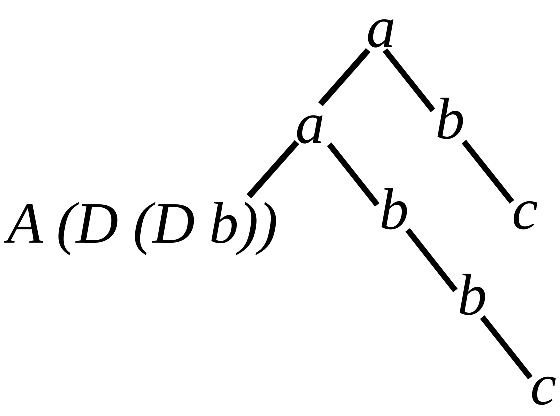
$S^{o}$ (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$

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 – definition by example

Ranked alphabet:

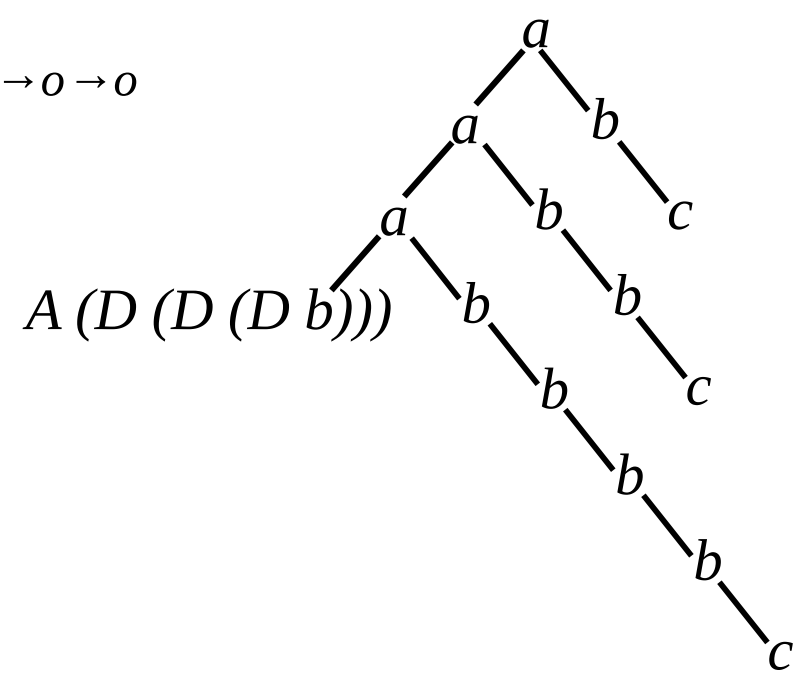
$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

$S^{o}$ (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$

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 – definition by example

Ranked alphabet:

$a^{o \to o \to o}$ of rank 2, $b^{o \to o}$ of rank 1, $c^{o}$ of rank 0

Nonterminals:

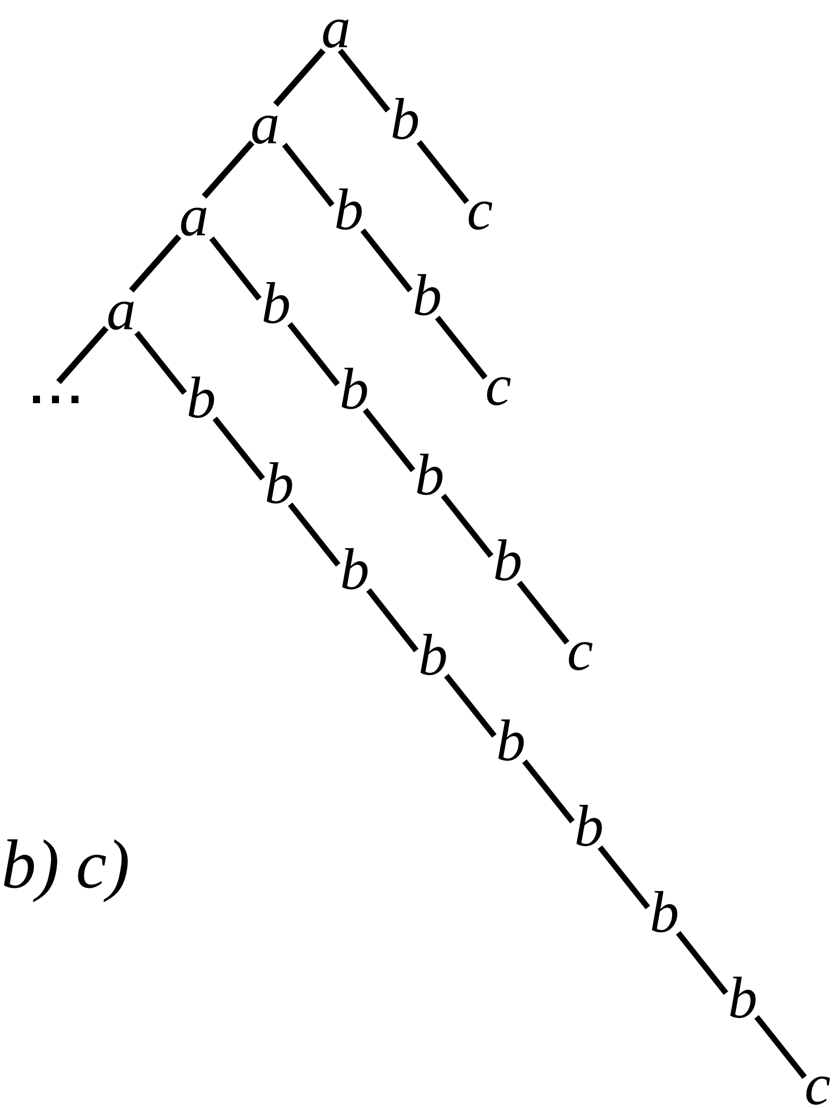
$S^{o}$ (starting), $A^{(o \to o) \to o}$, $D^{(o \to o) \to o \to o}$

Rules:

$S \quad \to A\ b$

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

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




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

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

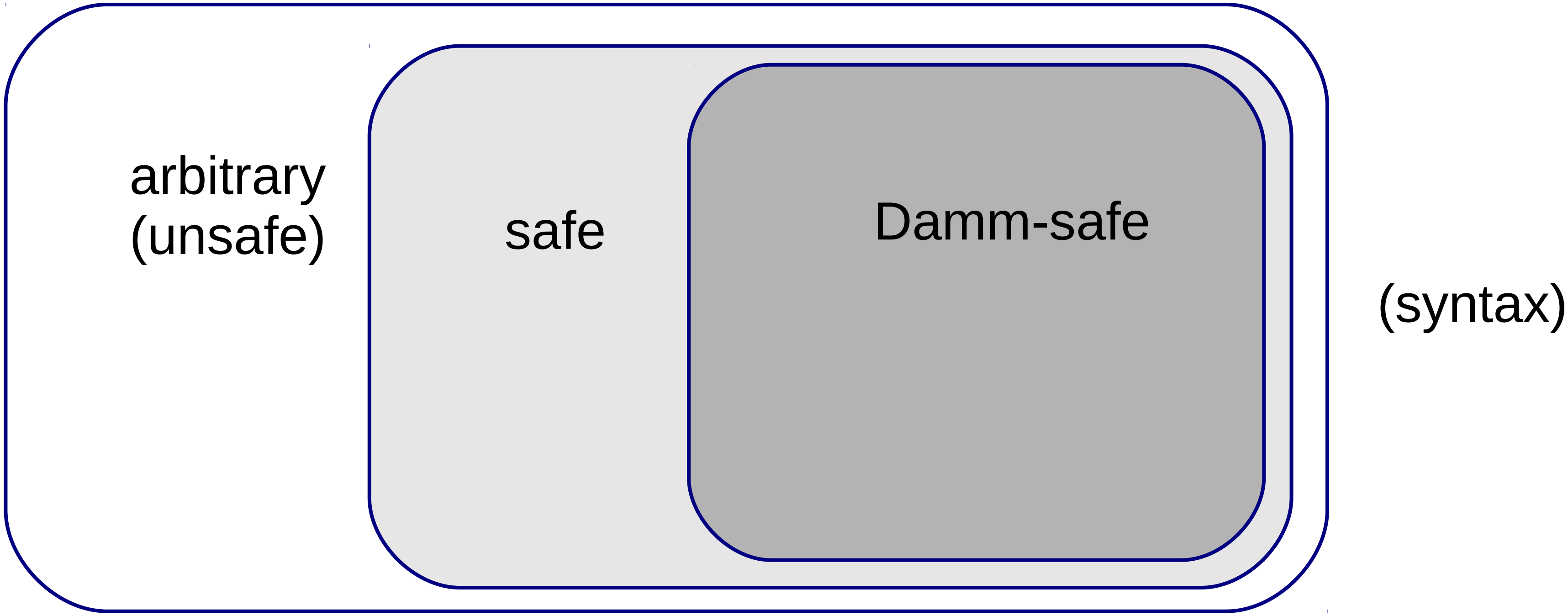
$D\ b\ c \to b\ (b\ c)$

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

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

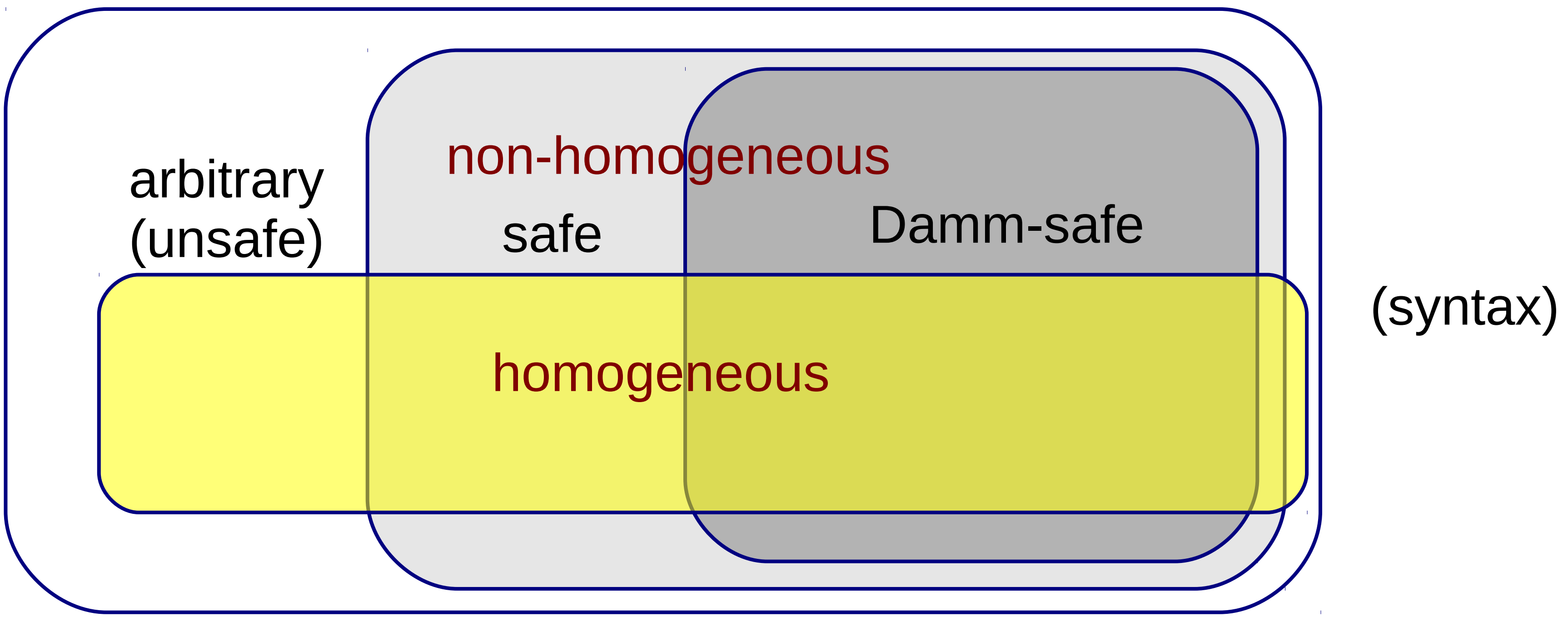
# Restrictions on recursion schemes

Goal of this paper: compare subclasses of recursion schemes
(everything was known here, we only provide new proofs)



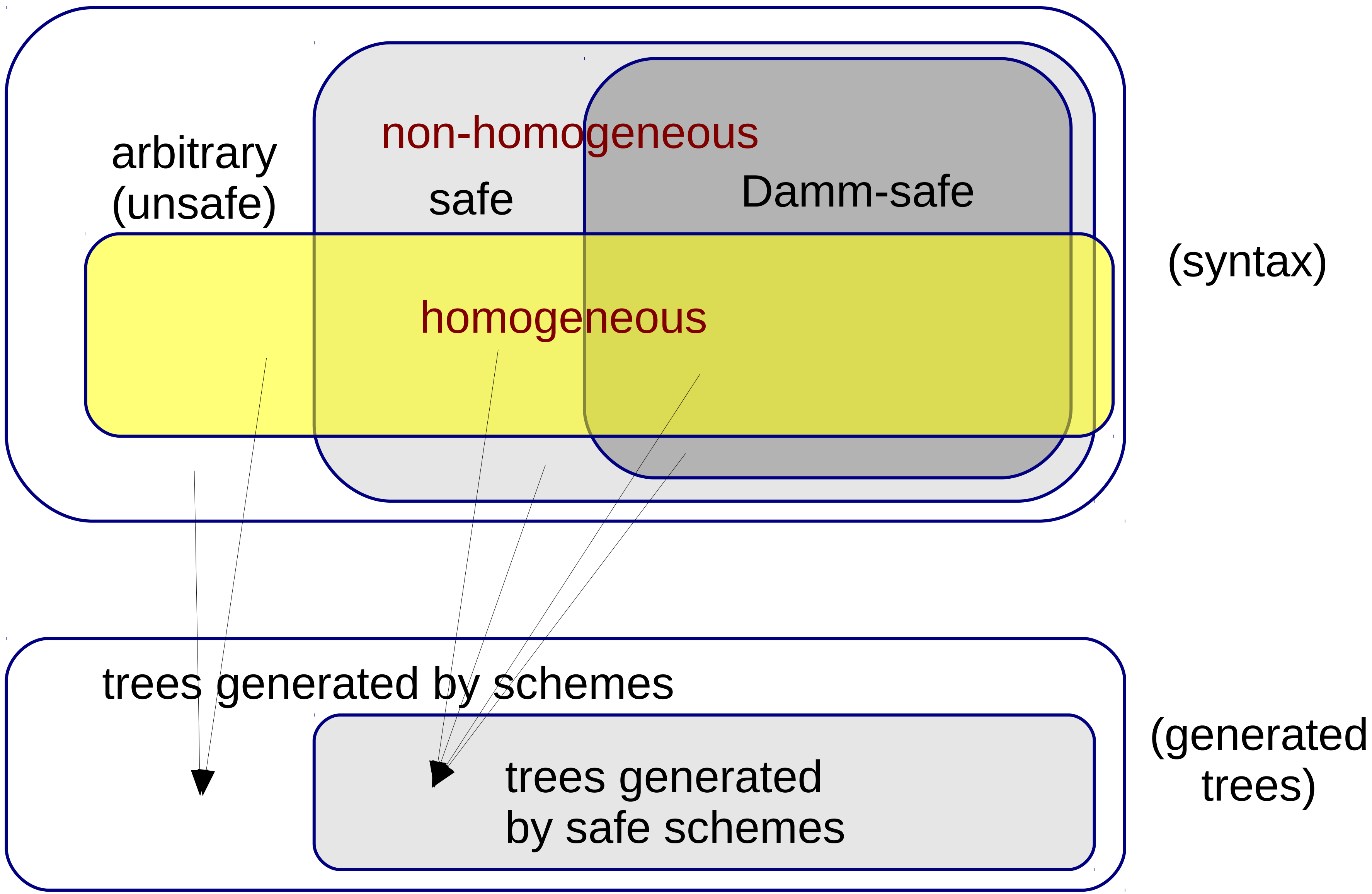
# Restrictions on recursion schemes

Goal of this paper: compare subclasses of recursion schemes
(everything was known here, we only provide new proofs)



# Restrictions on recursion schemes

Goal of this paper: compare subclasses of recursion schemes
(everything was known here, we only provide new proofs)



# Homogeneous schemes

A type $\alpha_1 \to \ldots \to \alpha_k \to o$ is <u>homogeneous</u> if $\mathrm{ord}(\alpha_1) \geq \ldots \geq \mathrm{ord}(\alpha_k)$, and all $\alpha_1, \ldots, \alpha_k$ are homogeneus (defined by induction)

E.g., $(o \to o) \to o \to o$ is homogeneous
$o \to (o \to o) \to o$ is **not** homogeneous

## Homogeneous schemes

A type $\alpha_1 \to \ldots \to \alpha_k \to o$ is homogeneous if $\mathrm{ord}(\alpha_1) \geq \ldots \geq \mathrm{ord}(\alpha_k)$, and all $\alpha_1, \ldots, \alpha_k$ are homogeneus (defined by induction)

E.g., $(o \to o) \to o \to o$ is homogeneous
$o \to (o \to o) \to o$ is **not** homogeneous

A recursion scheme is homogeneous if all nonterminals (and all subterms appearing in rules) have homogeneous types

## Homogeneous schemes

A type $\alpha_1 \to \ldots \to \alpha_k \to o$ is homogeneous if $\mathrm{ord}(\alpha_1) \geq \ldots \geq \mathrm{ord}(\alpha_k)$, and all $\alpha_1, \ldots, \alpha_k$ are homogeneus (defined by induction)

E.g., $(o \to o) \to o \to o$ is homogeneous
$o \to (o \to o) \to o$ is **not** homogeneous

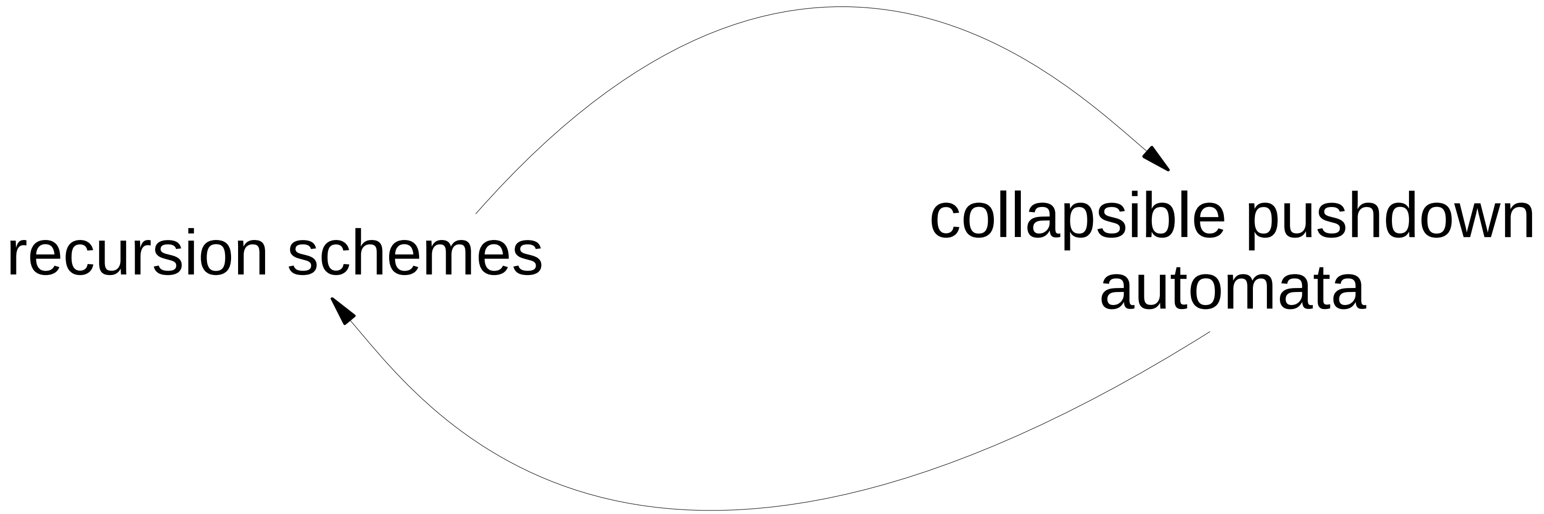
A recursion scheme is homogeneous if all nonterminals (and all subterms appearing in rules) have homogeneous types

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

## Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

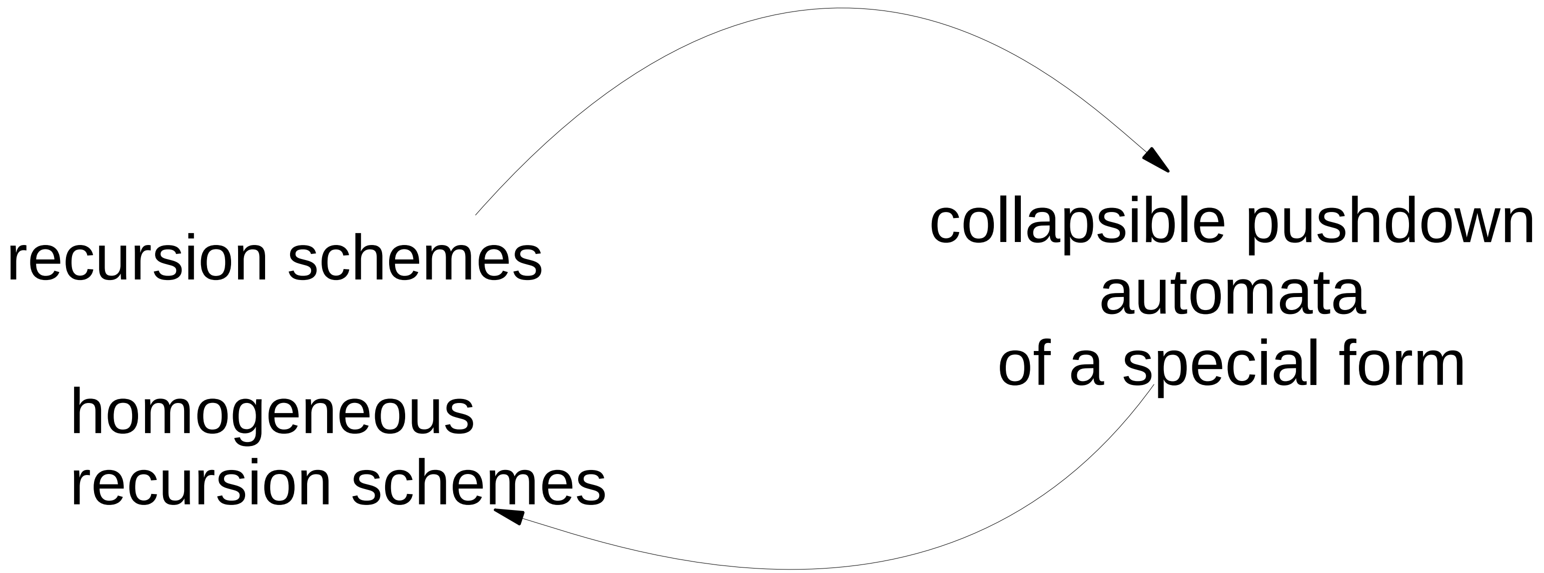
Proof [Broadbent – PhD thesis]



# Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

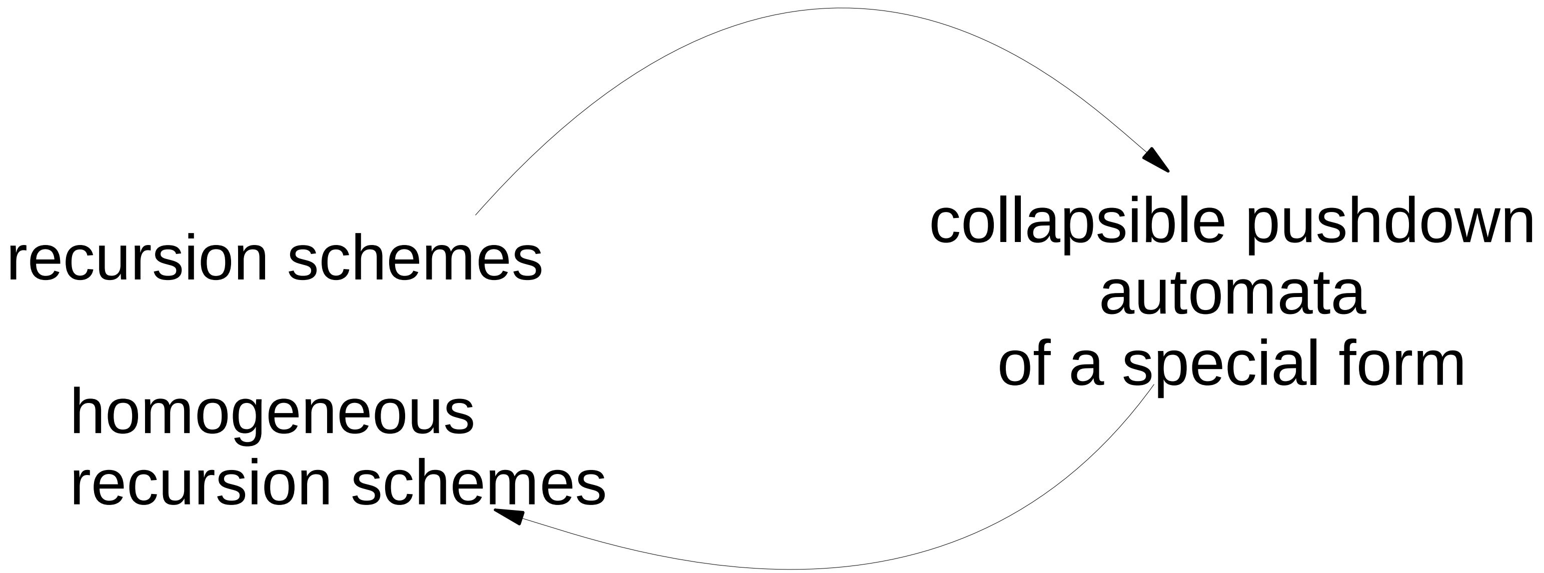
Proof [Broadbent – PhD thesis]



# Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Proof [Broadbent – PhD thesis]




Disadvantages:

- The translations between shcemes and collapsible pushdown automata are complicated itself; observing that the result can be of a special form is even more complicated
- The resulting scheme looks completely unrelated to the original scheme; how the homogeneity was ensured?

# Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

Suppose that we have a rule $D\ x\ f \rightarrow\ ?$, where $ord(x)<ord(f)$

- First idea (invalid) – swap parameters: consider $D'\ f\ x \rightarrow\ ?$
- This causes problems: maybe there are places, where we give only the first argument to $D$, e.g. $E\ (D\ a)$; we cannot replace there $D$ by $D'$

## Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

Suppose that we have a rule $D\ x\ f \to\ ?$, where $ord(x)<ord(f)$

- Second idea (correct) – increase the order of $x$
- We consider a rule $D'\ x'\ f \to\ ?$, where $ord(x')=ord(f)>ord(x)$; $x'$ is a constant function that returns $x$ when given something

# Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

Suppose that we have a rule $D\ x\ f \rightarrow\ ?$, where $ord(x)<ord(f)$

- Second idea (correct) – increase the order of $x$
- We consider a rule $D'\ x'\ f \rightarrow\ ?$, where $ord(x')=ord(f)>ord(x)$;
  $x'$ is a constant function that returns $x$ when given something
- Every use of $x$ is replaced by ($x'$ *something)*
- Every use of *D argument* is replaced by *D' (constant_function argument)*
- *constant_function* is a new nonterminal

# Homogeneous schemes

Theorem 1. For every scheme $G$ one can construct (in logarithmic space) a homogeneous scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

Suppose that we have a rule $D\ x\ f \rightarrow ?$, where $ord(x)<ord(f)$

- Second idea (correct) – increase the order of $x$
- We consider a rule $D'\ x'\ f \rightarrow ?$, where $ord(x')=ord(f)>ord(x)$; $x'$ is a constant function that returns $x$ when given something
- Every use of $x$ is replaced by ($x'$ *something)*
- Every use of $D$ *argument* is replaced by $D'$ *(constant_function argument)*
- *constant_function* is a new nonterminal
- notice that if $ord(x)=ord(f)$-$1$, we have $ord(argument)=ord(something)$, so the sort of *constant_function* is homogeneous
- if $ord(x)<ord(f)$-$1$, it would not be homogeneous; we have to raise the order of $x$ gradually by 1, applying e.g. $constant\_function_1\ (constant\_function_2\ (constant\_function_3\ argument))$

## Safe schemes

A modern definition:

- variables, constants, nonterminals are safe
- an application $M=K\,L_1 \ldots L_n$ is <u>safe</u> if $ord(x) \geq ord(M)$ for all free variables $x$ of $M$, and all $K, L_1, \ldots, L_n$ are safe (defined by induction)

  (notice that subterms $K\,L_1 \ldots L_k$ for $k<n$ need not to be safe)
- roughly: a subterm of order $k$ cannot have free variables of order $<k$

# Safe schemes

A modern definition:

- variables, constants, nonterminals are safe
- an application $M = K\,L_1 \dots L_n$ is safe if $ord(x) \geq ord(M)$ for all free variables $x$ of $M$, and all $K, L_1, \dots, L_n$ are safe (defined by induction)

  (notice that subterms $K\,L_1 \dots L_k$ for $k<n$ need not to be safe)
- roughly: a subterm of order $k$ cannot have free variables of order $<k$

A definition by Damm:

- variables, constants, nonterminals are Damm-safe
- an application $M = K\,L_1 \dots L_n$ is Damm-safe if $ord(L_i) \geq ord(M)$ for $1 \leq i \leq n$, and all $K, L_1, \dots, L_n$ are Damm-safe (defined by induction)

  (again, notice that subterms $K\,L_1 \dots L_k$ for $k<n$ need not to be Damm-safe)
- roughly: if we apply an argument of order $k$, then we need to apply all arguments of order $\geq k$

# Safe schemes

A modern definition:

- variables, constants, nonterminals are safe
- an application $M=K\ L_1\ ...\ L_n$ is <u>safe</u> if $ord(x) \geq ord(M)$ for all free variables $x$ of $M$, and all $K$, $L_1$, ..., $L_n$ are safe (defined by induction)

  (notice that subterms $K\ L_1\ ...\ L_k$ for $k<n$ need not to be safe)
- roughly: a subterm of order $k$ cannot have free variables of order $<k$

A definition by Damm:

- variables, constants, nonterminals are Damm-safe
- an application $M=K\ L_1\ ...\ L_n$ is <u>Damm-safe</u> if $ord(L_i) \geq ord(M)$ for $1 \leq i \leq n$, and all $K$, $L_1$, ..., $L_n$ are Damm-safe (defined by induction)

  (again, notice that subterms $K\ L_1\ ...\ L_k$ for $k<n$ need not to be Damm-safe)
- roughly: if we apply an argument of order $k$, then we need to apply all arguments of order $\geq k$

Easy to prove (by induction):

- Every Damm-safe term is safe.

# Safe schemes

Easy to prove (by induction):

- Every Damm-safe term is safe.

Difficult to prove [P. – LICS 2012]:

- There is a tree generated by an (unsafe) recursion scheme of order 2 that is not generated by any safe recursion scheme.

## Safe schemes

Easy to prove (by induction):

- Every Damm-safe term is safe.

Difficult to prove [P. – LICS 2012]:

- There is a tree generated by an (unsafe) recursion scheme of order 2 that is not generated by any safe recursion scheme.

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

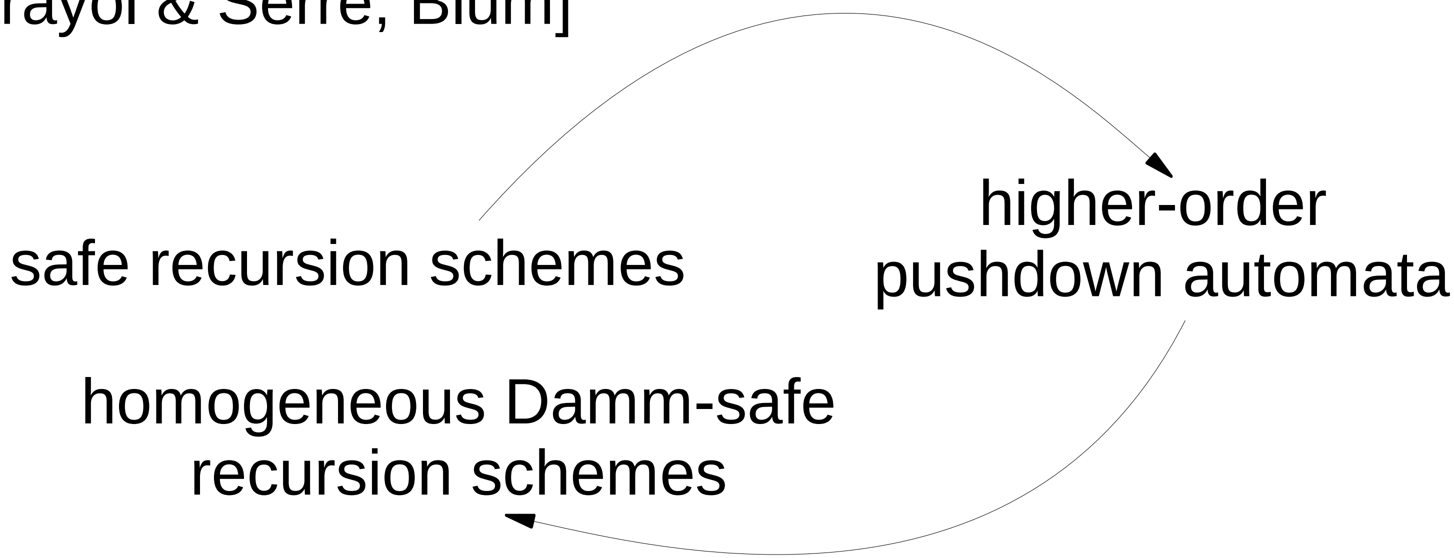
Theorem 3. For every Damm-safe scheme $G$ one can construct (in logarithmic space) a homogeneous Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

# Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Theorem 3. For every Damm-safe scheme $G$ one can construct (in logarithmic space) a homogeneous Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Proof [Carayol & Serre; Blum]




Disadvantages:
- as for theorem 1

## Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Theorem 3. For every Damm-safe scheme $G$ one can construct (in logarithmic space) a homogeneous Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

# Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

- Split every rule of $G$ into multiple simpler rules (create a new nonterminal for every subterm of the right side of every rule of $G$)

# Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

- Split every rule of $G$ into multiple simpler rules (create a new nonterminal for every subterm of the right side of every rule of $G$)

Example:

$$W^{((o\to o)\to o)\to o}\ f^{(o\to o)\to o} \to Y^{((o\to o)\to o)\to o}\ (X^{o\to(o\to o)\to o}\ (Y^{((o\to o)\to o)\to o}\ f))$$

- everything is safe here
- subterm $X\ (Y\ f)$ is not Damm-safe, because $ord(Y\ f)=0<2=ord(X\ (Y\ f))$

## Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

- Split every rule of $G$ into multiple simpler rules (create a new nonterminal for every subterm of the right side of every rule of $G$)

Example:

$$W^{((o\to o)\to o)\to o}\ f^{(o\to o)\to o} \to Y^{((o\to o)\to o)\to o}\ (X^{o\to(o\to o)\to o}\ (Y^{((o\to o)\to o)\to o}\ f))$$

- everything is safe here
- subterm $X\ (Y\ f)$ is not Damm-safe, because $ord(Y\ f)=0<2=ord(X\ (Y\ f))$

We transform this rule to:

$$W^{((o\to o)\to o)\to o}\ f^{(o\to o)\to o} \to Y^{((o\to o)\to o)\to o}\ (S^{((o\to o)\to o)\to(o\to o)\to o}\ f)$$

$$S^{((o\to o)\to o)\to(o\to o)\to o}\ f^{(o\to o)\to o}\ g^{o\to o} \to X^{o\to(o\to o)\to o}\ (Y^{((o\to o)\to o)\to o}\ f)\ g$$

# Safe schemes

Theorem 2. For every safe scheme $G$ one can construct (in logarithmic space) a Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Why is this correct?

After the transformation, right sides are in one of the following forms:

- $x\, y_1 \dots y_n$
- $a\, (X_1\, y_{11} \dots y_{1k_1}) \dots (X_n\, y_{n1} \dots y_{nk_n})$
- $Y\, (X_1\, y_{11} \dots y_{1k_1}) \dots (X_n\, y_{n1} \dots y_{nk_n})$

For subterms $X_i\, y_{i1} \dots y_{ik_i}$ safety = Damm-safety.

The whole term is of order 0, so it is (Damm-)safe.

Recall that:

- $M=K\, L_1 \dots L_n$ is safe if $ord(x) \geq ord(M)$ for all free variables $x$ of $M$, and all $K, L_1, \dots, L_n$ are safe
- $M=K\, L_1 \dots L_n$ is Damm-safe if $ord(L_i) \geq ord(M)$ for $1 \leq i \leq n$, and all $K, L_1, \dots, L_n$ are Damm-safe

## Safe schemes

Theorem 3. For every Damm-safe scheme $G$ one can construct (in logarithmic space) a homogeneous Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

Our proof – simple transformation of terms

Remark: the construction from Theorem 1 does not work: even if we start with a Damm-safe scheme $G$, the resulting homogeneous scheme $H$ is not safe / Damm-safe.
Indeed, a subterm *constant_function argument* is not Damm-safe, because it waits for a second argument of the same order as the first argument. Moreover, if *argument* is a variable, it is not safe.

# Safe schemes

Theorem 3. For every Damm-safe scheme $G$ one can construct (in logarithmic space) a homogeneous Damm-safe scheme $H$ of the same order as $G$, such that $H$ and $G$ generate the same tree.

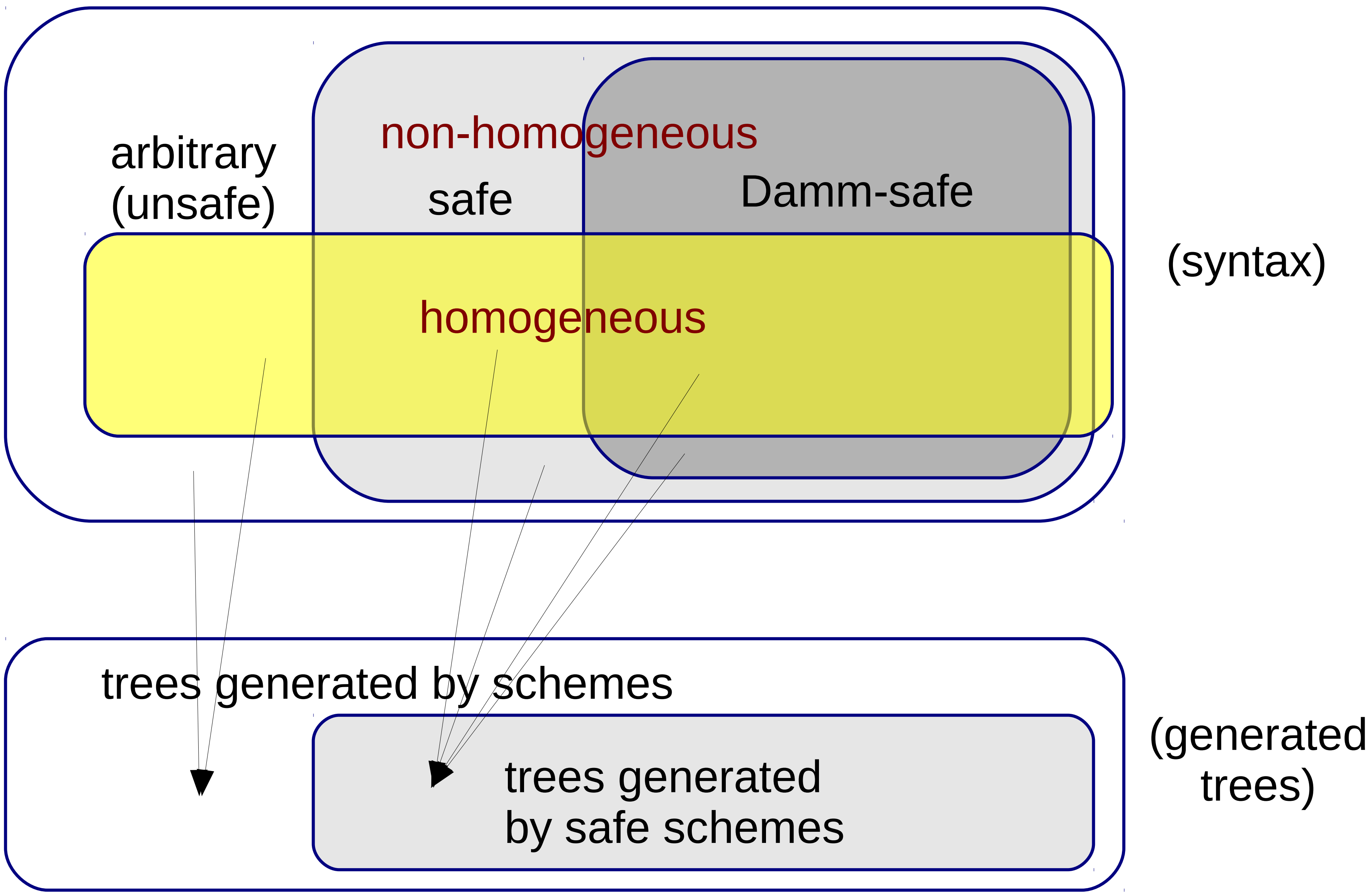
Our proof – simple transformation of terms

Suppose that we have a rule $D\ x\ f \to ?$, where $ord(x)<ord(f)$

- This time we simply swap parameters: we consider $D'\ f\ x \to ?$
- Because our scheme is Damm-safe, whenever we give the first argument $x$ to $D$, we also give the second argument $f$ (a subterm $D$ *something* is not Damm-safe),
- Thus, we can swap the arguments whereever $D$ is used.

- Remark: it is important to assume that the scheme is Damm-safe.
  For a safe scheme, the transformation does not work
  (we have to transform to a Damm-safe scheme first)

# Thank you!



arbitrary
(unsafe)
non-homogeneous
safe
Damm-safe
(syntax)
homogeneous
trees generated by schemes
trees generated
by safe schemes
(generated
trees)