Well-founded Orderings for Operads

Jean-Pierre Jouannaud

Ecole Polytechnique, France

Joint work with Nachum Dershowitz and Jianqi Li

Seoul National University, Seoul, March 4, 2016.

Outline

Outline

Outline

Outline

Outline

From Algebraic Geometry to Algebraic Topology

Algebraic geometry is (originally) the theory of surfaces described by polynomials that take their values in a metric space.

Algebraic topology is the algebraic theory of topological properties of surfaces described by polynomials which take their values in symbolic structures: finite **d**irected, **r**ooted, l**a**belled **g**raphs, which we call **drag**s.

These polynomials live in an algebraic structure called **Operad**.

Rewriting in Operads = Groener basis for Operads
Requires a total well-founded ordering on drags.

Operads in Computer Science

They occur in two areas which have a relationship to algebraic topology, namely: Concurrency and Type theory

To know more about Operads:

> Vladimir Dotsenko and Murray Bremner.
> Algebraic operads: An algorithmic companion.
> Research monograph. Springer Verlag.
> To appear this month. xvii+365pp.

To know more about algebraic topology and concurrency:

> Lisbeth Fajstrup, Martin Raussen and Eric Goubault
> Algebraic topology and concurrency, TCS 357:(1-3), pp241–278, 2006.

To know more about algebraic topology and type theory:

Nicolas Gambino, A Unified Approach to Univalent Foundations and Homotopical Algebra, invited lecture, RTA 2014.

Outline

Examples of drags

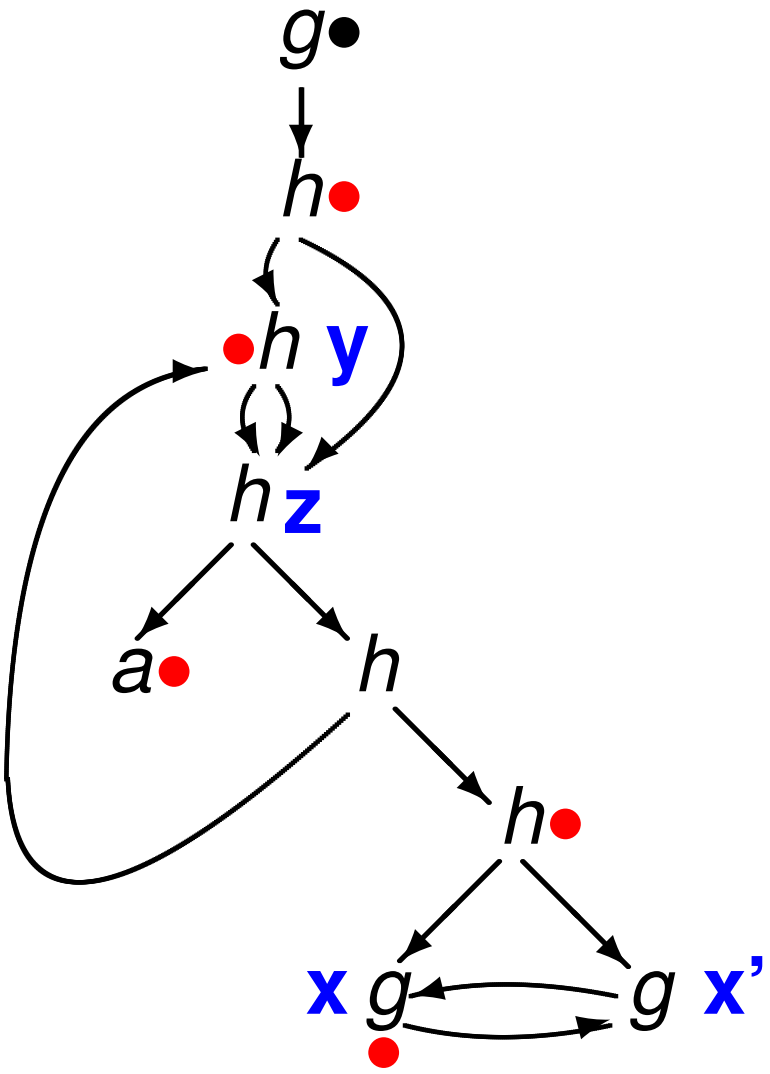


Figure: Two drags, their strict subdrags and shared vertices

Examples of drags

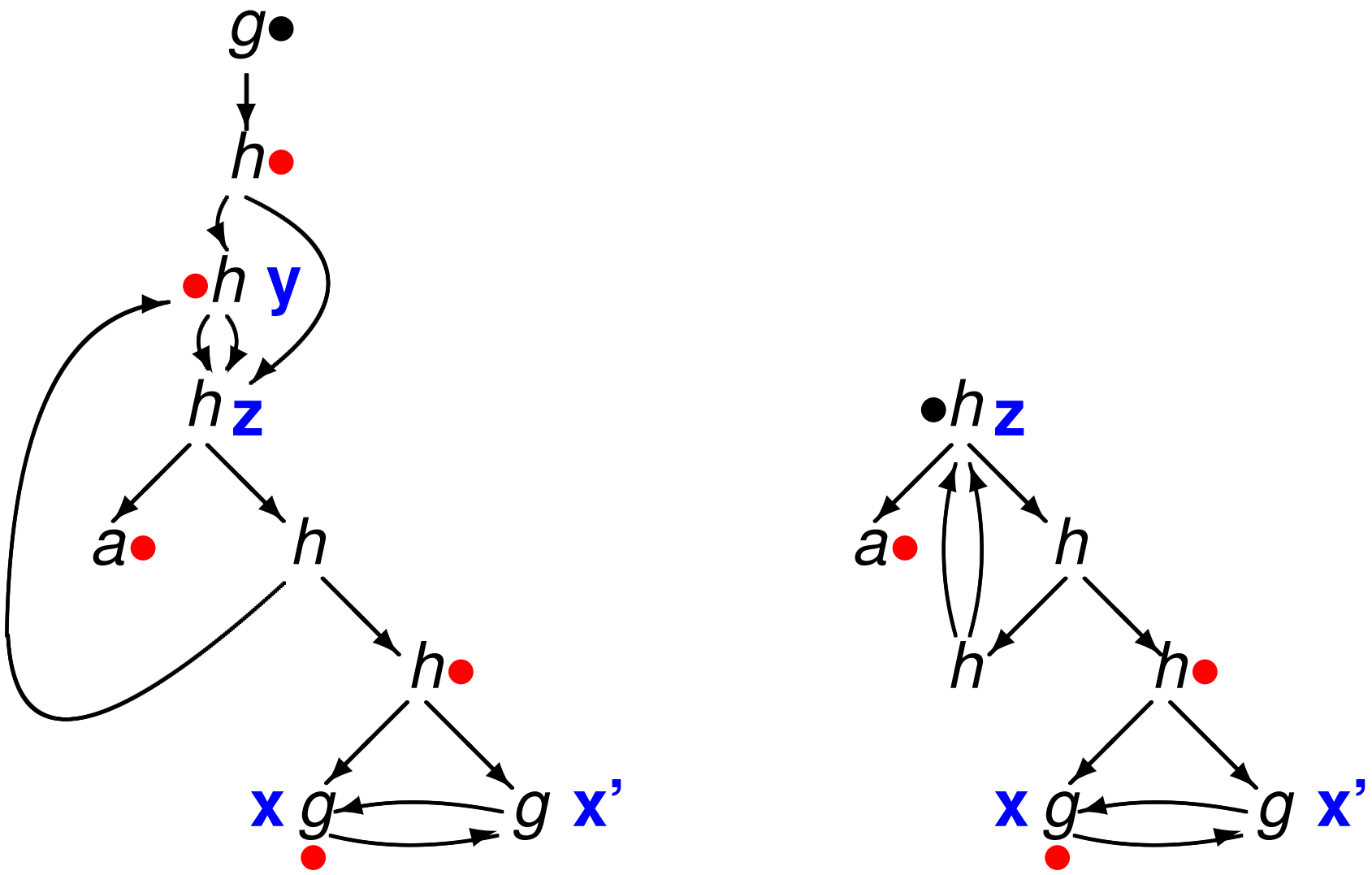


Figure: Two drags, their strict subdrags and shared vertices

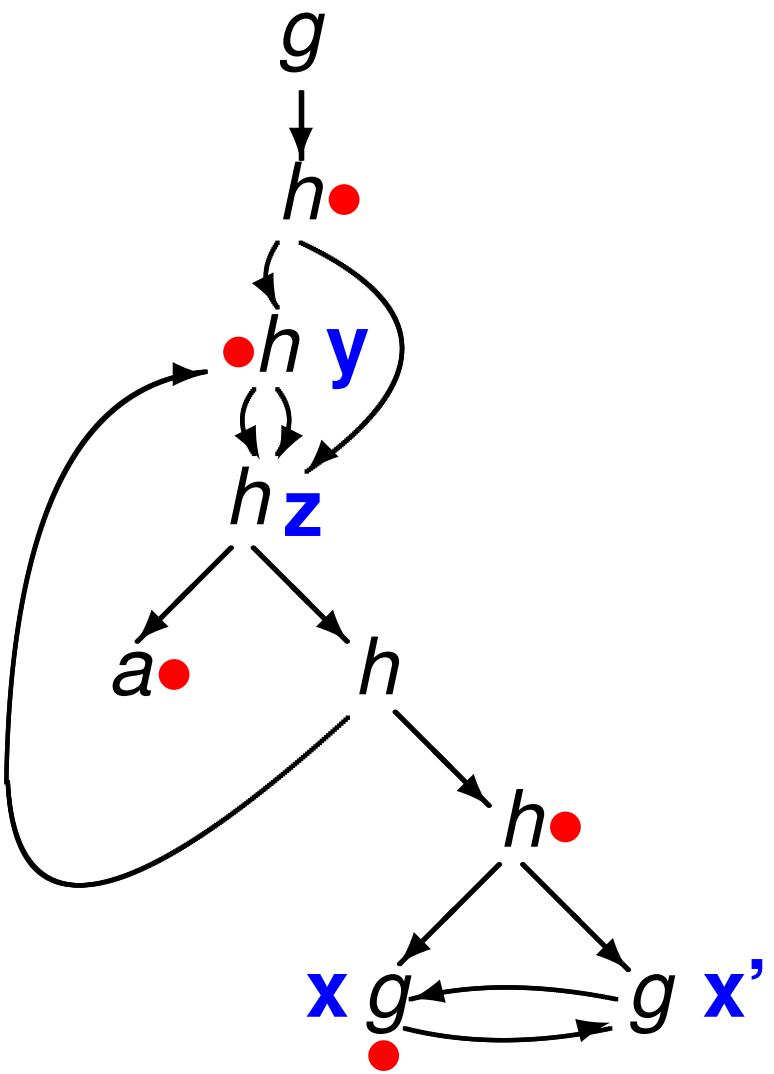
Drag expressions

We use the so-called mu-terms syntax.
mu-terms are expressions with a binding construct to express sharing and back arrows:

$$s, t \;:=\; x \mid f(\overline{s}) \mid [\overline{x = f(\overline{s})}]\, t$$

where
$x \in \mathcal{X}$, $f \in \mathcal{F}$
$\overline{x} \in \mathcal{X}$ is a list of distinct variables,
$\overline{s}$, a list of mu-terms
t, body of the abstraction, is a binder-free term
f is the *root symbol* of $f(\overline{s})$, of $[\overline{y = g(\overline{s})}]f(\overline{s})$, and of $[\overline{x = f(\overline{s})}]x$ such that $x = f(\overline{s}) \in [\overline{x = f(\overline{s})}]$.

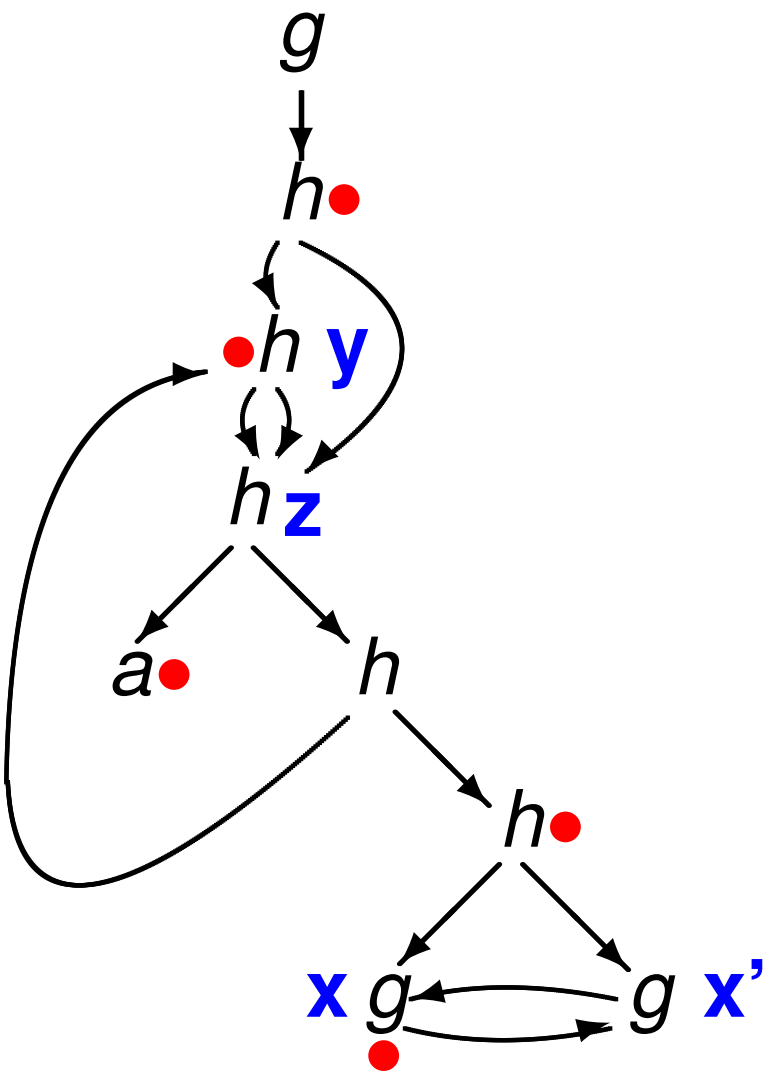
Construction of the drag expressions of a drag (1/6)



[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Construction of the drag expressions of a drag (2/6)

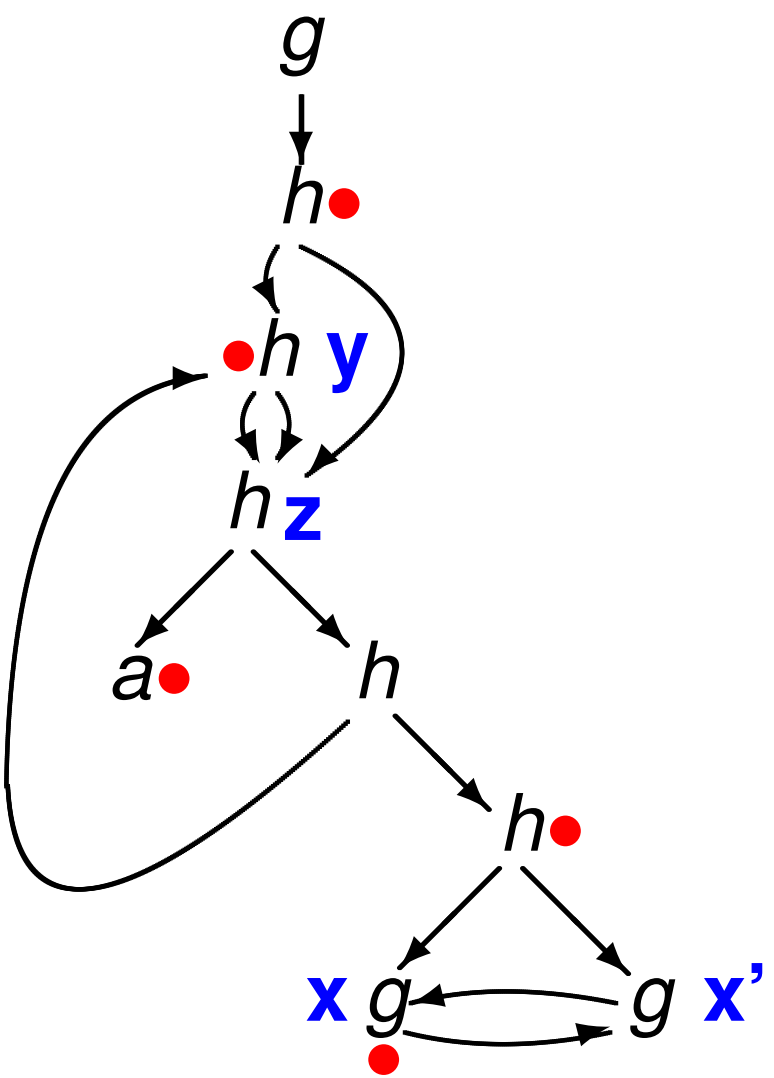


[x=g(x'), x'=g(x)] h(x,x')

[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Calculating drag expressions 3/6



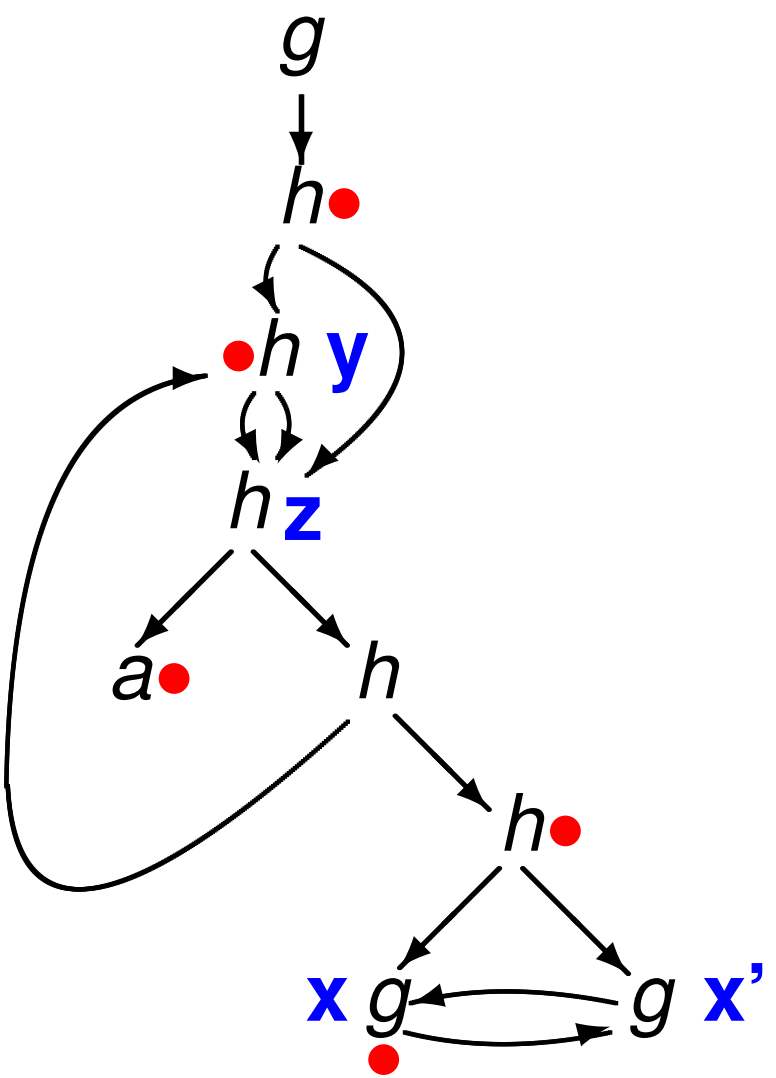
a

[x=g(x'), x'=g(x)] h(x,x')

[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Calculating drag expressions 4/6



[y=h(z,z), z=h(a, h(y,
[x=g(x'), x'=g(x)] h(x,x'))))] y

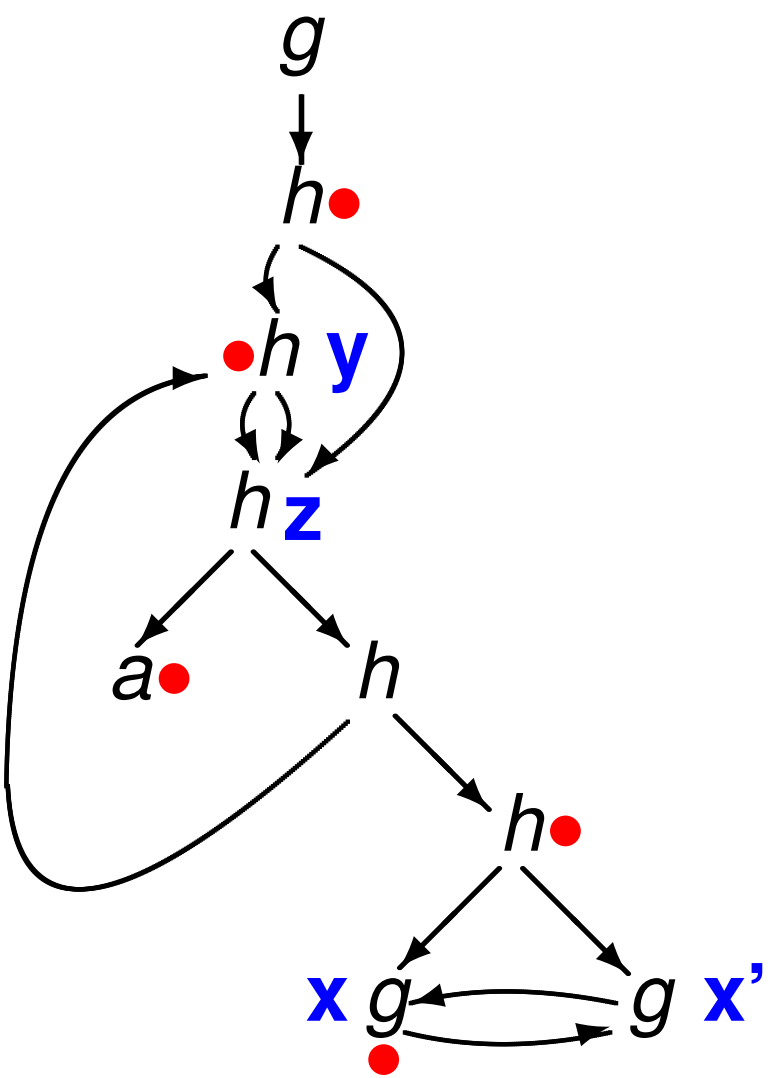
a

[x=g(x'), x'=g(x)] h(x,x')

[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Calculating drag expressions 5/6



[y=. . . , z=. . .] h(y, z)

[y=h(z,z), z=h(a, h(y,
[x=g(x'), x'=g(x)] h(x,x'))))] y

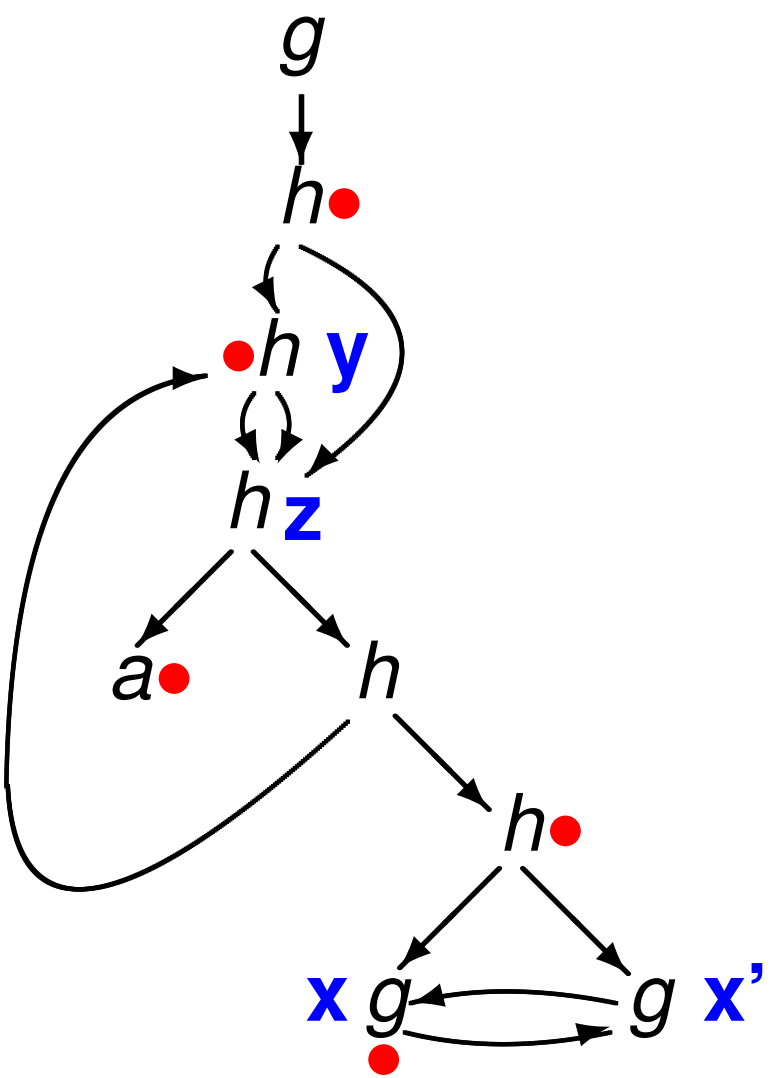
a

[x=g(x'), x'=g(x)] h(x,x')

[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Calculating drag expressions 6/6



g([y=. . . , z. . .] h(y, z))

[y=. . . , z=. . .] h(y, z)

[y=h(z,z), z=h(a, h(y,
[x=g(x'), x'=g(x)] h(x,x'))))] y

a

[x=g(x'), x'=g(x)] h(x,x')

[x=g(x'), x'=g(x)] x and [. . .] x'

Figure: A drag and the construction of its drag expressions

Outline

Building blocks for GPO 1/2

1. Drag expressions are terms with no free variable.
2. Subdrags expressions are drag sub-expressions of a drag-expression.
3. Subterm membership is well-founded.
4. Precedence $\gtrsim$ should satisfy:

   (i) $\dot{>}$ is well-founded

   (ii) $\doteq$ contains variable renaming and permutations in assignements

   (iii) $\dot{\geq}$ is weakly monotonic wrt $\succ$:

   $u \succ v$ implies $f(\overline{s}, u, \overline{t}) \dot{\geq} f(\overline{s}, v, \overline{t})$

Definition

$$t \succ v$$

∇: $\quad \nabla t \succeq v$

$\dot{\succ}$: $\quad t \dot{\succ} v$ and $t \succ \nabla v$

$\doteq$: $\quad t \doteq v$, $t \succ \nabla v$ and $\nabla t \succ_{lex} \nabla v$

where
$\dot{\succeq}$ is the *precedence* quasi-ordering, and
∇ is our notation for the *immediate subdrag expressions*.

$\dot{\succeq}$ plays a key role, by discriminating among drags having different roots.

Example 1/3

Goal:
$t = [x = f(x, x'), x' = g(a)]\, x \succ$
$s = [y = f(f(y, z), z'), z = a, z' = b]\, y.$

We assume that the order $\dot{\geq}$ compares roots, and that $g \dot{\geq} f \dot{\geq} a, b$.

Since s, t have the same root, we recursively compare $t \succ \{a, b\}$ which succeeds since $f \dot{\geq} a, b$.

We now need to compare $\{g(a)\}$ and $\{a, b\}$, which succeeds since $g \dot{\geq} a, b$.

Example 2/3

Goal:

$s = [y = f(f(y, z), z'), z = a, z' = a]\, y \succ$
$t = [x = f(x, x'), x' = g(a)]\, x.$

- Comparing roots is not enough here.
- A term is entirely characterized by its root symbol (also called head), and its list of subterms.
- Similarly, a drag is entirely characterized by its head and its list of subdrags: $\dot{\geq}$ should compared drags heads.
- The head of a drag is obtained by replacing subdrags by free variables standing for its subdrags.

Example 3/3

$[y = f(f(y,z),z'), z = a, z' = a]\, y \succ [x = f(x,x'), x' = g(a)]\, x$
Goal: $[y = f(f(y,z),z')]\, y \succeq [x = f(x,x')]\, x$.

Let $\succeq$ compare first the root function symbol, in the order $f > g > a$, then the total length of the cycles of the head of the terms.

For the first, we get $(f,2)$, and for the second, $(f,1)$, the looser.

Subgoal:
$s = [y = f(f(y,z),z'), z = a, z' = a]\, y \succ g(a)$
Heads are now $(f,2)$ and $(g,0)$.

Subgoal:
$s = [y = f(f(y,z),z'), z = a, z' = a]\, y \succ a$

which yields the head comparison $(f,2) \succ (a,0)$ and succeeds.

Head of a drag

Definition

Given a bijection ξ from drag expressions (modulo permutation and variable renaming) to variables, the head $\widehat{t}$ of a drag t is obtained by replacing each subdrag s by $\xi(s)$, thus $t = \widehat{t}\xi^{-1}$.

Characterization: heads are trees of depth 1 or drags whose root is accessible from all non-variable vertices. Head expressions are described by the grammar:

$$s := f(\overline{y}) \mid u$$

$$u := [\overline{x = g(\overline{v})}]\, x \qquad v, w := y \mid g(\overline{w}) \mid g(\overline{u})$$

$$\text{where } x \in \mathcal{FV}ar(\overline{v}, \overline{w}, \overline{u})$$

Choosing a precedence

The idea is to define the precedence on heads, and let

$$t \dot{\geq} s \text{ iff } \widehat{t} \dot{\geq} \widehat{s}$$

Example: take for $\dot{\geq}$ on heads the lexicographic order on tuples $(l, m, n, \overline{x}, f)$, where l is the total length of elementary cycles, m is the number of elementary cycles, n the number of variables, $\overline{x}$ the multiset of variables, and f its root symbol compared in a precedence on symbols in $\mathcal{F}$.

The discriminating power of heads is of course by far superior to that of roots. But is-it enough for our purpose ?

Equivalences on drags and drags expressions

Equivalence of two drags is graph isomorphism. Our interest is a well-founded order on drags that is compatible with isomorphism of drags, and total in classes of isomorphic drags.

An equivalence on drag expressions can be generated from an equivalence on their heads by letting

$$s \simeq t \text{ iff } \widehat{s} \simeq \widehat{t} \text{ and } \nabla s \simeq \nabla t$$

Lemma

if $\doteq$ contains $\simeq$, then $\succ$ is compatible with $\simeq$

The equivalence on heads should include renaming of their bound variables, permuting variable assignements in assignements, and possibly moving assignements up and down.

Main properties of GPO

1. transitivity
2. well-foundedness
3. compatibility with drag isomorphism if true of $\dot{\geq}$ on heads
4. monotonicity (for drag expressions)
5. totality in $\simeq$-equivalence classes, if $\dot{\geq}$ is total on $\simeq$-equivalence classes of heads, where $\dot{\geq}$ on free variables in heads is defined by comparing their mappings via ξ^{-1}.

Still open: a total precedence on head expressions whose equivalence coincides with equivalence classes modulo isomorphism

Outline

Conclusion

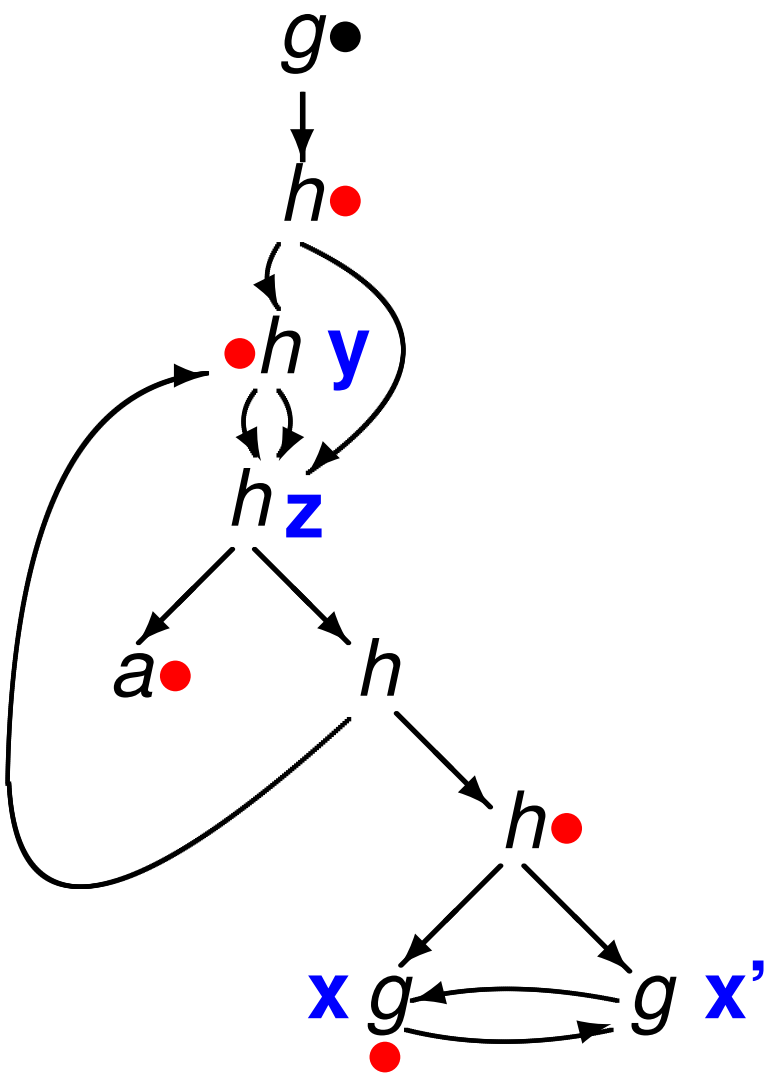
The characterization of drag expressions by head and subterms allow us to define GPO by building in $\succ$ compatibility with $\simeq$ via $\doteq$.

This is also how Rubio defined his *fully syntactic* associative commutative path ordering ACPO.

We are now interested in a general framework for Equational Path Orderings, with LPO, MPO, RPO, ACPO and GPO as instances.

We are also interested in QPO, a path ordering for rational trees (program schemas), for which the semantics of drag expressions, hence their intended equivalence, is completely different.

Unfolding cycles in drags 1/2



Unfolding cycles in drags 2/2