Comparing Unification Algorithms in First-Order Theorem Proving

Kryštof Hoder, Andrei Voronkov
University of Manchester

Saturation-based Theorem Provers

- Try to find a contradiction by generating new clauses according to a set of rules

Binary resolution: $$\frac{\underline{A} \vee C \qquad \underline{\neg B} \vee D}{(C \vee D)\sigma}$$ σ is mgu of A and B

- Need to retrieve all atoms/terms that are unifiable with a query atom/term
  - Often 10^5 or more candidates
- Too much to try one by one, indexing structures are used
- We compared the performance of several unification algorithms inside an indexing structure

Unification Algorithms

- For terms s and t, the algorithm either gives a most general unifier σ (then sσ=tσ), or it fails
- Robinson algorithm (1965, simple, exponential)
- Martelli-Montanari algorithm (1982, almost linear)
- Escalada-Ghallab (1988, almost linear, efficient)
- Paterson-Wegman (1976, inefficient, linear)

Occurs Check

- Cycle detection
  - Avoids situations such as {x -> f(y), y->x}*
- Expensive, in Prolog usually omitted
- Inline occurs check (Robinson algorithm)
  - The cycle detection is done immediately when a variable is bound
  - Only the relevant part is traversed
- Post occurs check (MM, EG)
  - The cycle detection is performed on the whole substitution after the binding part is over

PROB

- The Robinson algorithm's exponential behaviour is rare
- $p(x_0, \quad f(x_0,x_0), \quad x_1,...$
  $p(f(y_0,y_0), \quad y_1, \quad f(y_1,y_1),...$
  - The triangle form of the substitution is polynomial
    - $\{x_0 \text{->} f(y_0,y_0),\ y_1 \text{->} f(x_0,x_0),\ x_1 \text{->} f(y_1,y_1),...\}^*$
    - (the non-triangle form is $\{x_0 \text{->} f(y_0,y_0),\ y_1 \text{->} f(f(y_0,y_0),f(y_0,y_0)),...\}$)
- Without the repeated work during the occurs check and unifying already unified terms, the algorithm runs in polynomial time

Indexing Structures

- Their key role is in simplifying rules
  - Keeping the collection of clauses small
  - Just matching, not unification
- Some more suitable for the *perfect* unifier retrieval
  - Substitution trees (new Vampire, old Fiesta, Spass)
  - Context trees (new Fiesta)
- Some are less
  - Discrimination trees (Waldmeister)
  - Path Indexing

Substitution Trees

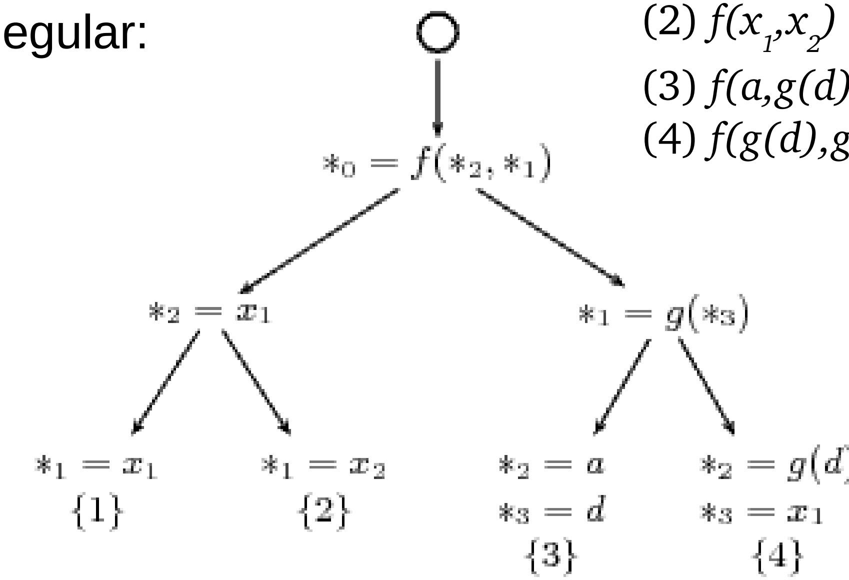
(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

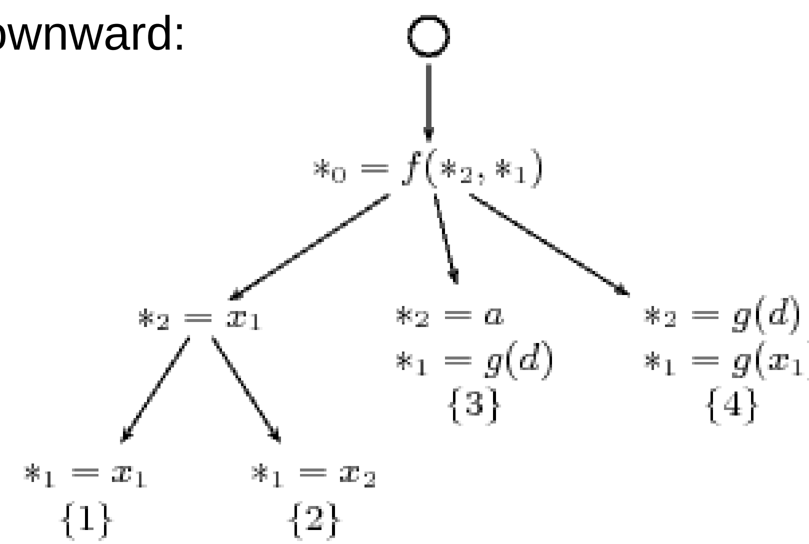
(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$

Regular:



Downward:



- A substitution in each node
- Indexed terms $*_0\sigma$ in the leafs
  - σ is a composition of substitutions on path from the root to the leaf
- Downward subst. trees
  - A newly inserted term has a deterministic position

Unification in Substitution Trees

- Only a simple interface between a substitution tree and a substitution object is necessary
  - tryToExtendToUnify(queryTerm, indexTerm):bool
  - undoLastUnification()
  - getBoundTopSymbol(variable):fnSymbol?
    - not necessary, just allows for an optimization in downward substitution trees

Retrieval from Substitution Trees

Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

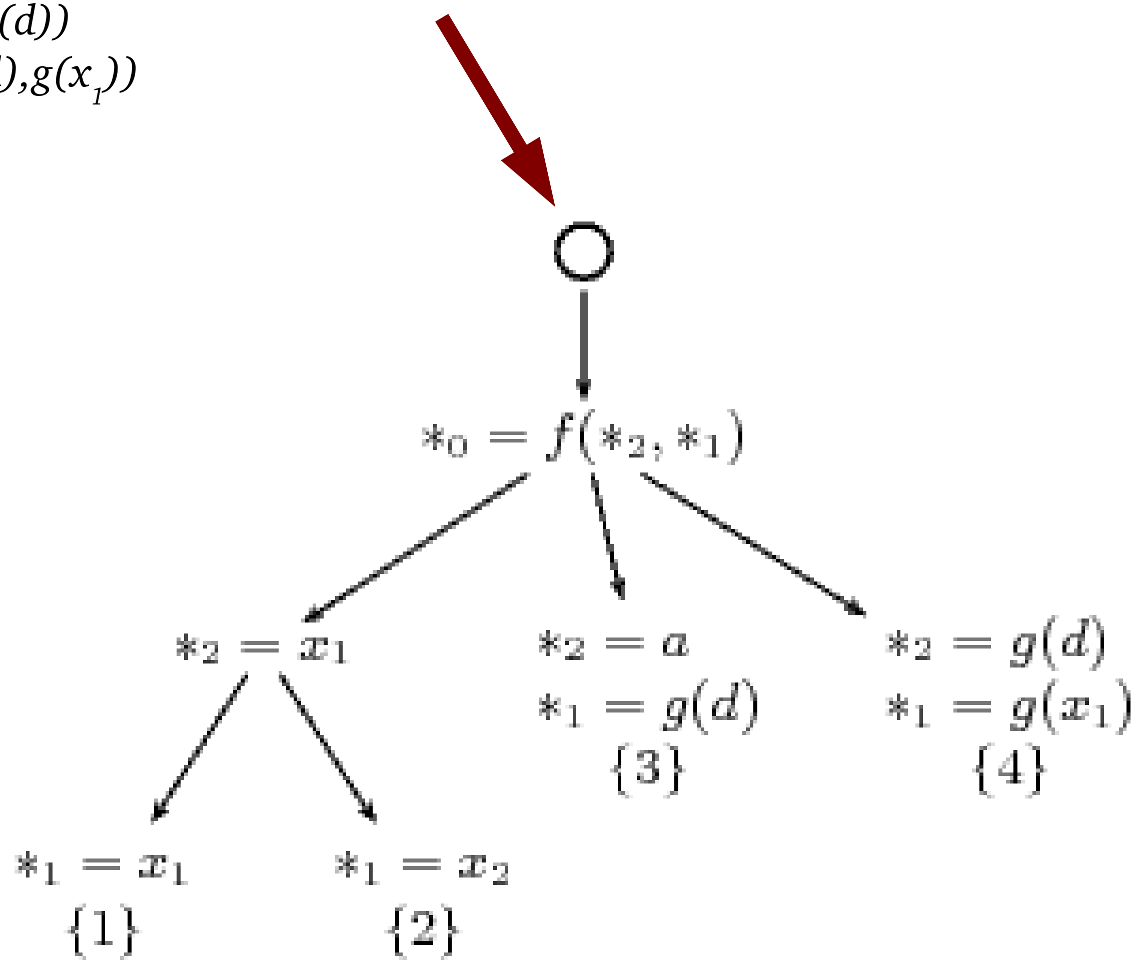
(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$

σ:

$*_0=f(f(a,y_1),y_1)$

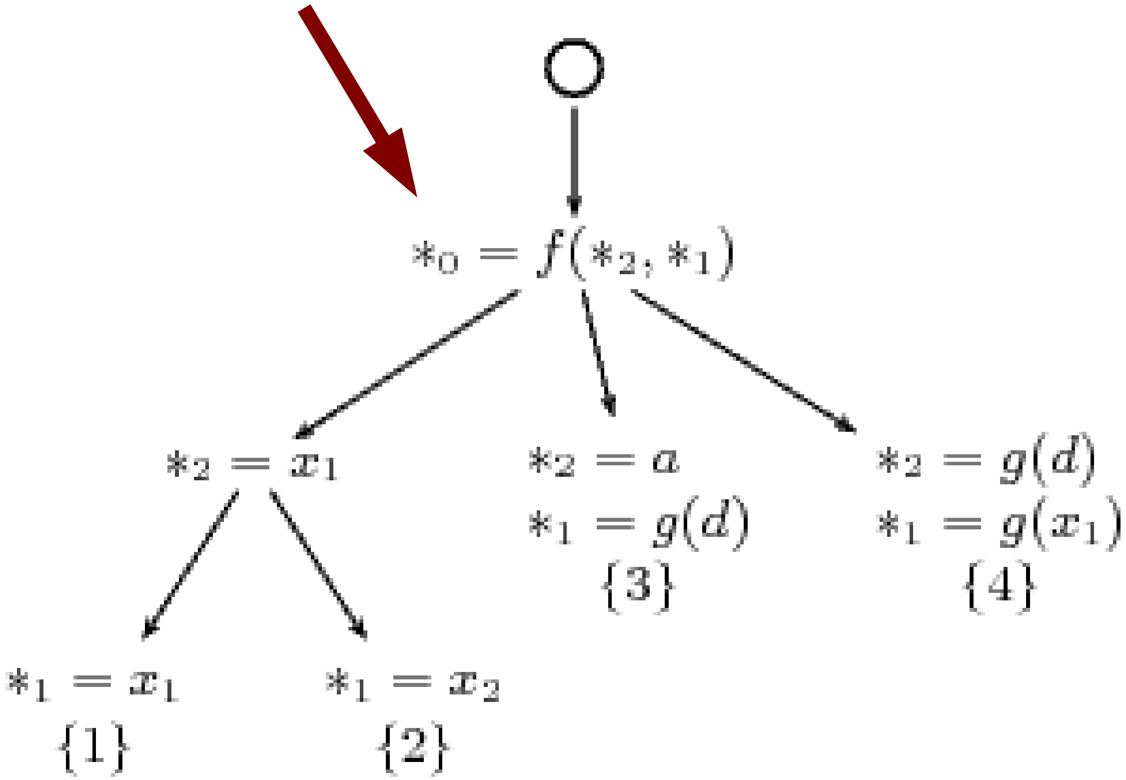


Checked by inline occurs check

Retrieval from Substitution Trees

Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$
(2) $f(x_1,x_2)$
(3) $f(a,g(d))$
(4) $f(g(d),g(x_1))$



σ:

$*_0=f(f(a,y_1),y_1)$

$*_1=y_1$

$*_2=f(a,y_1)$

Checked by inline occurs check

Retrieval from Substitution Trees

Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$

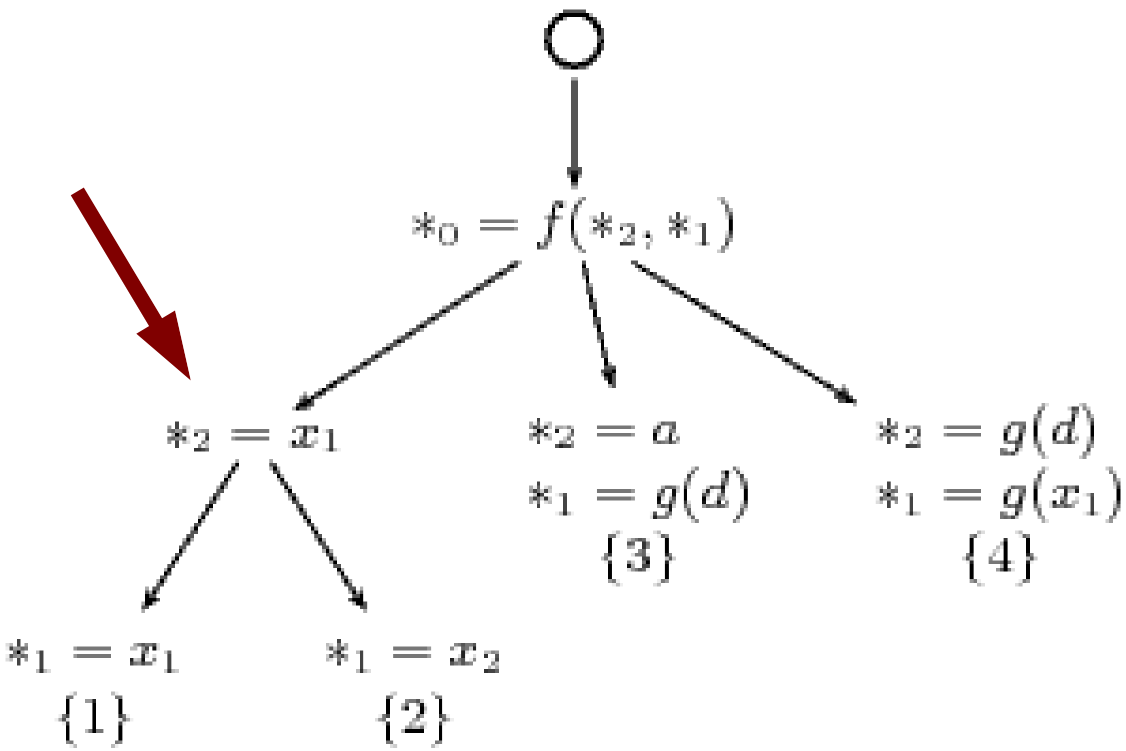
σ:

$*_0 = f(f(a,y_1),y_1)$

$*_1 = y_1$

$*_2 = f(a,y_1)$

$x_1 = f(a,y_1)$



Checked by inline occurs check

Retrieval from Substitution Trees

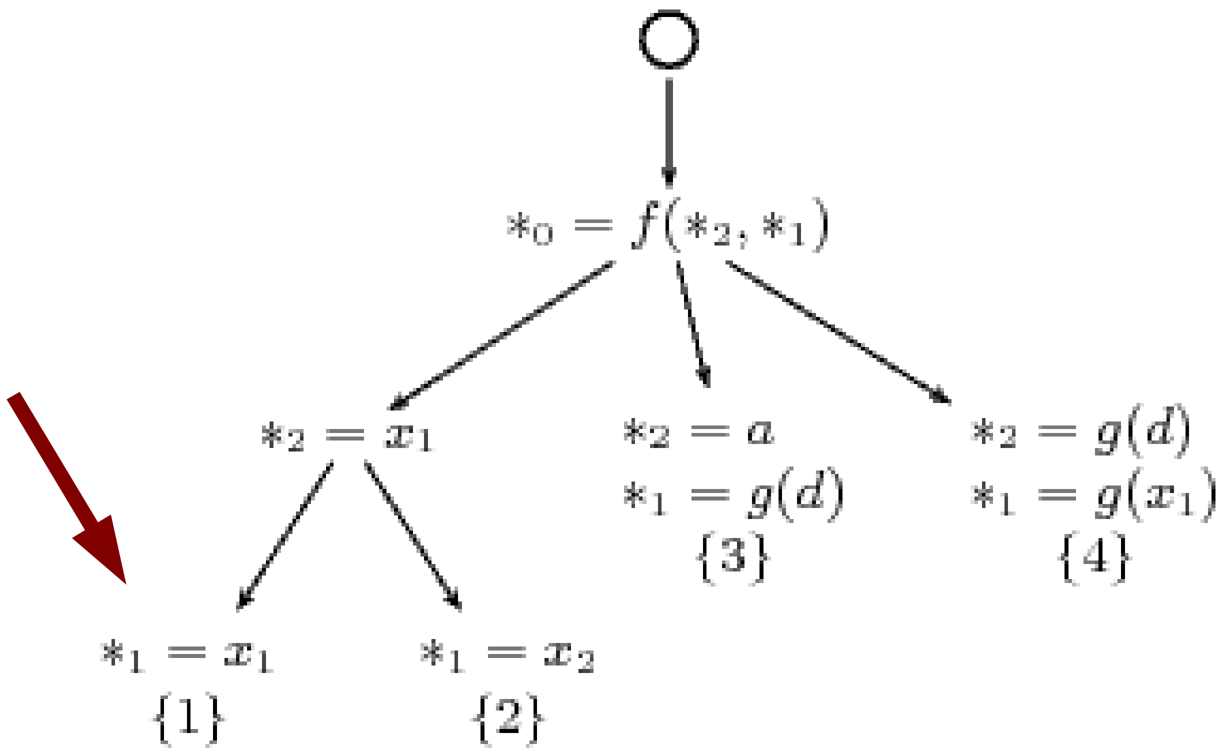
Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$



σ:

$*_0 = f(f(a,y_1),y_1)$

$*_1 = y_1$

$*_2 = f(a,y_1)$

$x_1 = f(a,y_1)$

$y_1 = x_1$

occurs check failure

Checked by inline occurs check

Retrieval from Substitution Trees

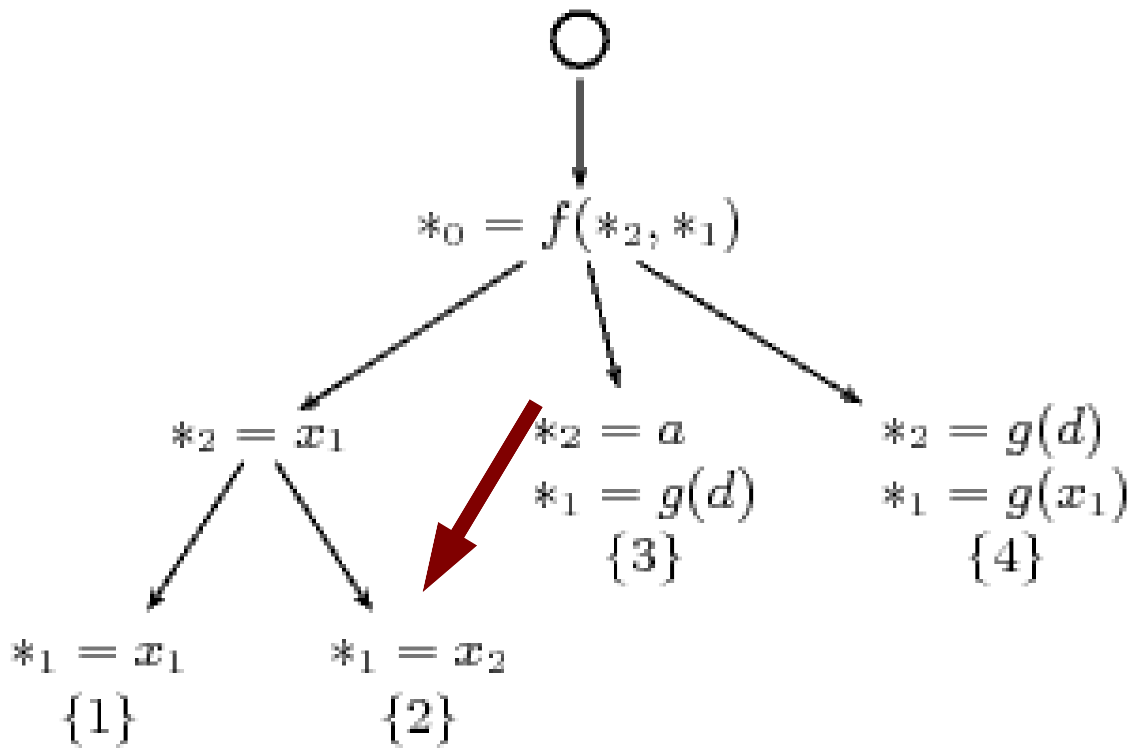
Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$



σ:

$*_0 = f(f(a,y_1),y_1)$

$*_1 = y_1$

$*_2 = f(a,y_1)$

$x_1 = f(a,y_1)$

$x_2 = y_1$

success

Checked by inline occurs check

Retrieval from Substitution Trees

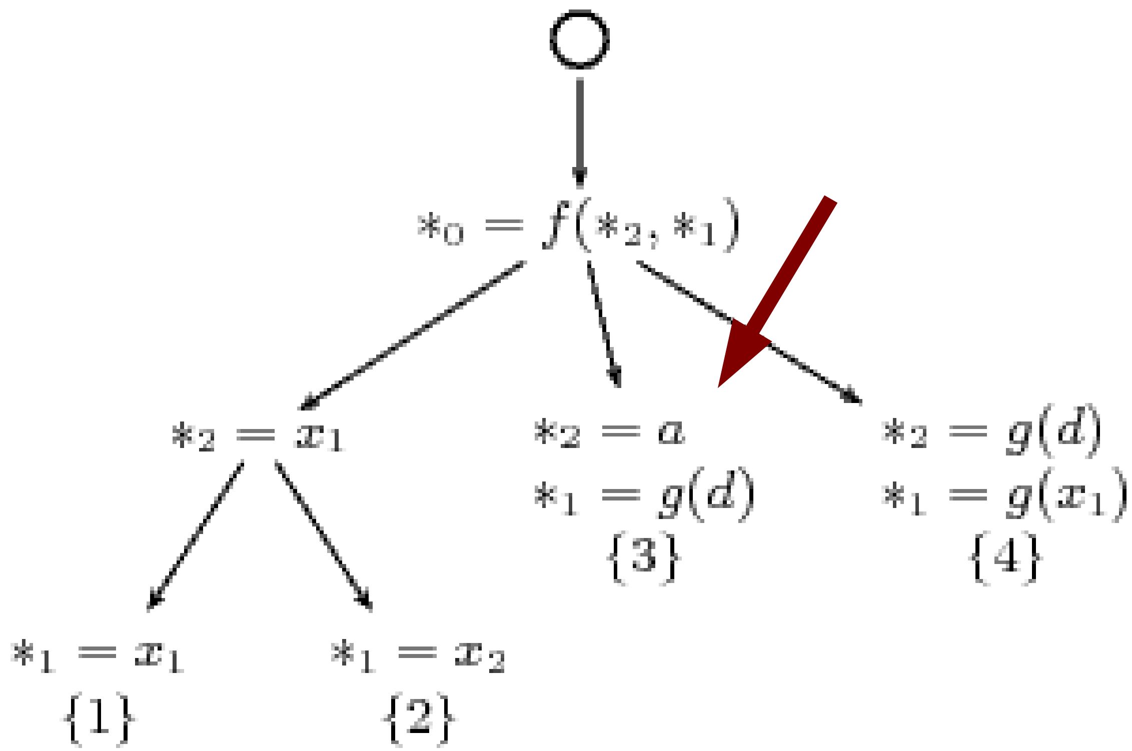
Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$



σ:

$*_0=f(f(a,y_1),y_1)$

$*_1=y_1$

$*_2=f(a,y_1)$

$*_2=a$

mismatch

In downward subst. trees we can skip the node based on top symbols of the $*_2$ bindings

Retrieval from Substitution Trees

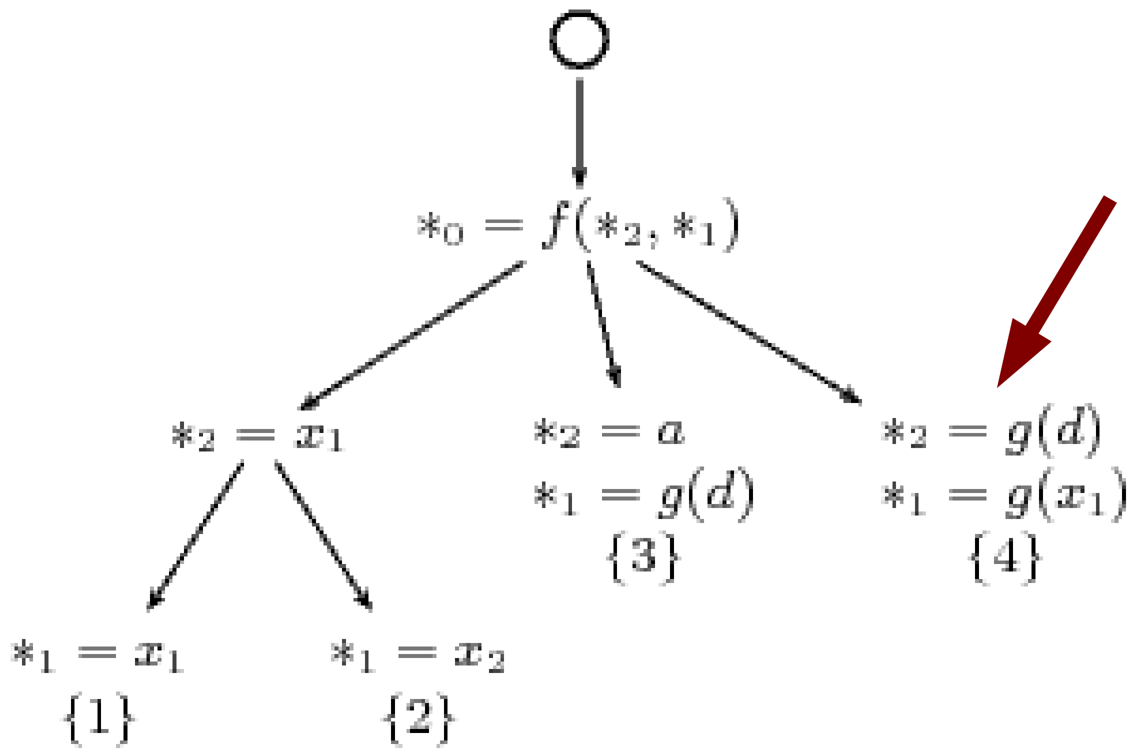
Retrieval of terms unifiable with $f(f(a,y_1),y_1)$

(1) $f(x_1,x_1)$

(2) $f(x_1,x_2)$

(3) $f(a,g(d))$

(4) $f(g(d),g(x_1))$



σ:

$*_0 = f(f(a,y_1),y_1)$

$*_1 = y_1$

$*_2 = f(a,y_1)$

$*_2 = g(d)$

mismatch

In downward subst. trees we can skip the node based on top symbols of the $*_2$ bindings

Our Experiments

- We created benchmarks from several prover runs
  - Operations on the unification index recorded (insertions, deletions, queries)
  - 765 benchmarks (about a half from the resolution index and a half from the backward superposition index)
- ROB, MM, EG and PROB unification algorithms implemented with the required interface
- The index operations performed on each variant of the substitution trees and the time measured

Results

Algorithm	Rel. time (resolution index)	Rel. time (superposition index)
ROB	1.00	1.00
MM	6.96	6.00
EG	1.36	1.30
PROB	1.01	1.01

- The inline occurs check algorithms appear to be more suitable for the substitution trees
  - Lots of unification requests on a single substitution
    - *Inline o. c.* check just the relevant part of a big substitution
    - *Post o. c.* have to be performed after each unification request, not just once per the result substitution

Questions...