Declare Your Language

Chapter 8: Constraint Resolution II

Hendrik van Antwerpen

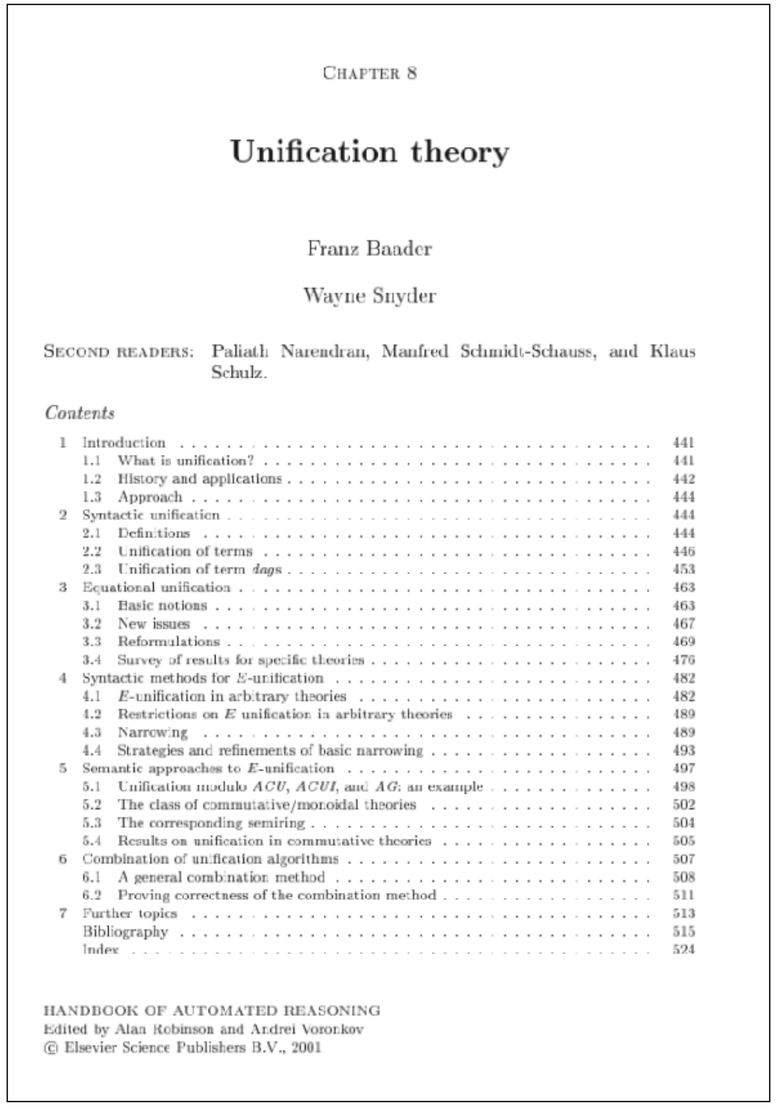
IN4303 Compiler Construction
TU Delft
September 2017

TU Delft

Reading Material

Unification

Baader, F., and W. Snyder
"Unification Theory”
Ch. 8 of Handbook of Automated Deduction
Springer Verlag, Berlin (2001)



CHAPTER 8

Unification theory

Franz Baader

Wayne Snyder

SECOND READERS: Paliath Narendran, Manfred Schmidt-Schauss, and Klaus Schulz.

Contents

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov

http://www.cs.bu.edu/~snyder/publications/UnifChapter.pdf

Efficient Unification with Union-Find

Complexity of Unification (recap)

Space complexity

- Exponential
- Representation of unifier

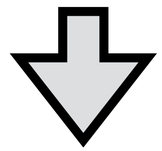
Time complexity

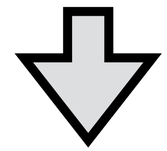
- Exponential
- Recursive calls on terms

Solution

- Union-Find algorithm
- Complexity growth can be considered constant

$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) ==$
$h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$





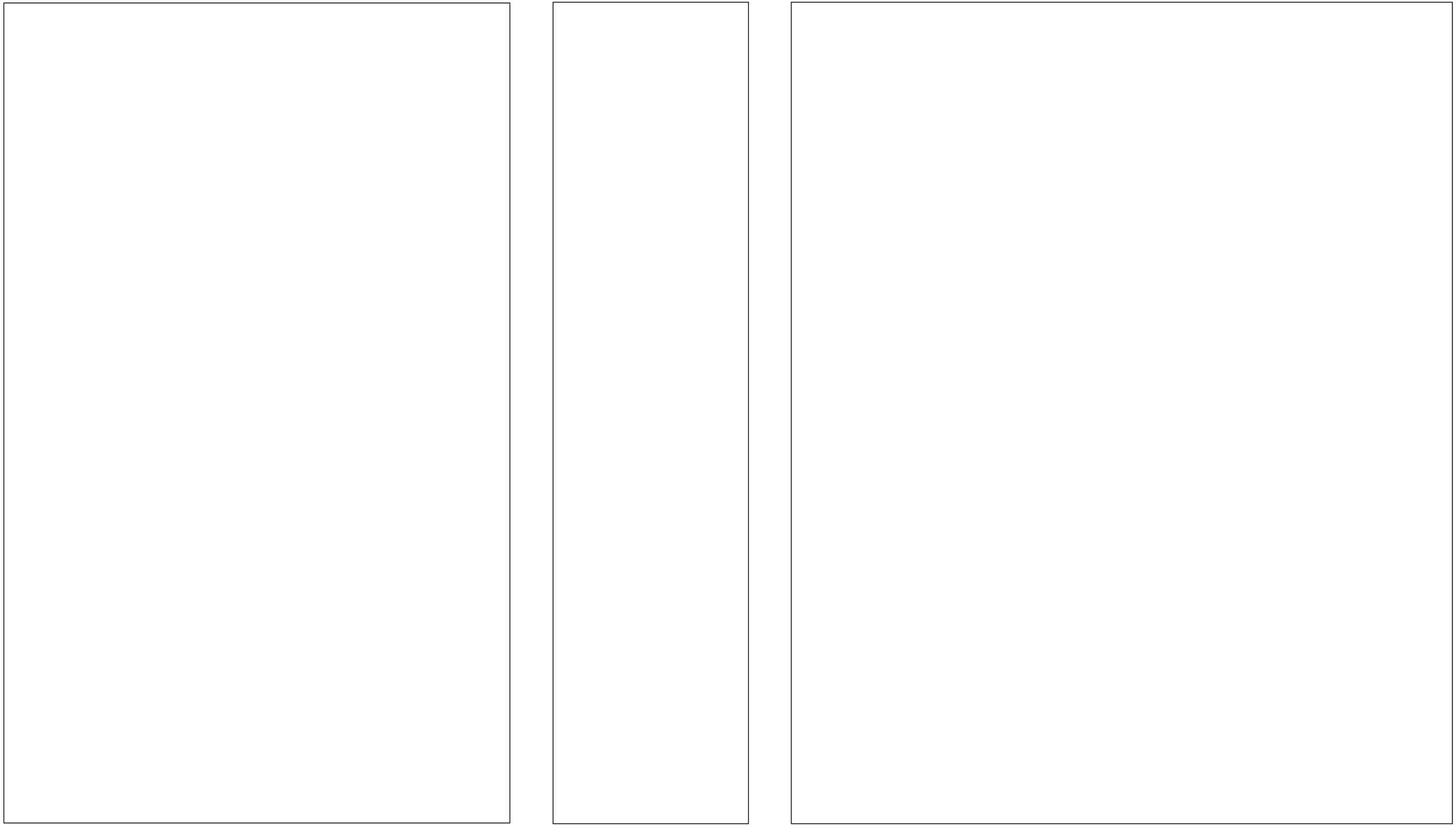
a_1 -> $f(a_0,a_0)$
a_2 -> $f(f(a_0,a_0), f(a_0,a_0))$
a_i -> … $2^{i+1}-1$ subterms …
b_1 -> $f(a_0,a_0)$
b_2 -> $f(f(a_0,a_0), f(a_0,a_0))$
b_i -> … $2^{i+1}-1$ subterms …

fully applied

a_1 -> $f(a_0,a_0)$
a_2 -> $f(a_1,a_1)$
a_i -> … 3 subterms …
b_1 -> $f(a_0,a_0)$
b_2 -> $f(a_1,a_1)$
b_i -> … 3 subterms …

triangular

Set Representatives

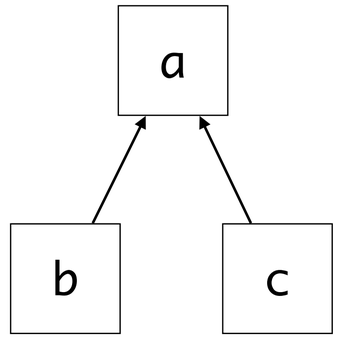


Set Representatives

```
a == b
c == a
```

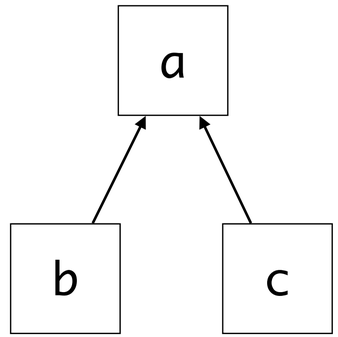
Set Representatives

a == b
c == a



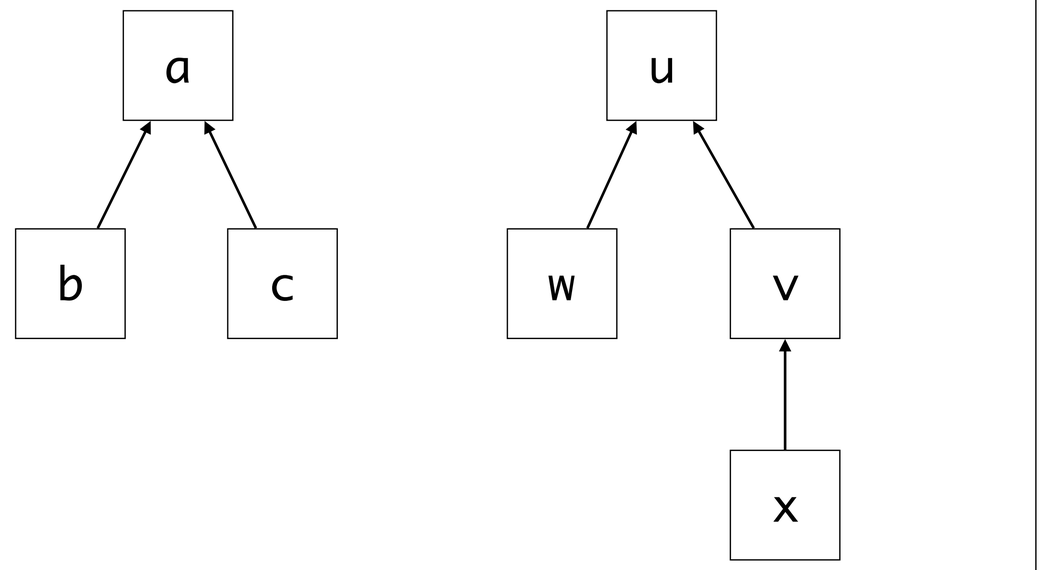
Set Representatives

a == b
c == a
u == w
v == u
x == v



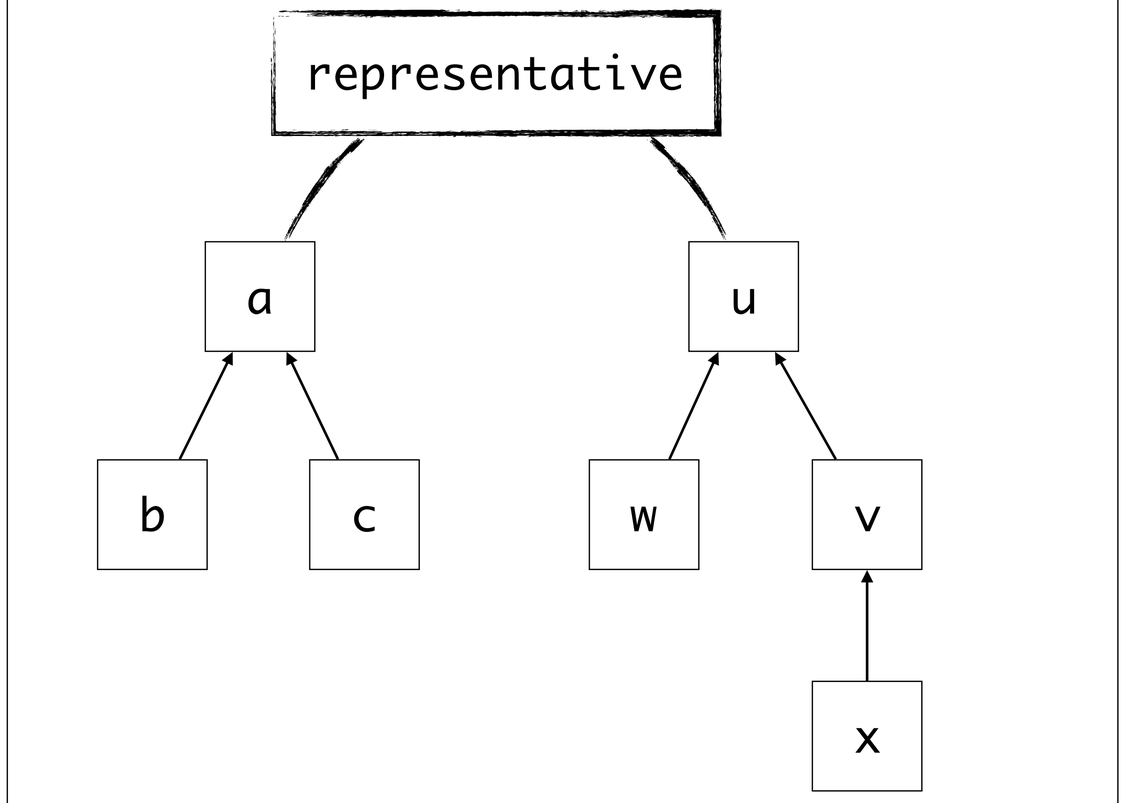
Set Representatives

a == b
c == a
u == w
v == u
x == v



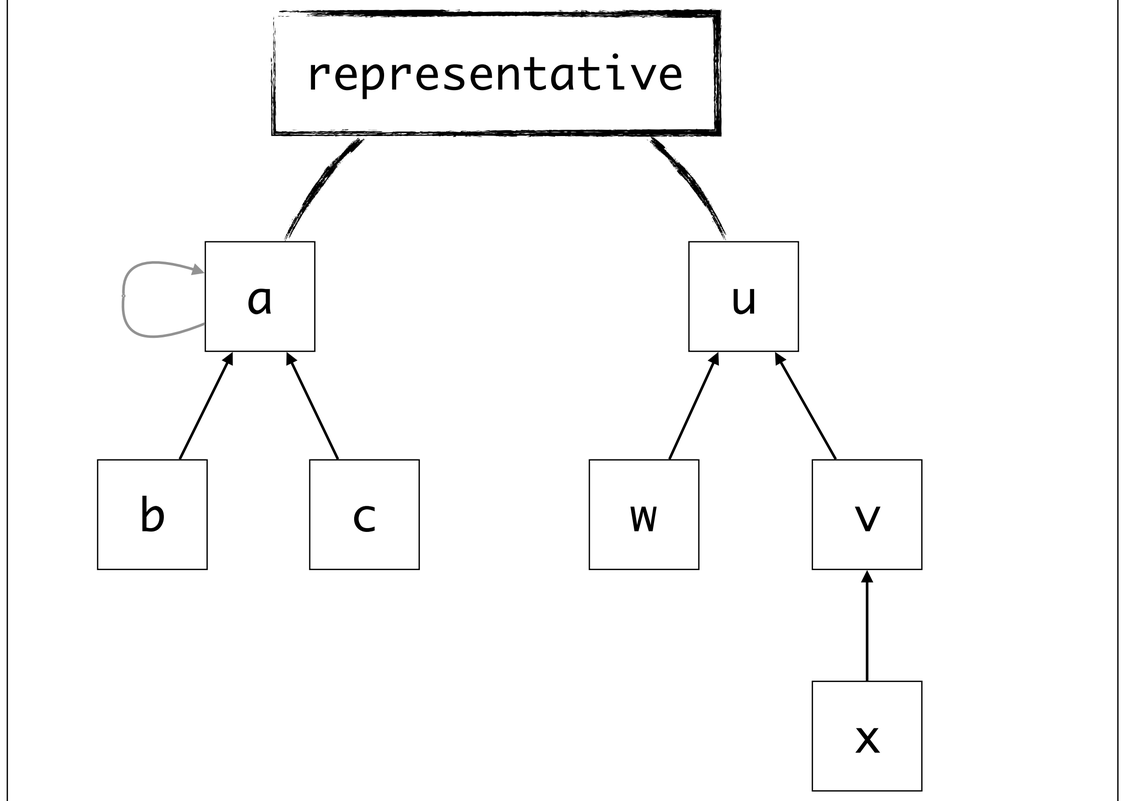
Set Representatives

a == b
c == a
u == w
v == u
x == v



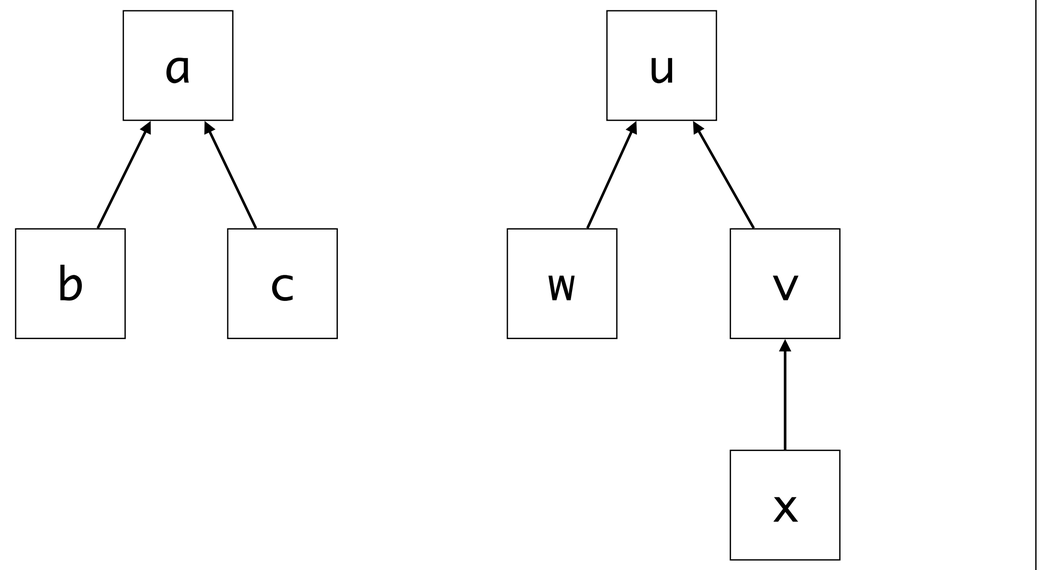
Set Representatives

a == b
c == a
u == w
v == u
x == v



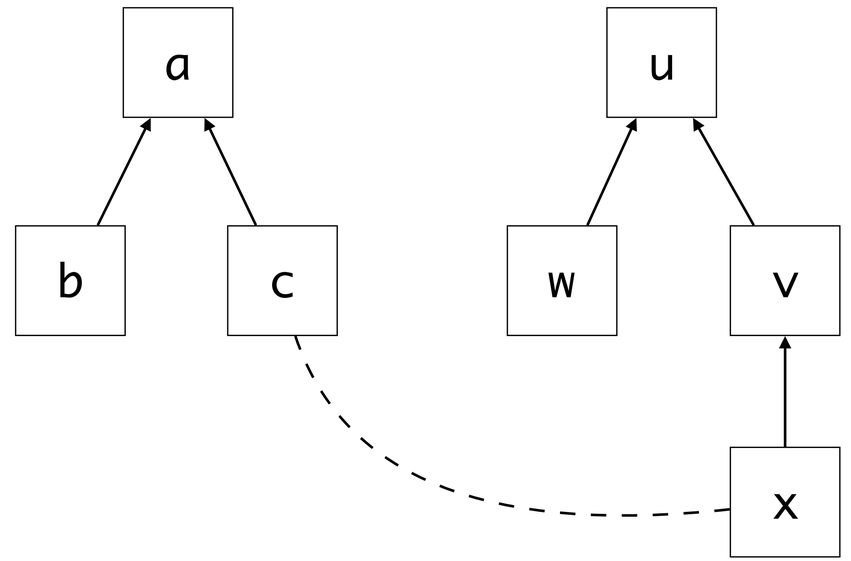
Set Representatives

a == b
c == a
u == w
v == u
x == v



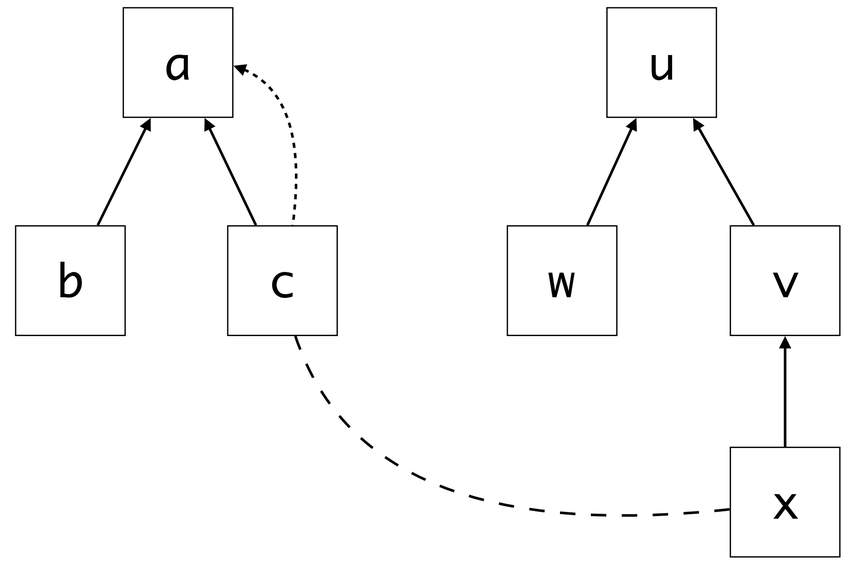
Set Representatives

a == b
c == a
u == w
v == u
x == v
x == c



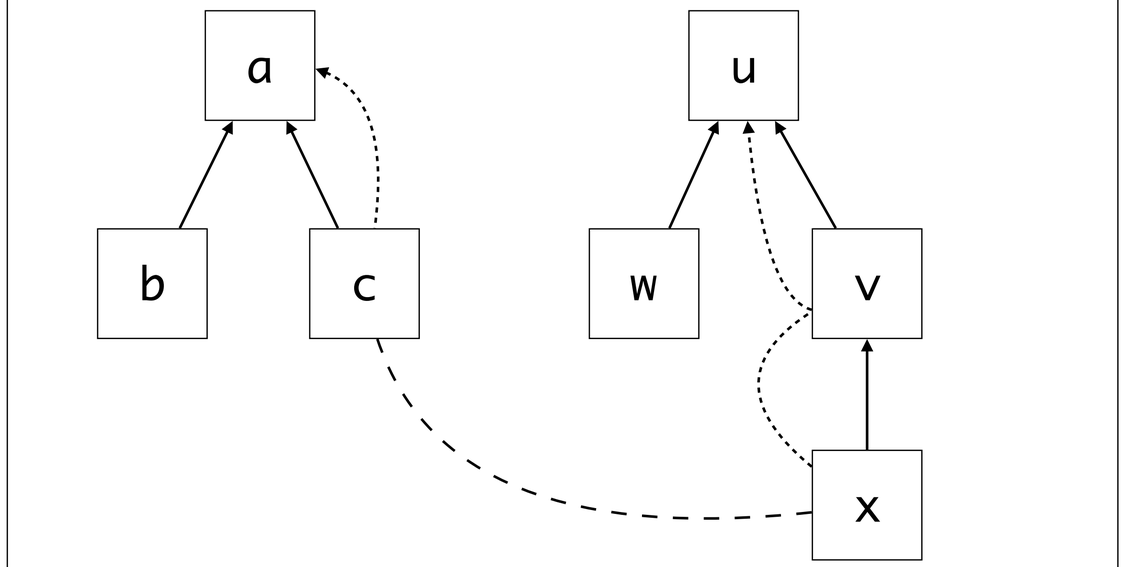
Set Representatives

a == b
c == a
u == w
v == u
x == v
x == c



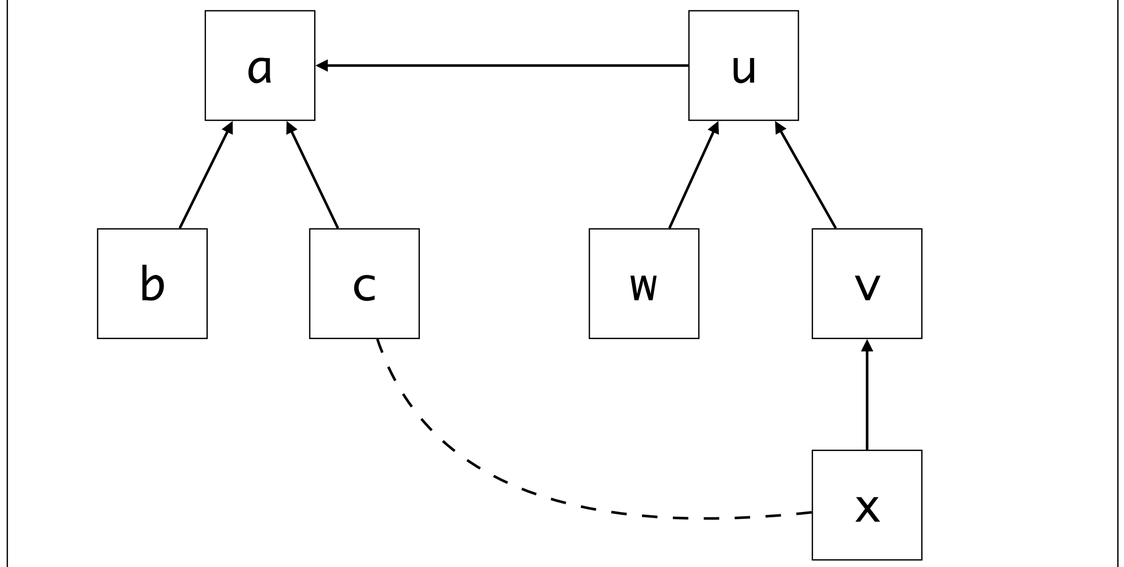
Set Representatives

a == b
c == a
u == w
v == u
x == v
x == c



Set Representatives

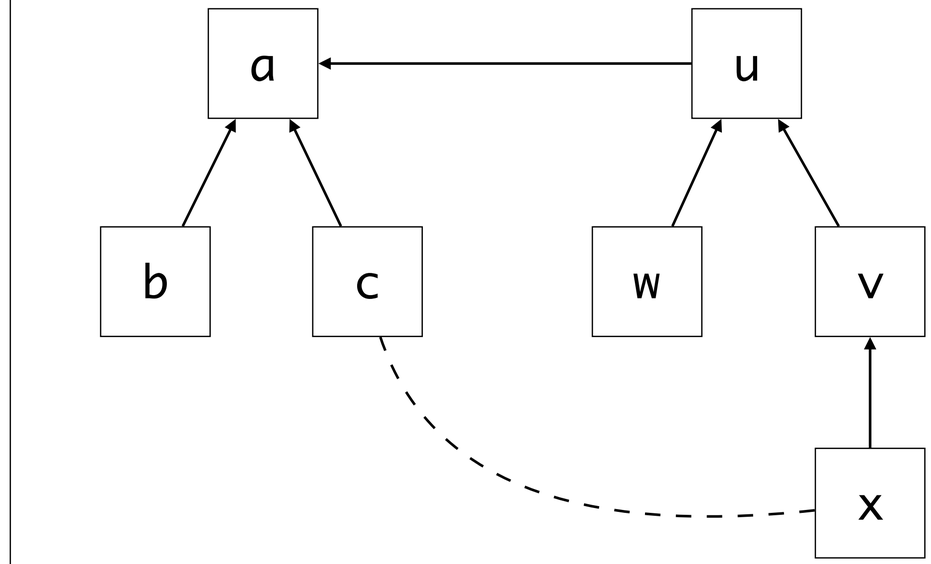
a == b
c == a
u == w
v == u
x == v
x == c



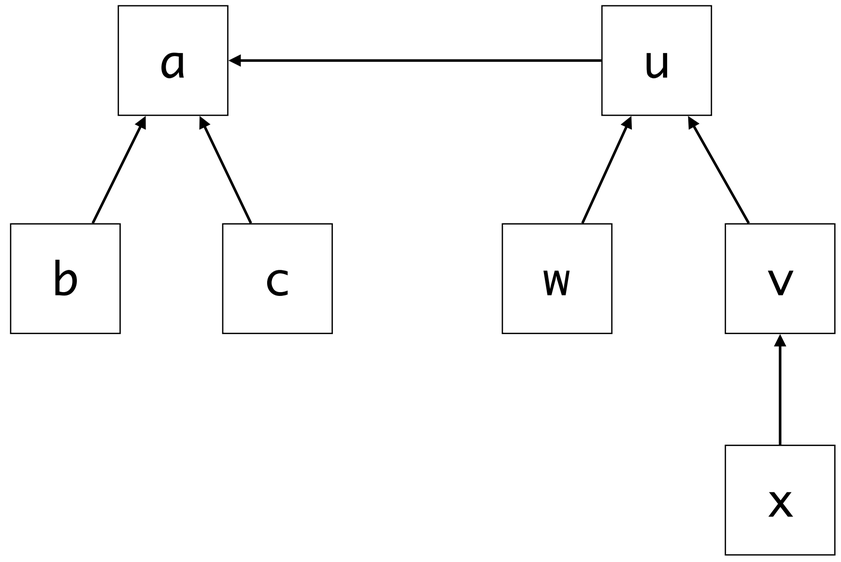
Set Representatives

```
FIND(a):
  b := rep(a)
  if b == a:
     return a
  else
     return FIND(b)



UNION(a1,a2):
  b1 := FIND(a1)
  b2 := FIND(a2)
  LINK(b1,b2)

LINK(a1,a2):
  rep(a1) := a2
```

```
a == b
c == a
u == w
v == u
x == v
x == c
```



Path Compression



Path Compression

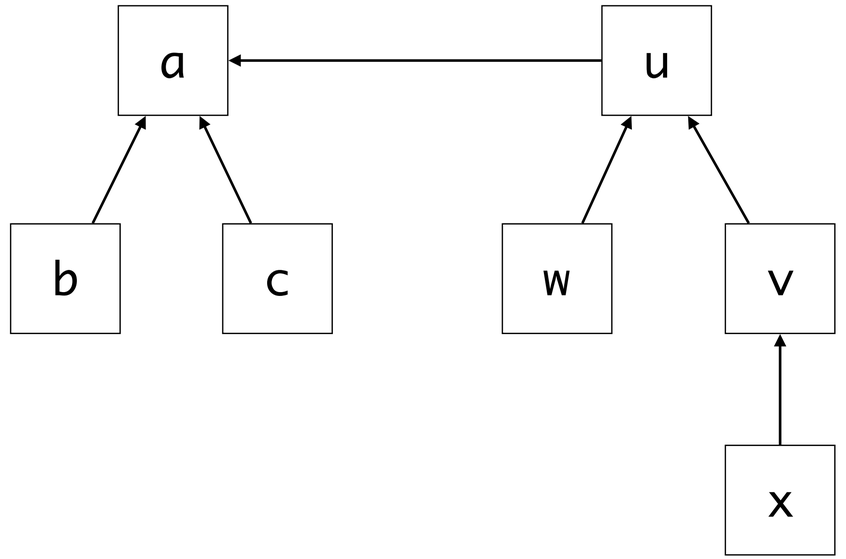
...

x == b

x == c

x == w

x == v



Path Compression

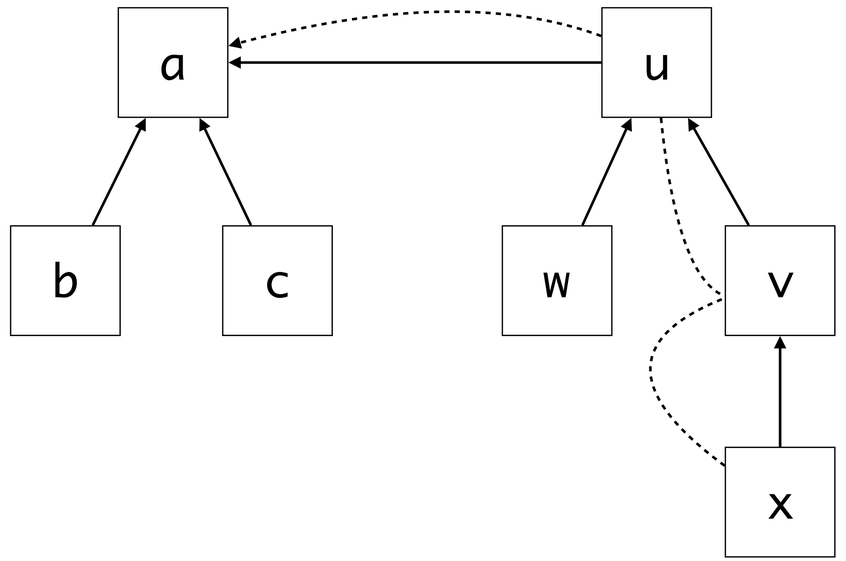
...

x == b

x == c

x == w

x == v



Path Compression

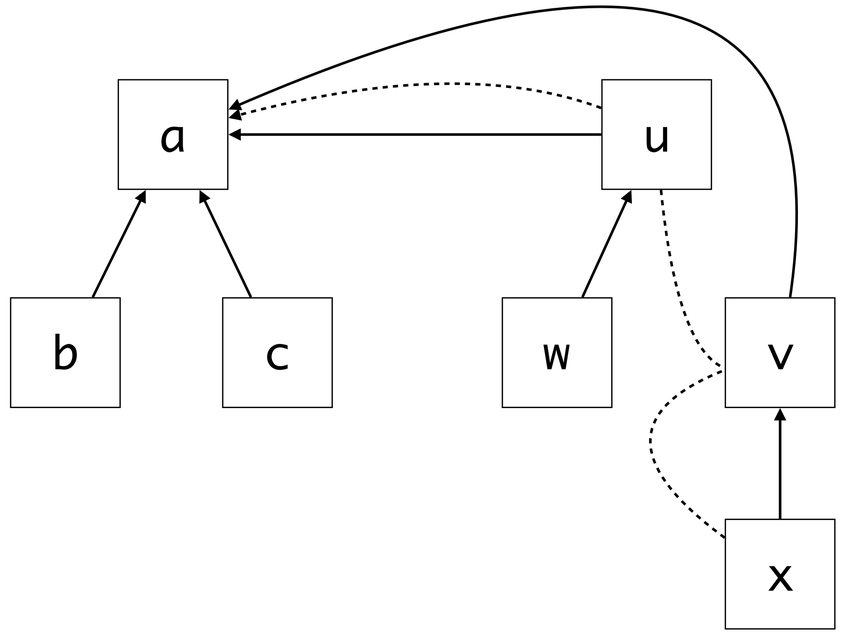
...

x == b

x == c

x == w

x == v



Path Compression

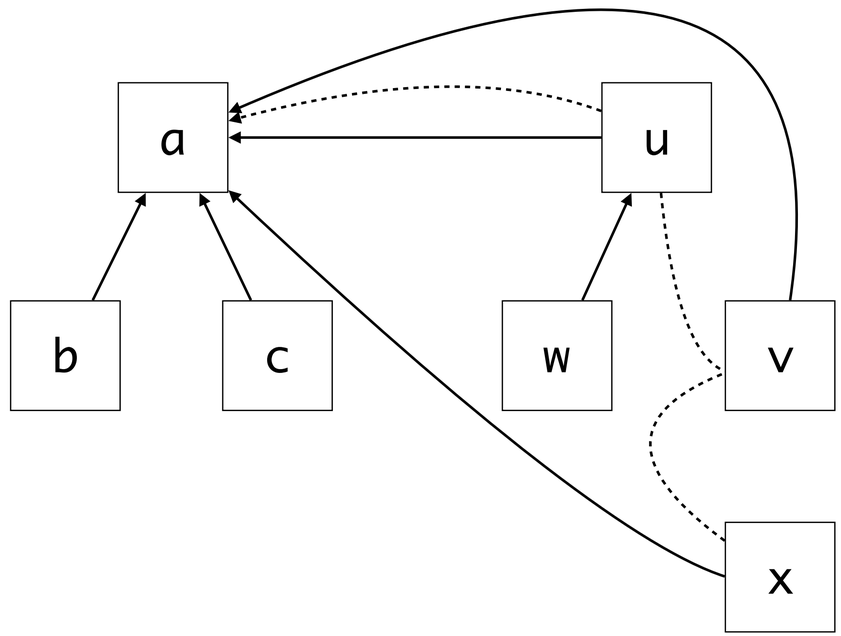
…

x == b

x == c

x == w

x == v



Path Compression

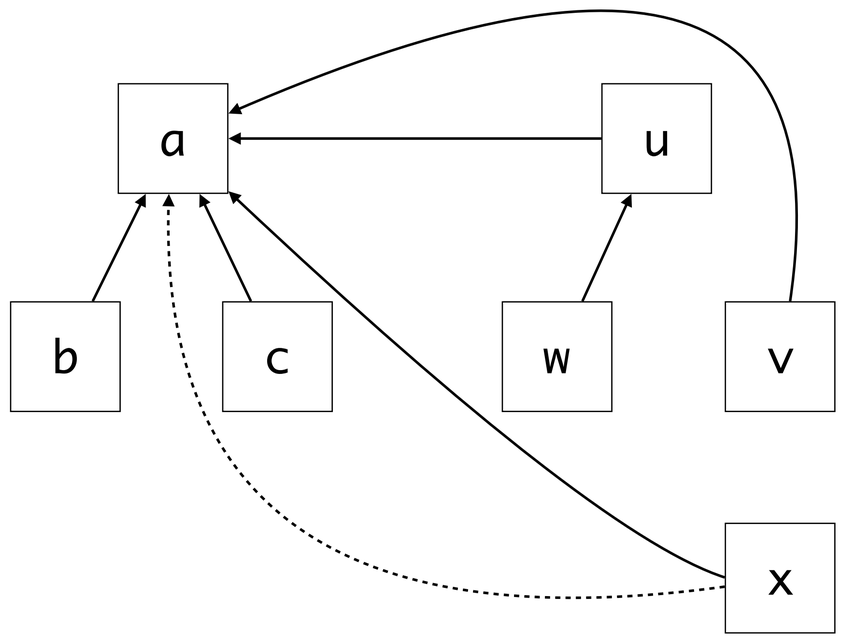
...

x == b

x == c

x == w

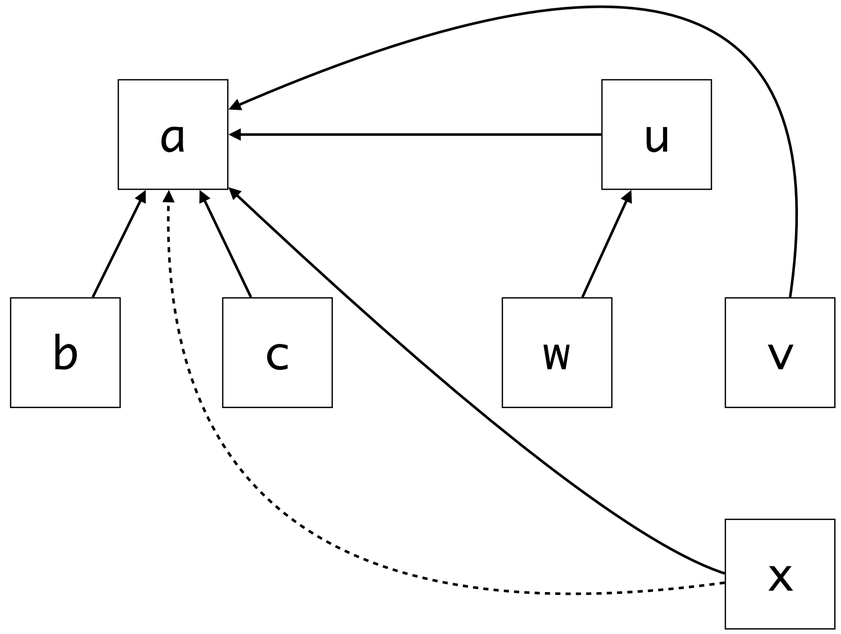
x == v



Path Compression

```
FIND(a):
  b := rep(a)
  if b == a:
     return a
  else
     b := FIND(b)
     rep(a) := b
     return b

UNION(a1,a2):
  b1 := FIND(a1)
  b2 := FIND(a2)
  LINK(b1,b2)

LINK(a1,a2):
  rep(a1) := a2
```

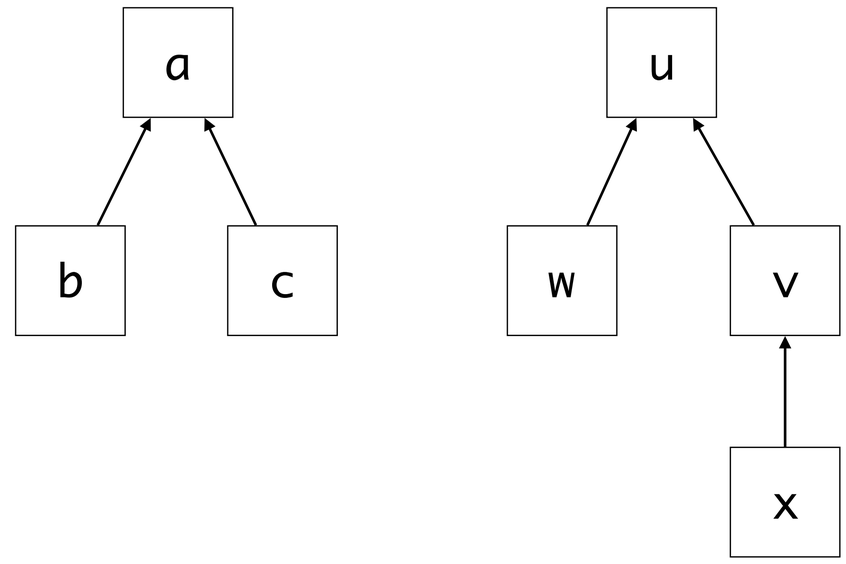
```
…
x == b
x == c
x == w
x == v
```



Tree Balancing

...

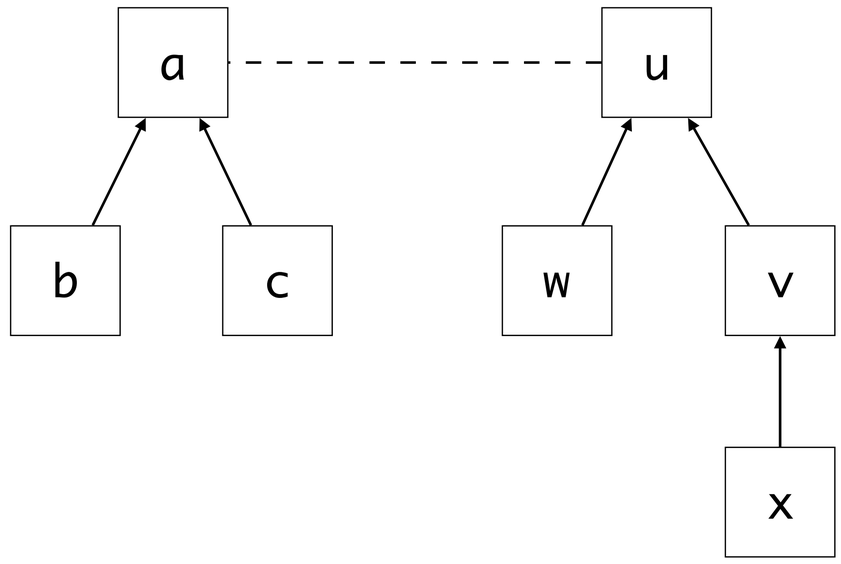
x == c



Tree Balancing

...

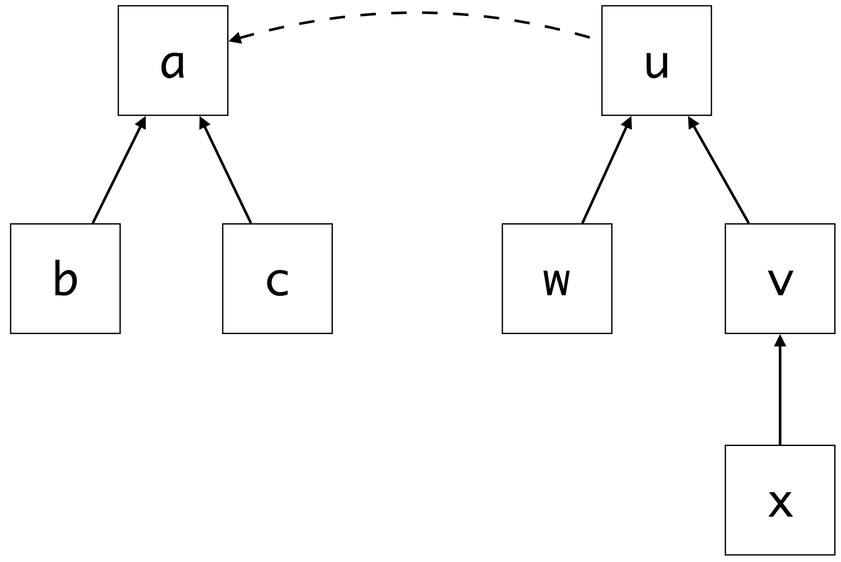
x == c



Tree Balancing

...

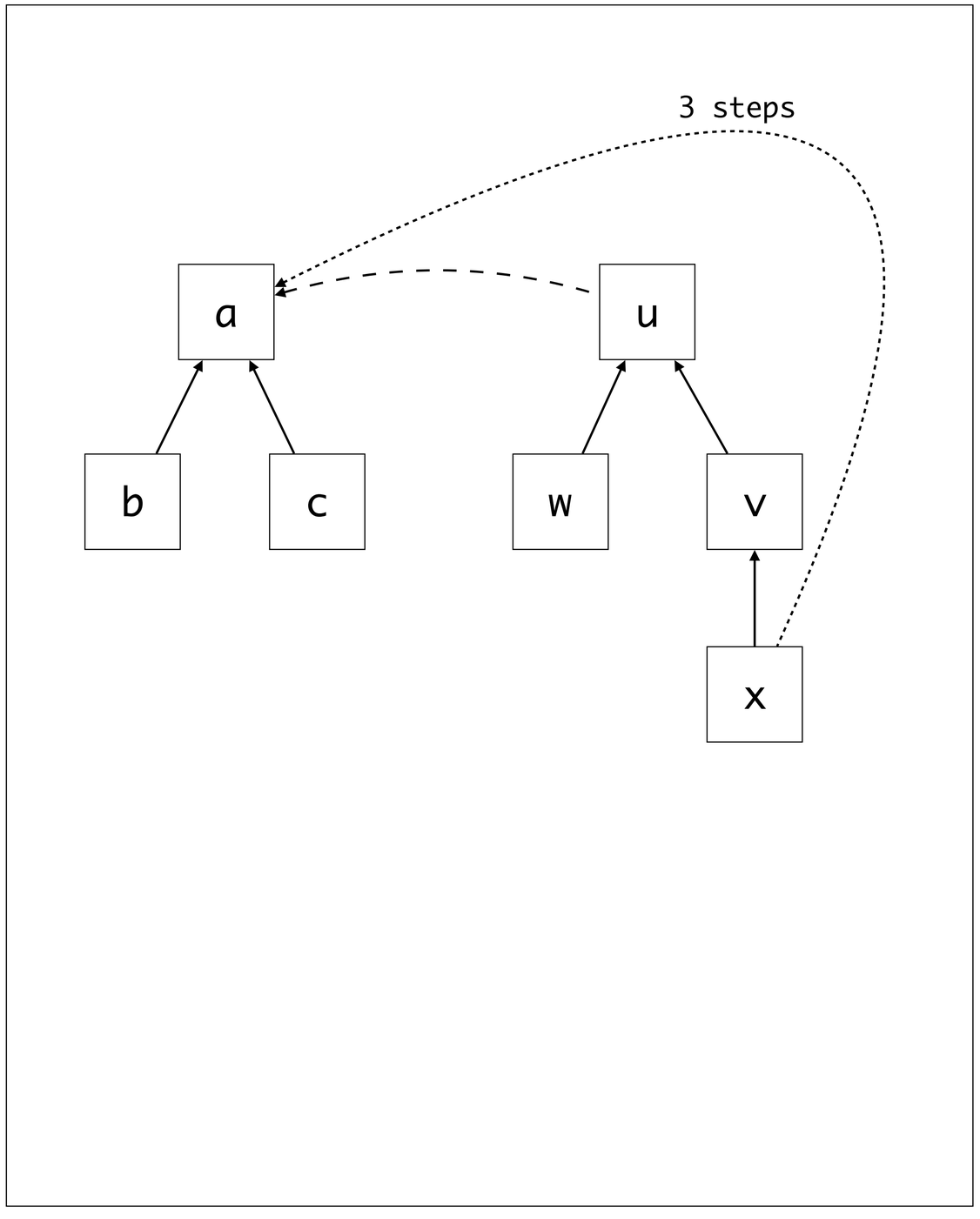
x == c



Tree Balancing

...

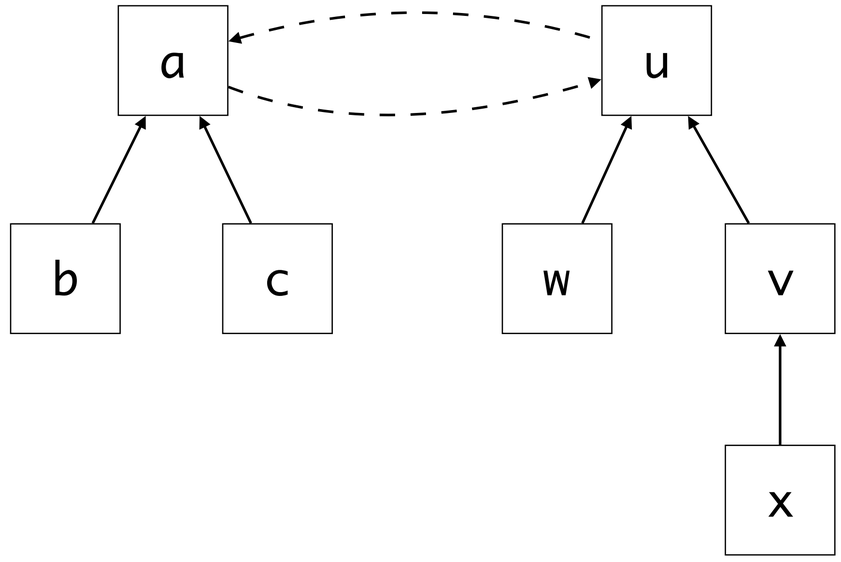
x == c



Tree Balancing

...

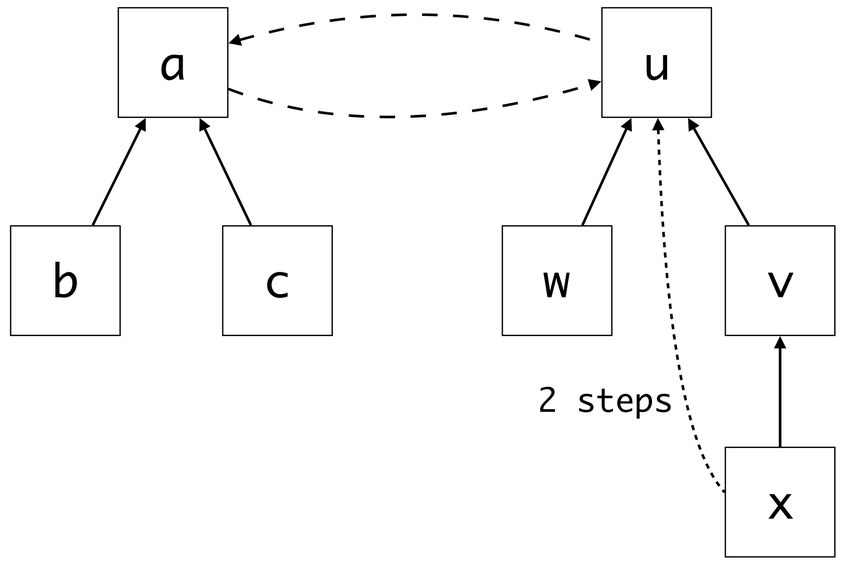
x == c



Tree Balancing

...

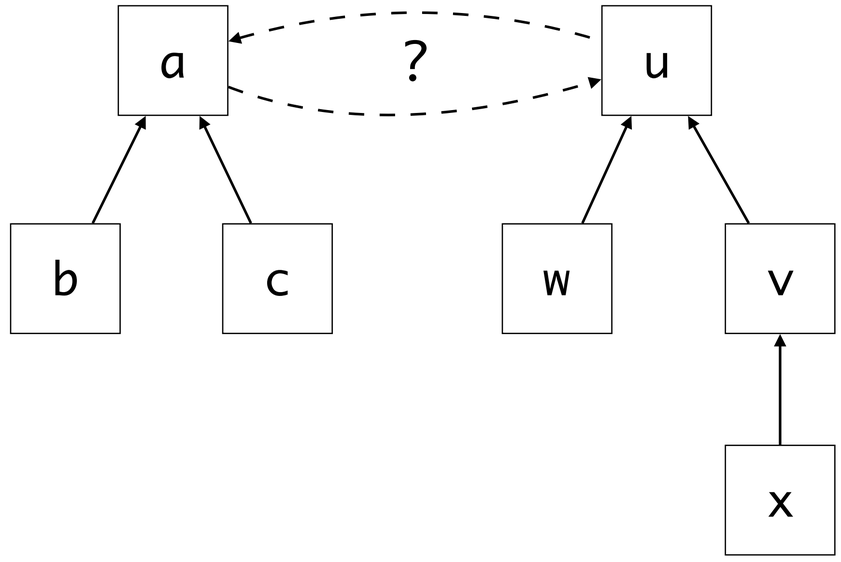
x == c



Tree Balancing

...

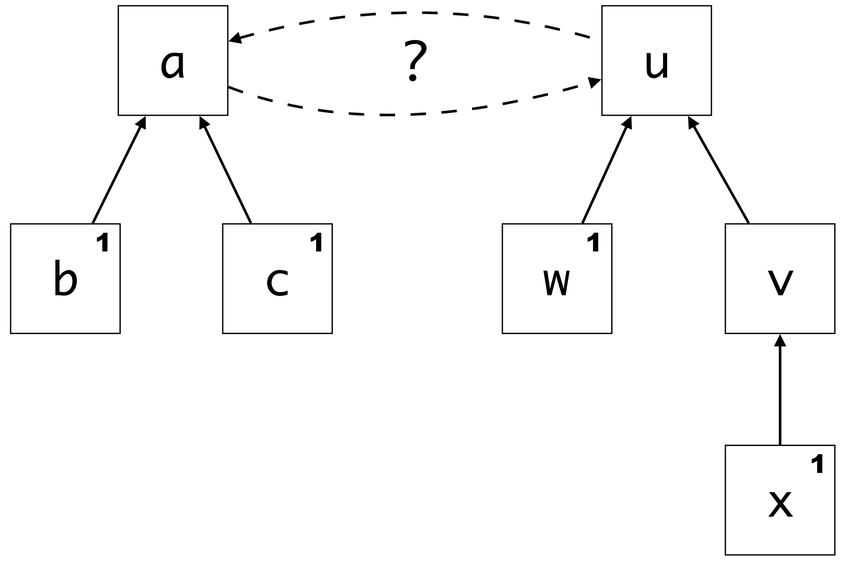
x == c



Tree Balancing

...

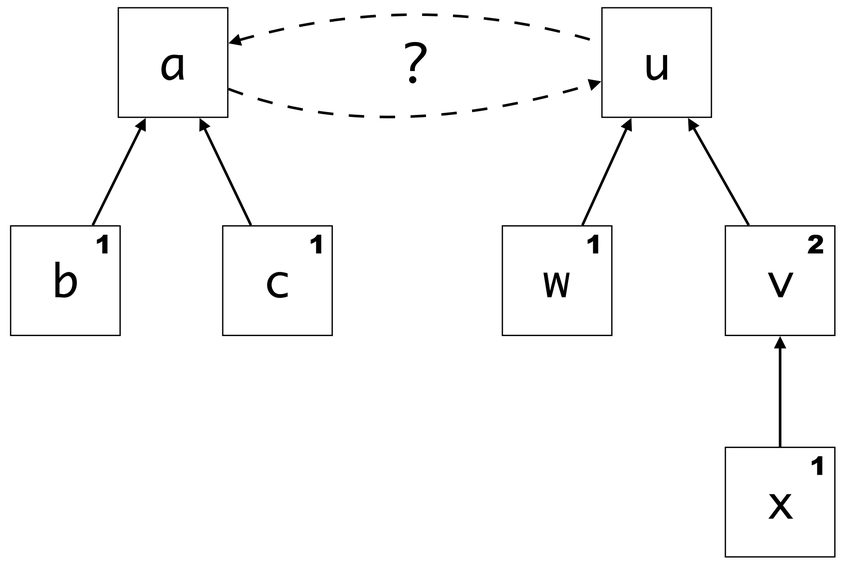
x == c



Tree Balancing

...

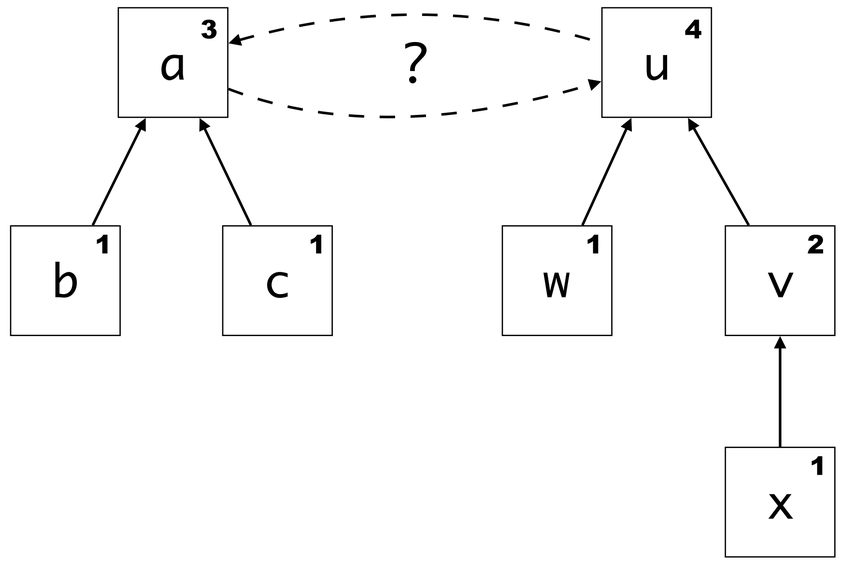
x == c



Tree Balancing

...

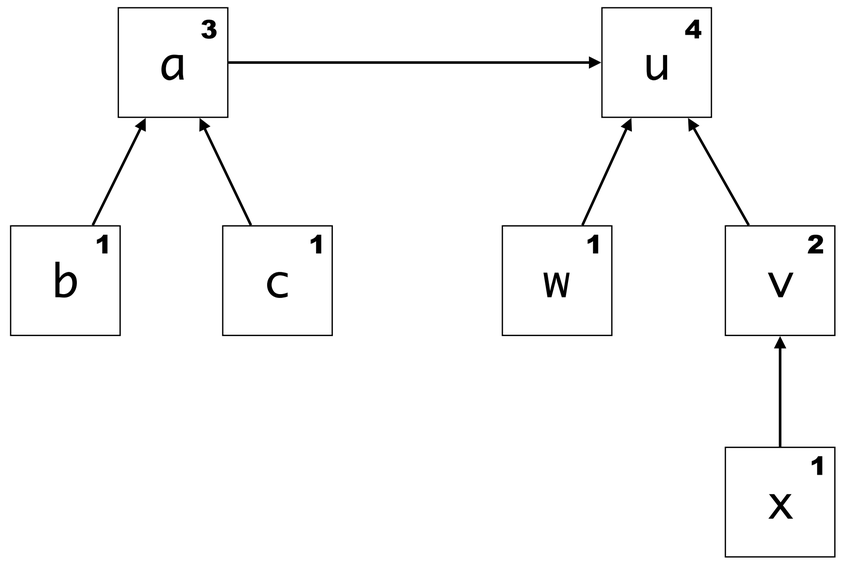
x == c



Tree Balancing

...

x == c

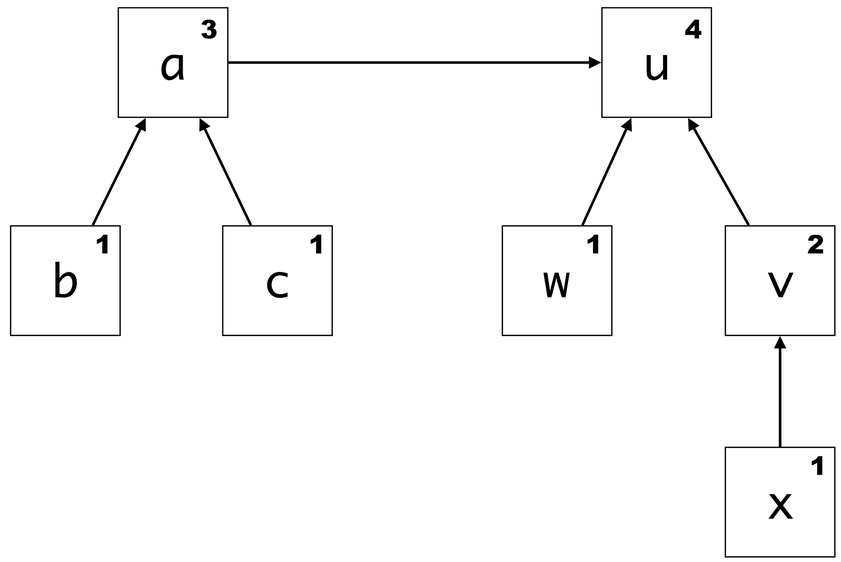


Tree Balancing

```
FIND(a):
  b := rep(a)
  if b == a:
     return a
  else
     b := FIND(b)
     rep(a) := b
     return b

UNION(a1,a2):
  b1 := FIND(a1)
  b2 := FIND(a2)
  LINK(b1,b2)

LINK(a1,a2):
  if size(a2) > size(a1):
     rep(a1) := a2
     size(a2) += size(a1)
  else:
     rep(a2) := a1
     size(a1) += size(a2)
```

…

x == c

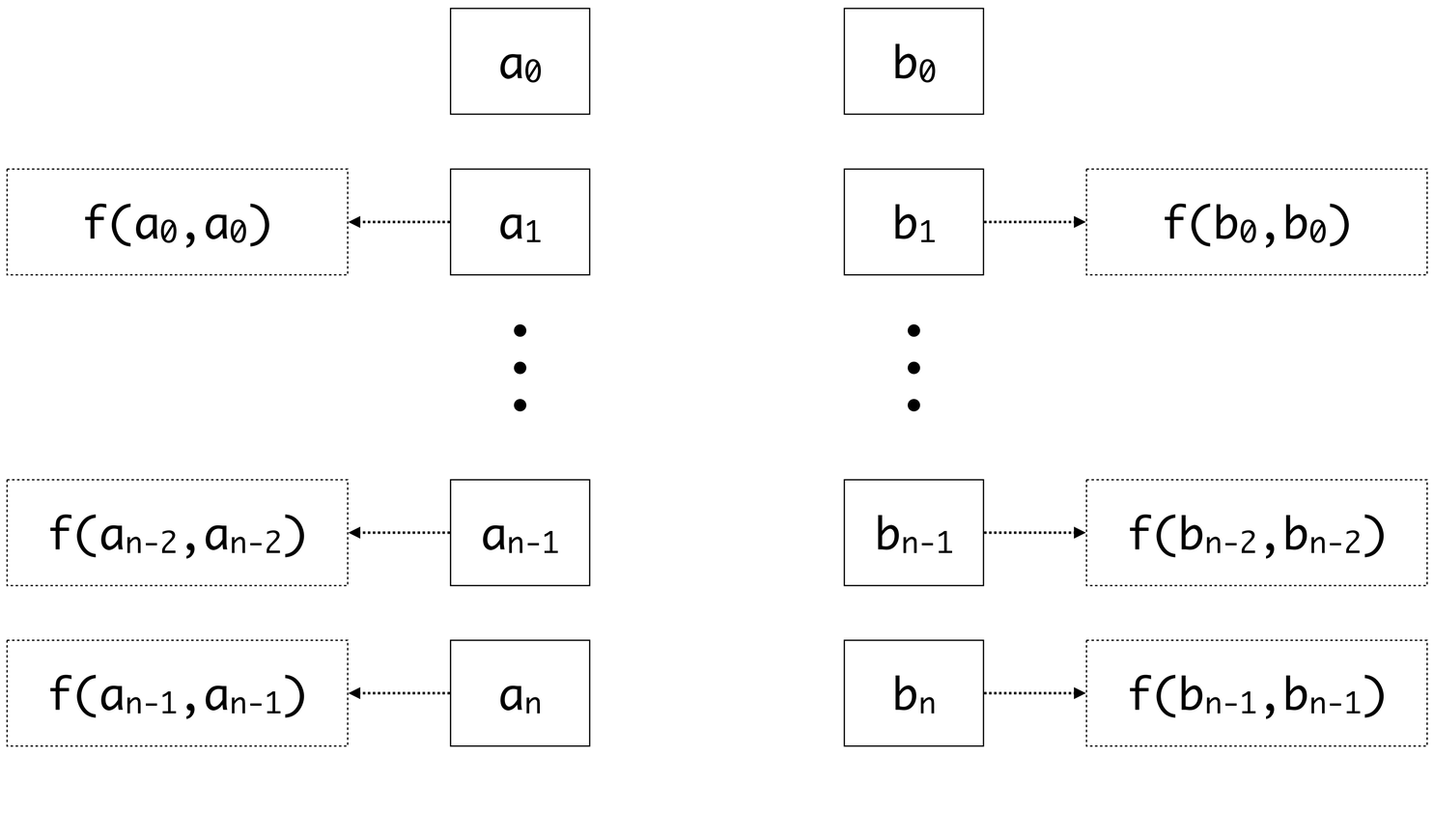


The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$

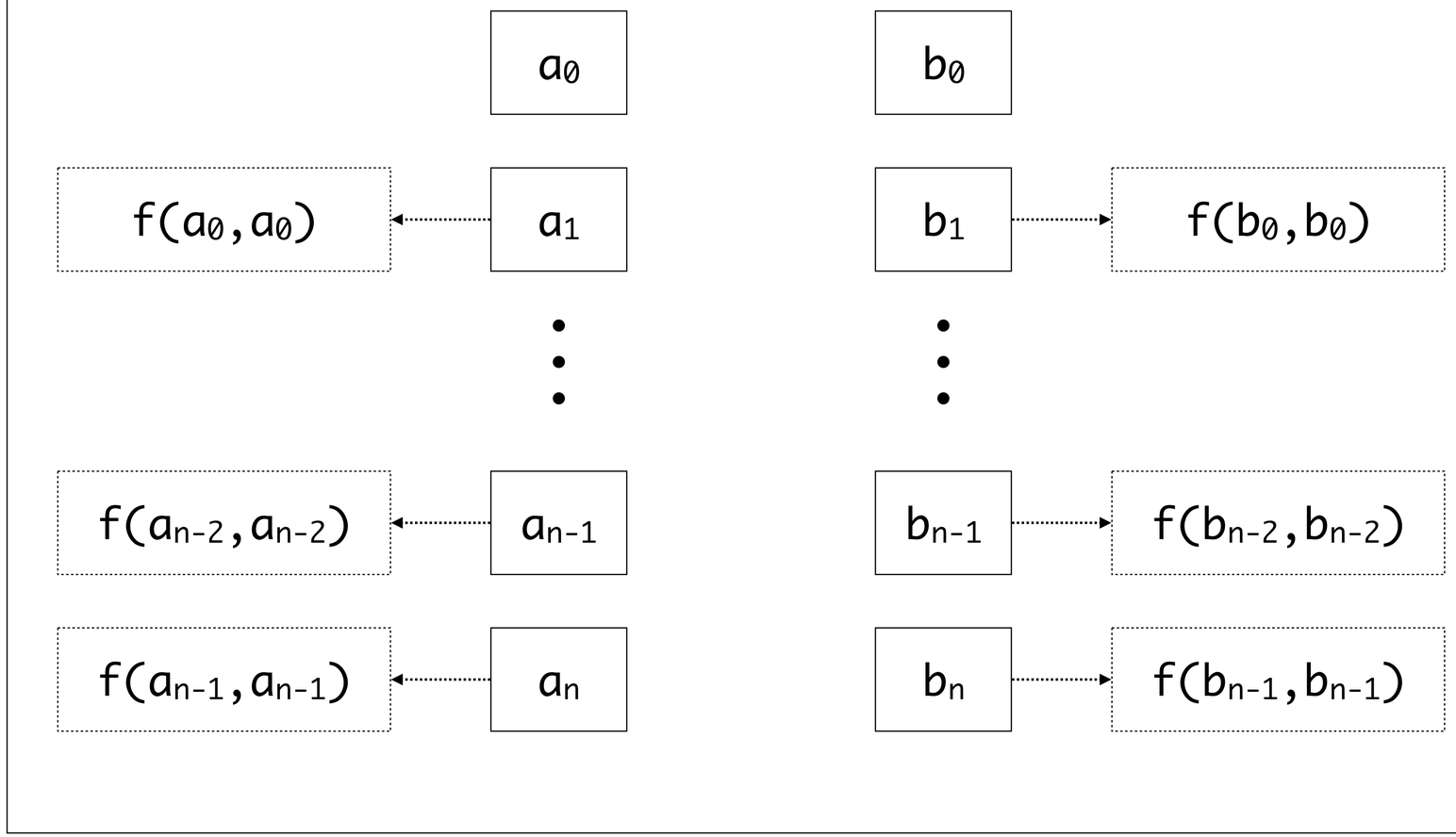
The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



The Complex Case

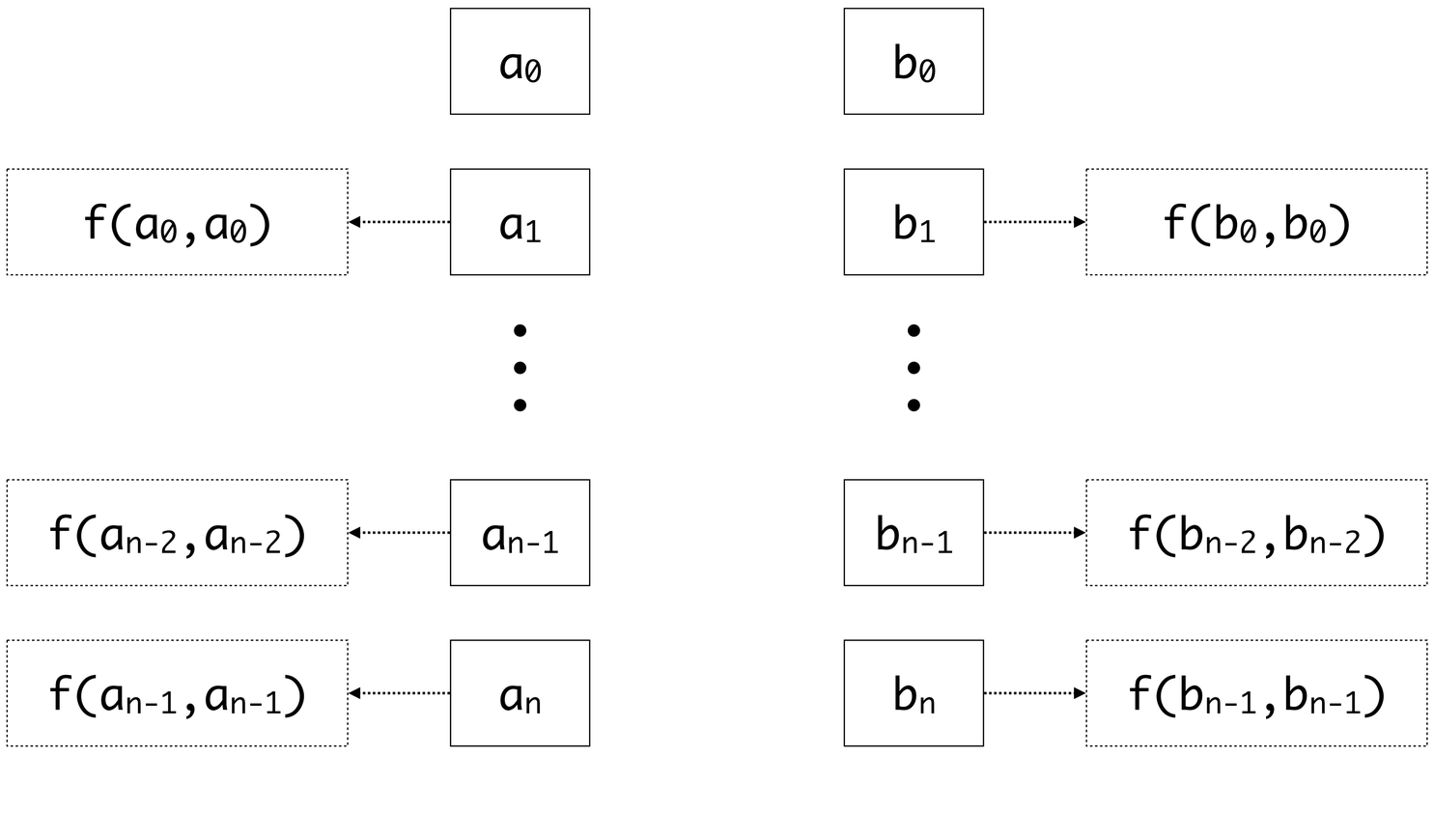
$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$



$a_n == b_n$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$

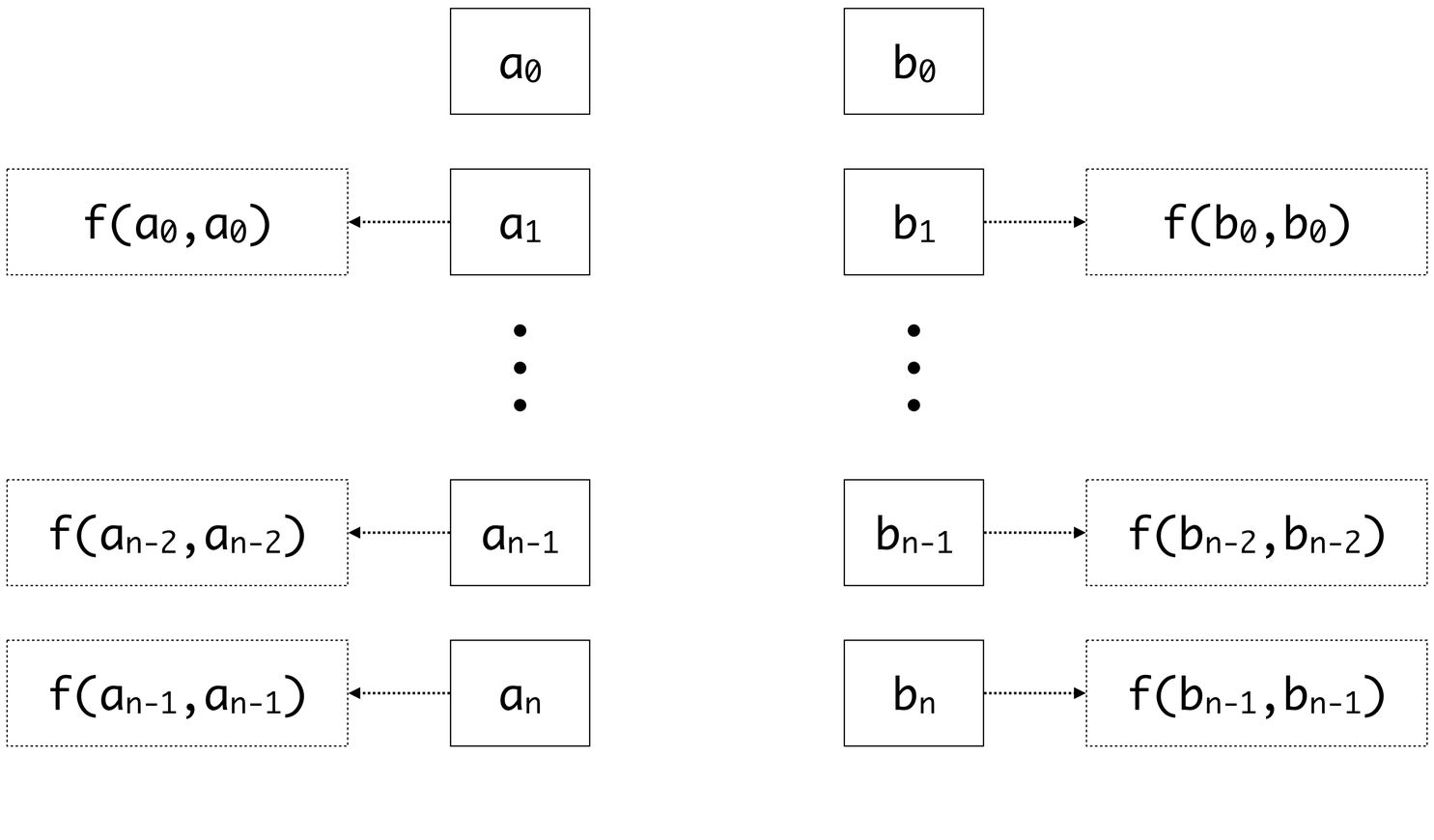


$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



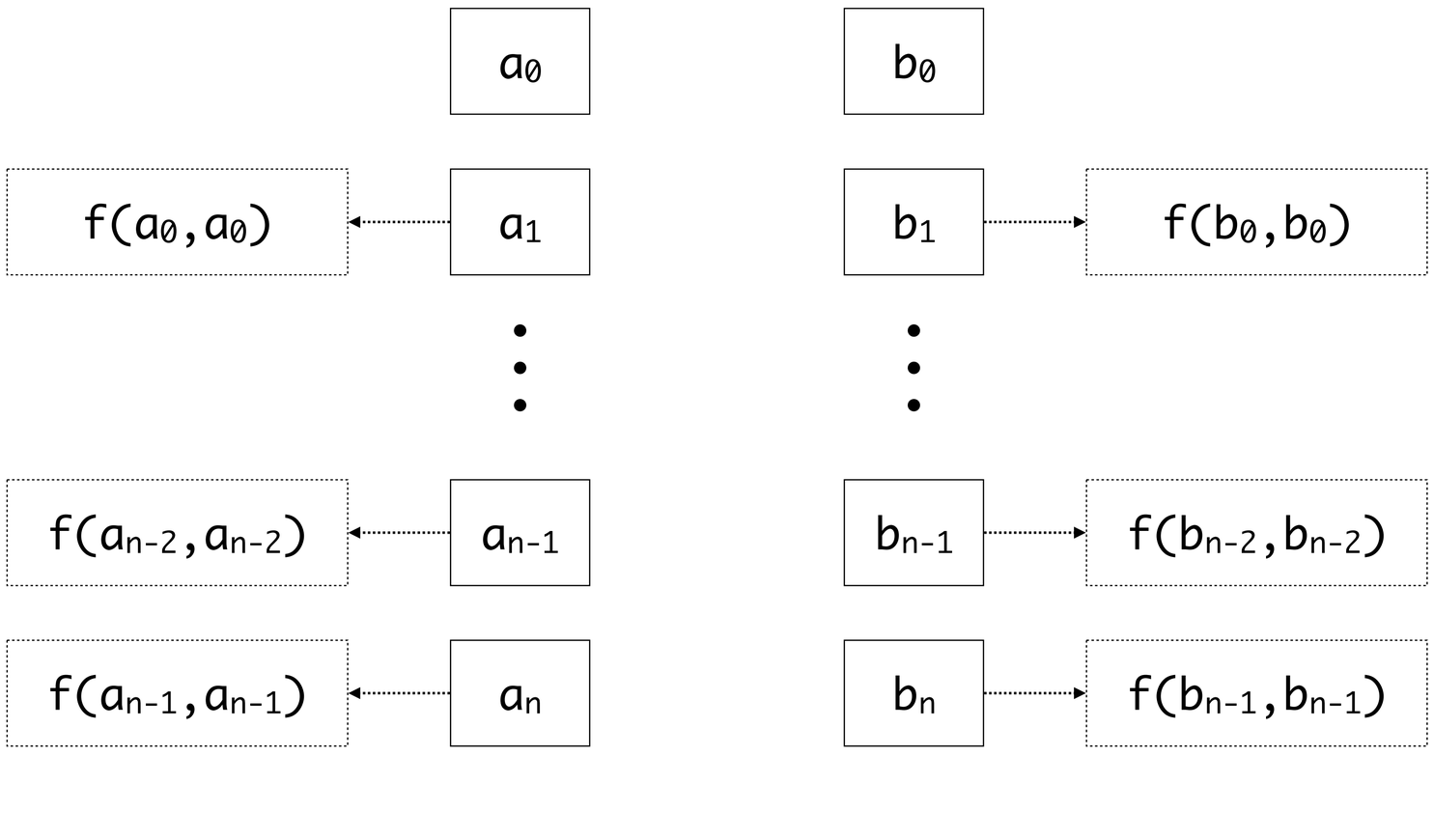
$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

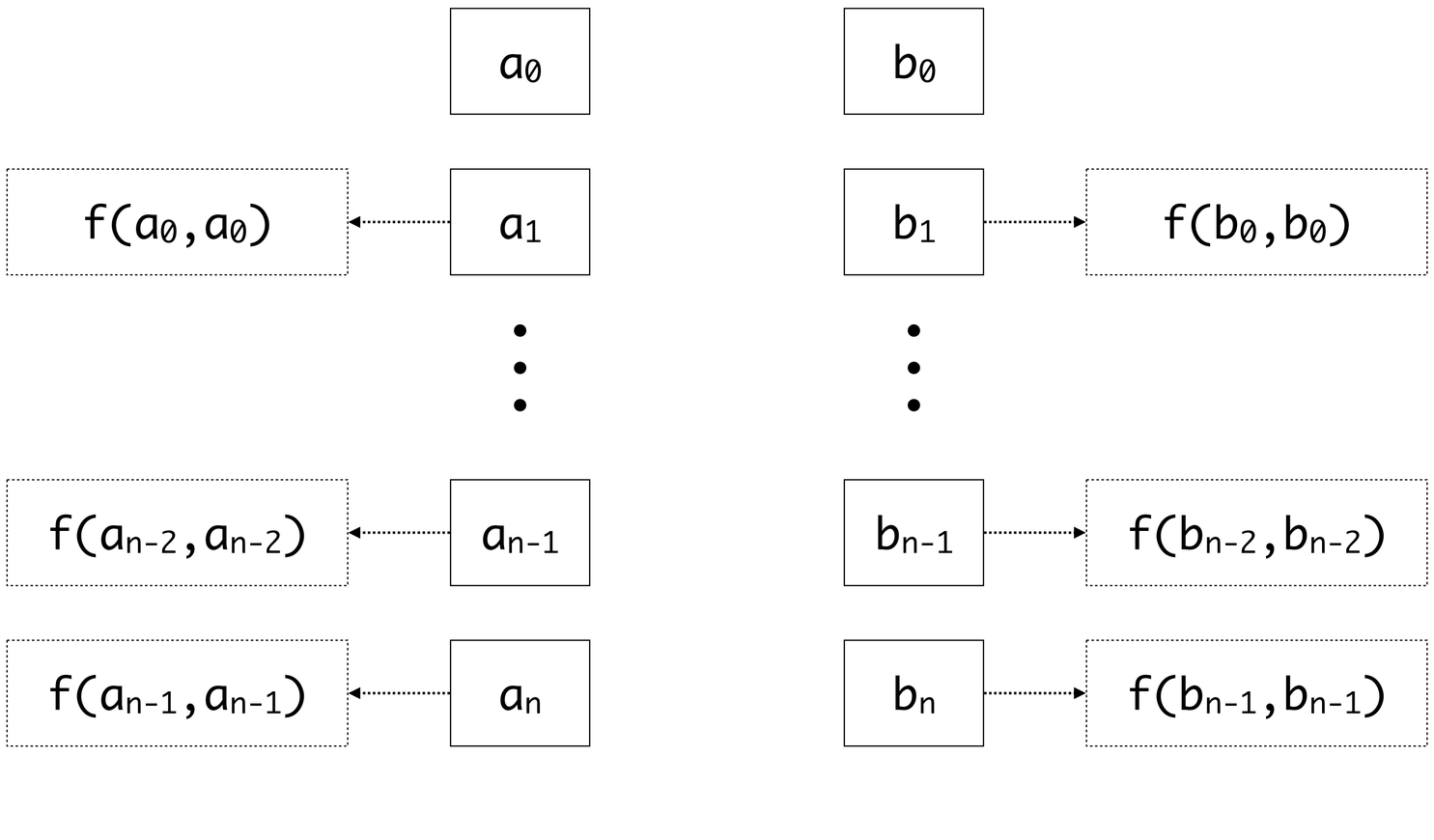
$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

The Complex Case

$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$



$a_n == b_n$

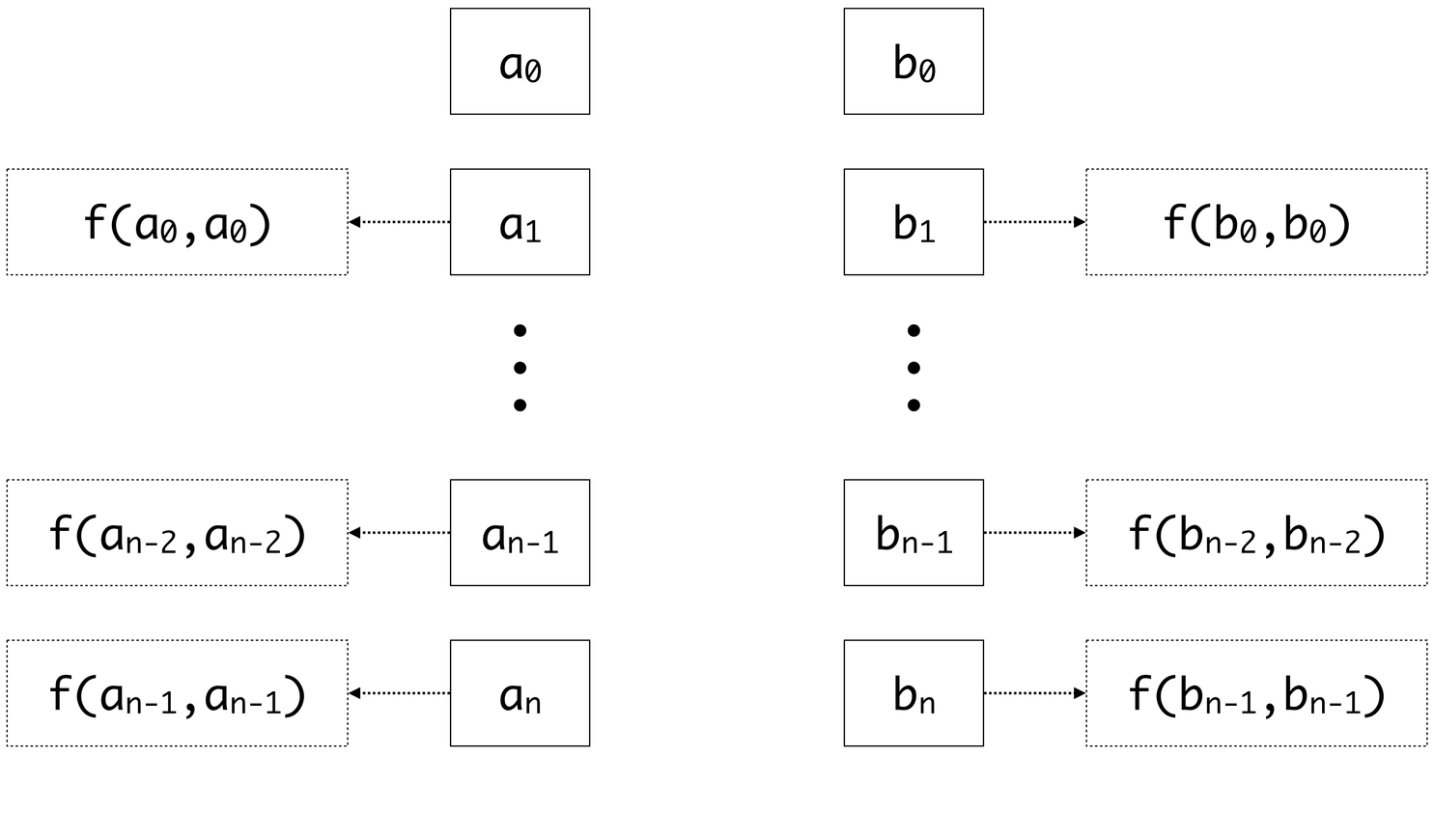
$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$

$a_{n-1} == b_{n-1}$ $\quad$ $a_{n-1} == b_{n-1}$

$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$

The Complex Case

$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$



$a_n == b_n$

$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$

$a_{n-1} == b_{n-1}$ $\quad$ $a_{n-1} == b_{n-1}$

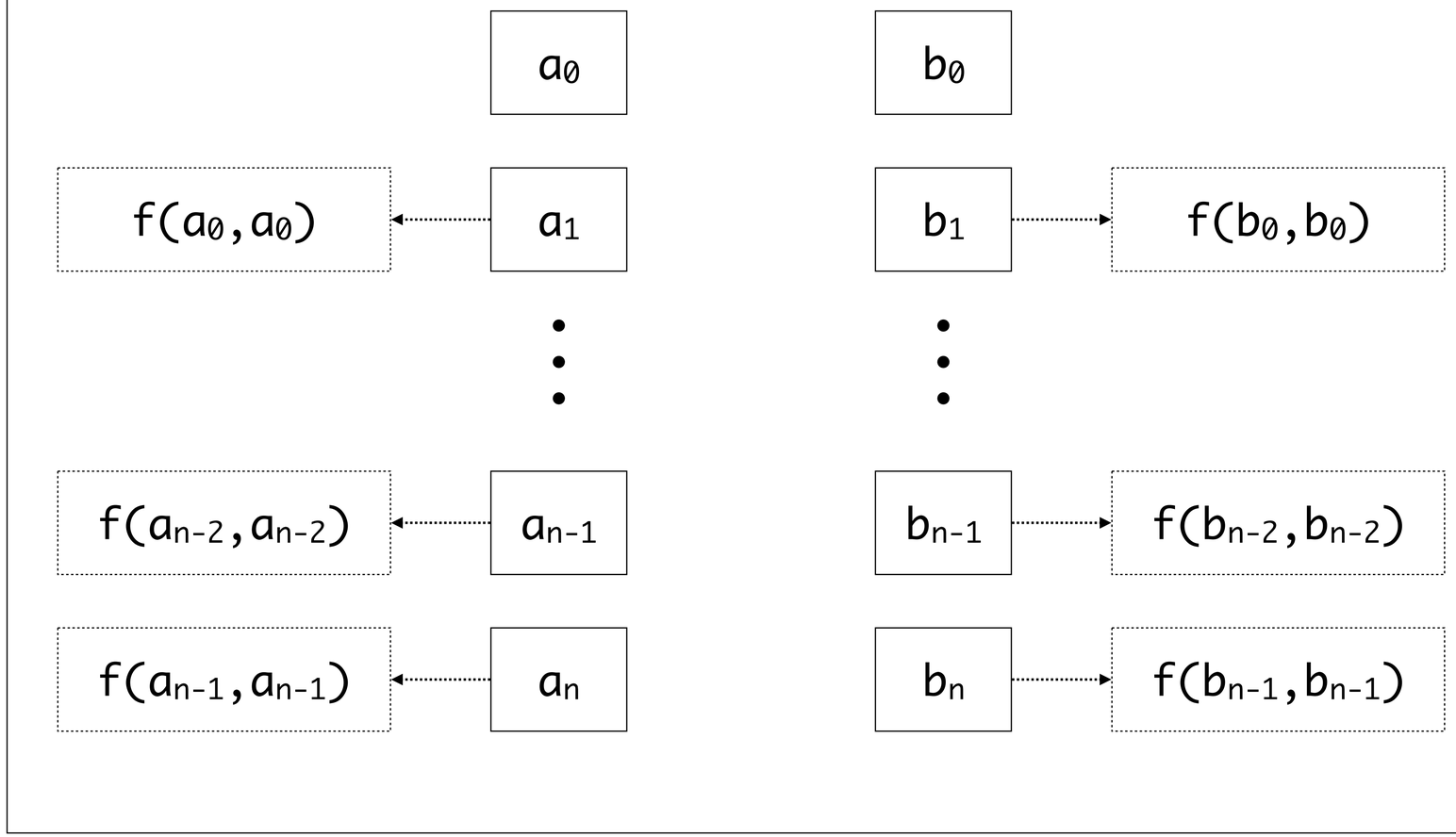
$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$

$\vdots$

$a_1 == b_1$ $\quad$ $a_1 == b_1$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

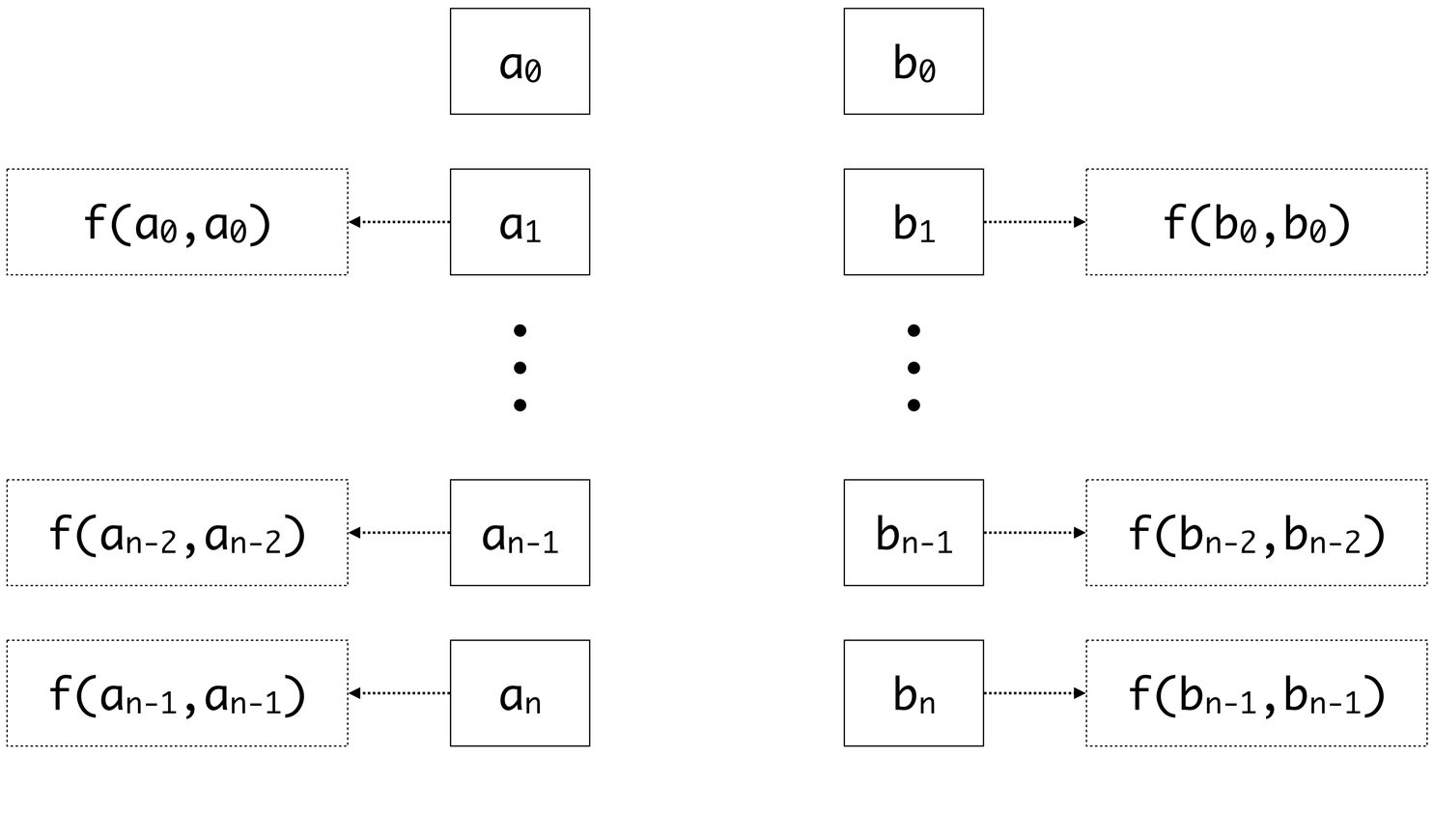
$$\vdots$$

$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

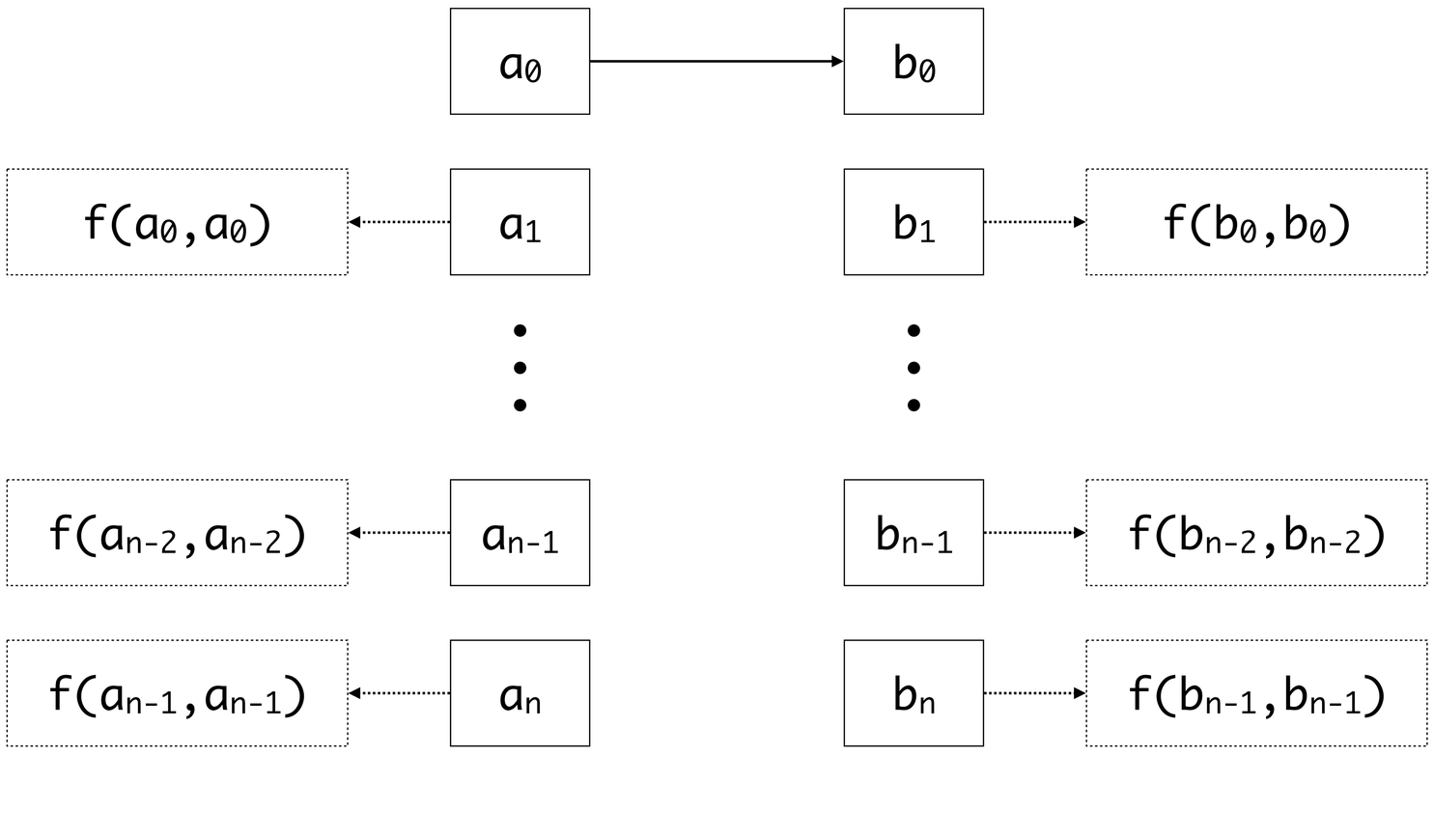
$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

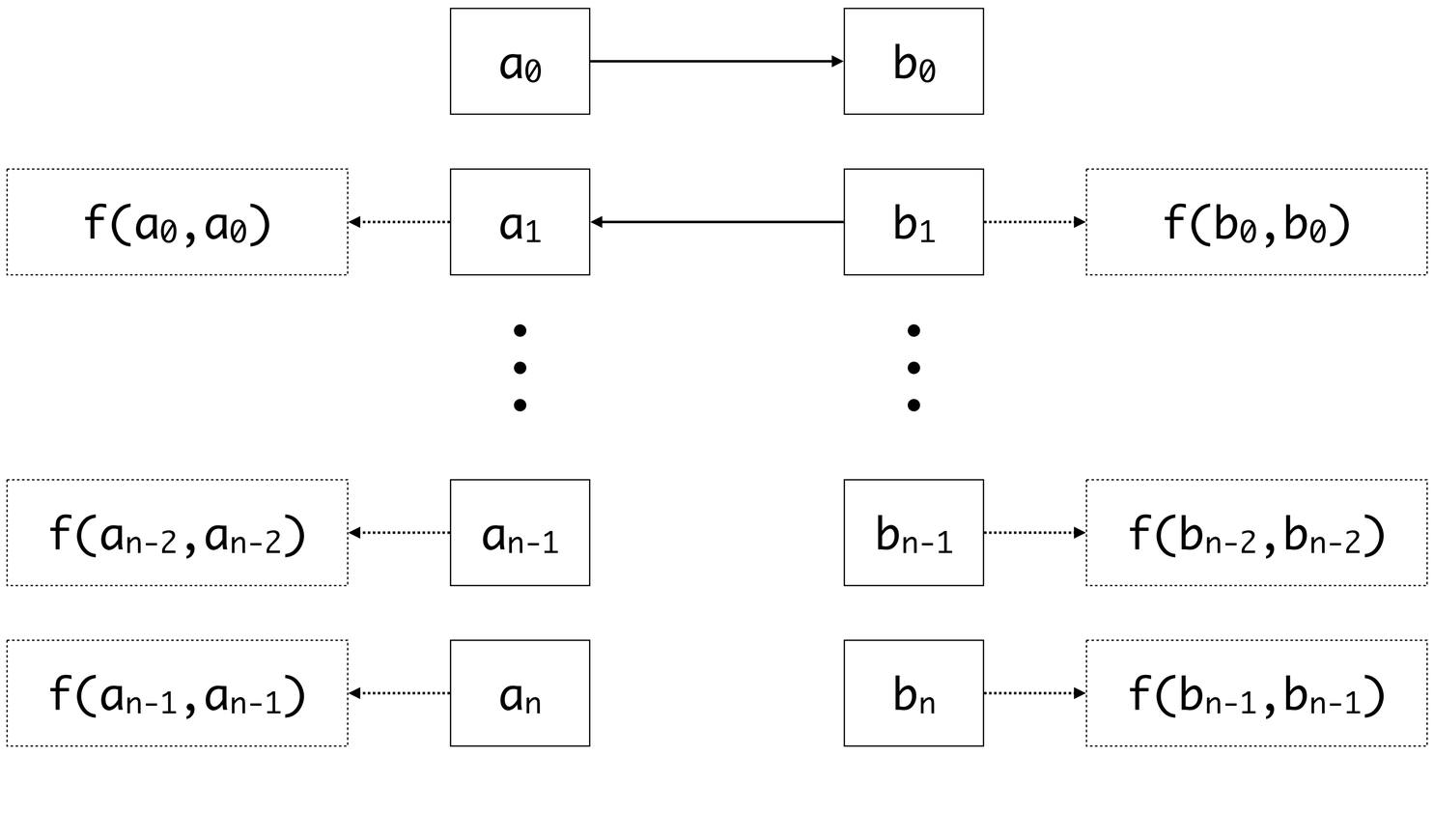
$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

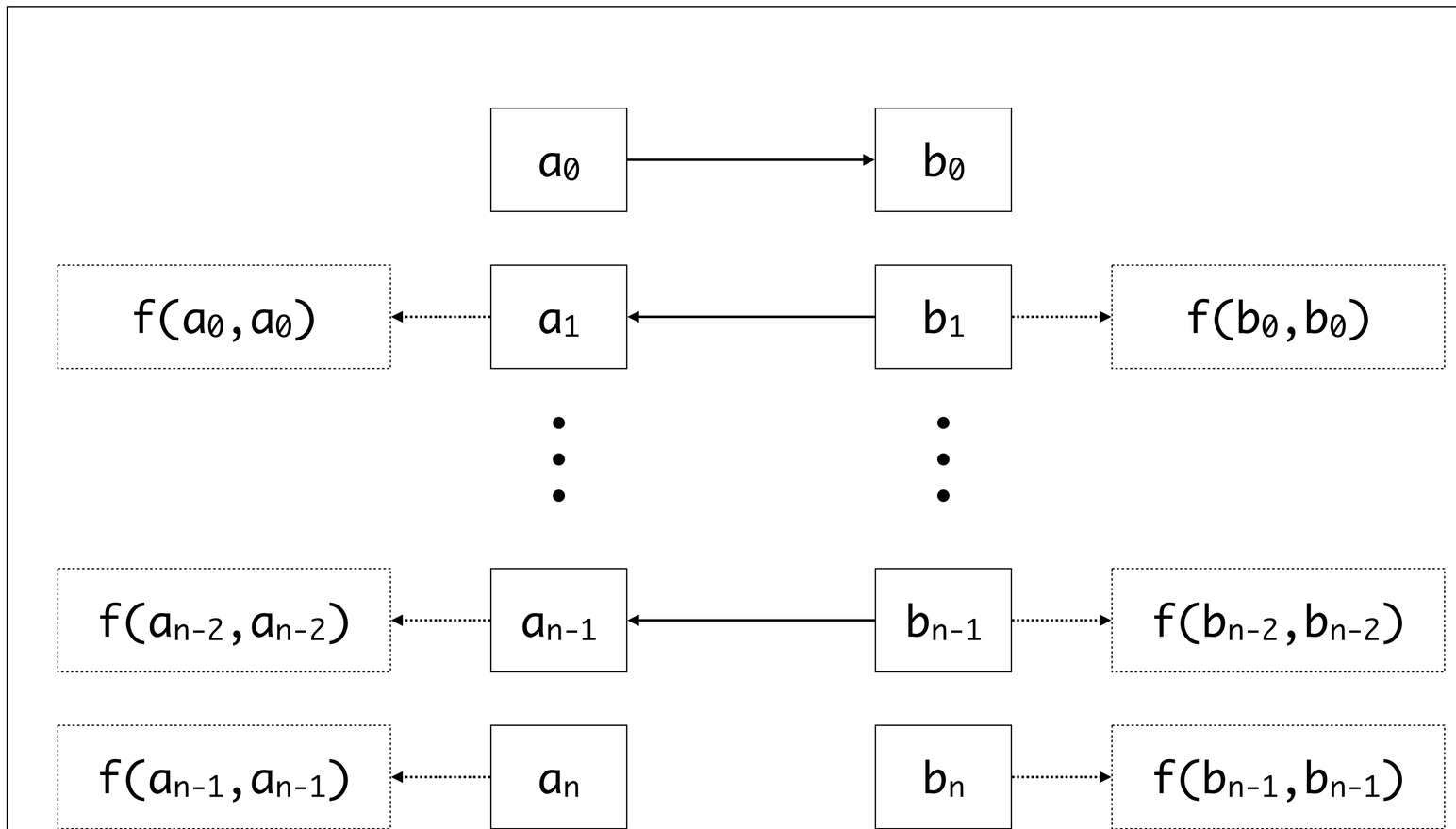
$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

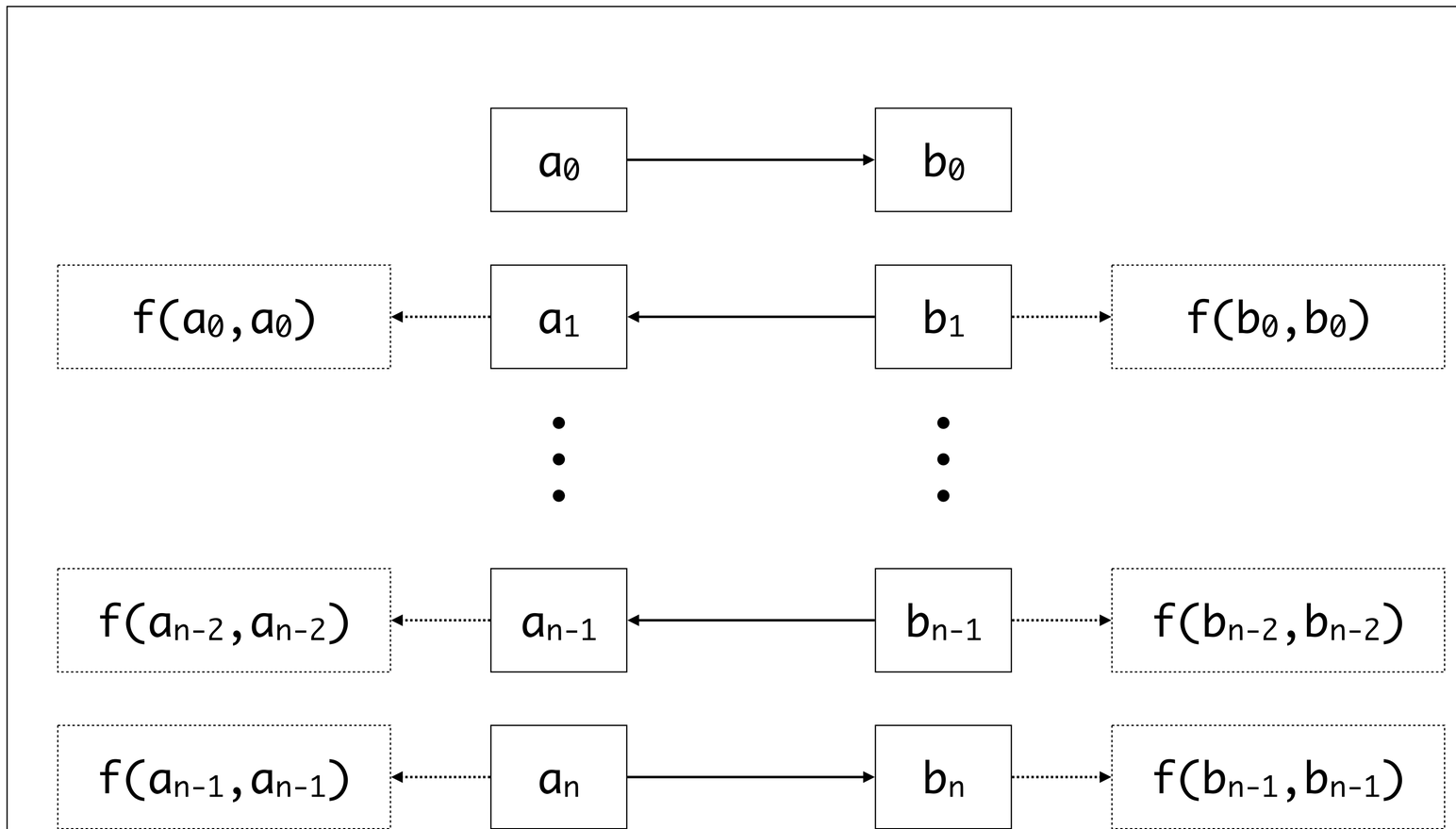
$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

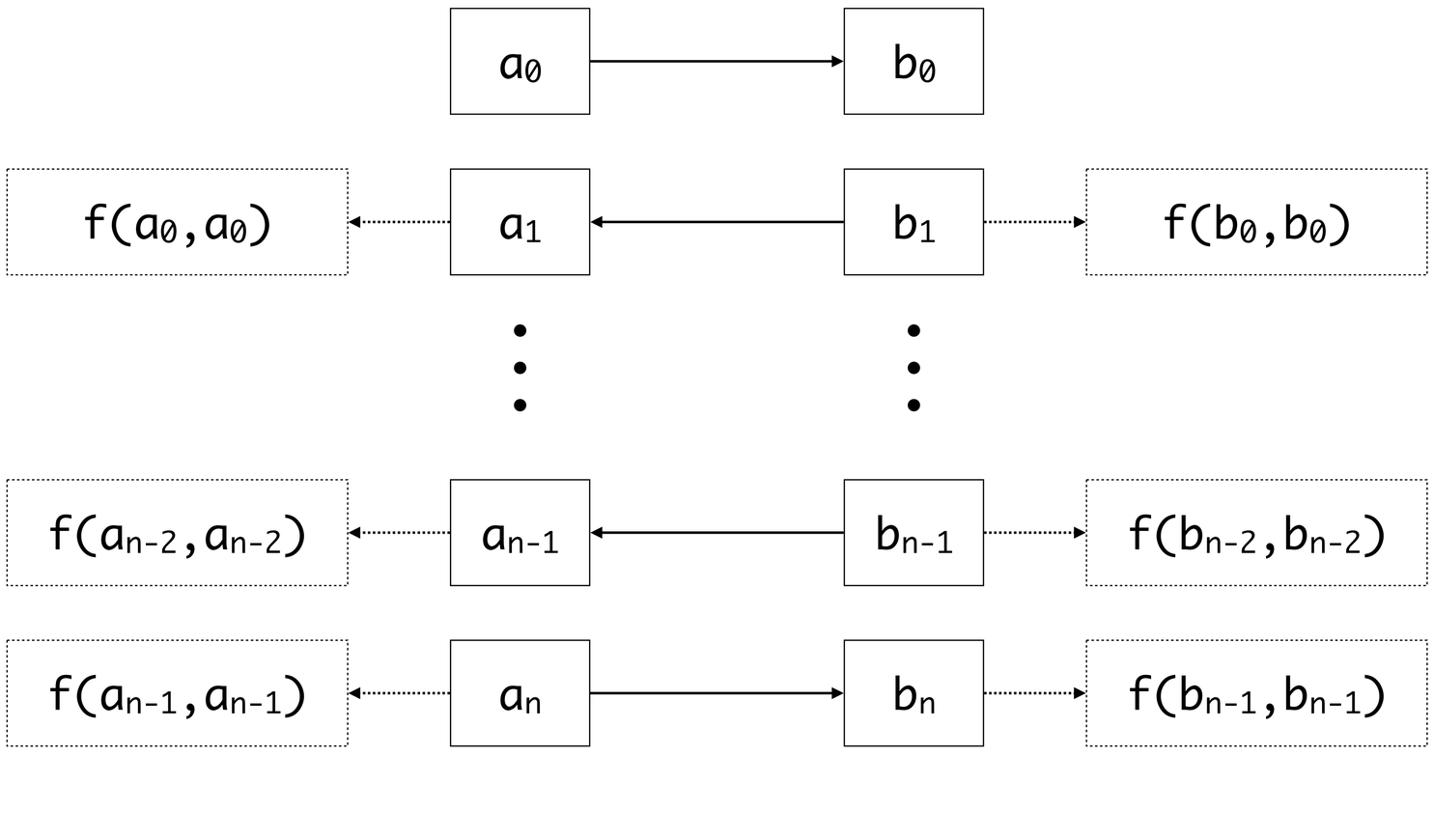
$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

$$a_1 == b_1 \qquad a_1 == b_1$$

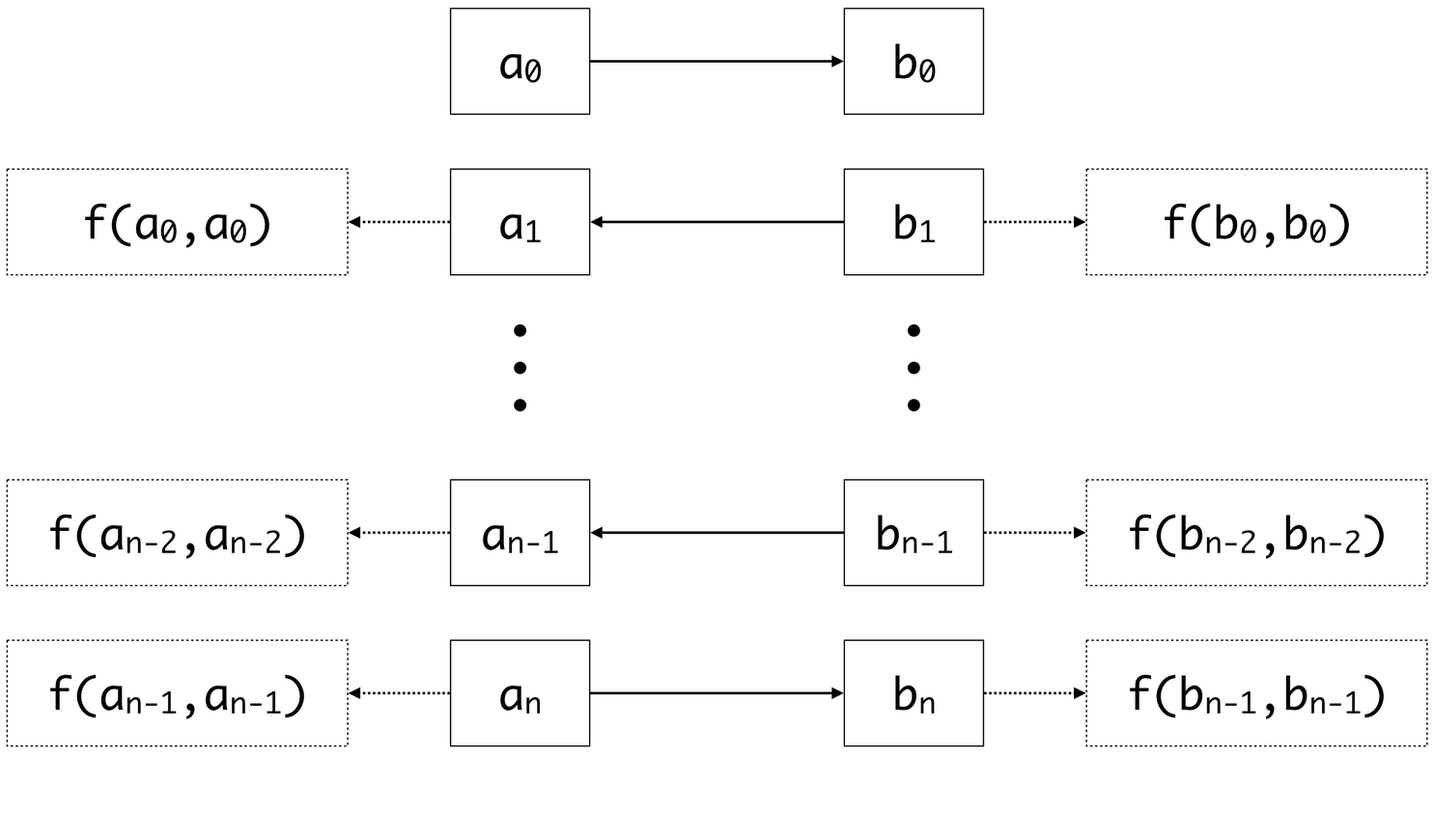
$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

How about occurrence checks?

The Complex Case

$$h(a_1, \ldots, a_n, f(b_0,b_0), \ldots, f(b_{n-1},b_{n-1}), a_n) == h(f(a_0,a_0), \ldots, f(a_{n-1},a_{n-1}), b_1, \ldots, b_{n-1}, b_n)$$



$$a_n == b_n$$

$$f(a_{n-1},a_{n-1}) == f(b_{n-1},b_{n-1})$$

$$a_{n-1} == b_{n-1} \qquad a_{n-1} == b_{n-1}$$

$$f(a_{n-2},a_{n-2}) == f(b_{n-2},b_{n-2})$$

$$\vdots$$

$$a_1 == b_1 \qquad a_1 == b_1$$

$$f(a_0,a_0) == f(b_0,b_0)$$

$$a_0 == b_0 \qquad a_0 == b_0$$

How about occurrence checks? Postpone!

Union-Find

Main idea

- Represent unifier as graph
- One variable represent equivalence class
- Replace substitution by union & find operations
- Testing equality becomes testing node identity

Optimizations

- Path compression make recurring lookups fast
- Tree balancing keeps paths short

Complexity

- Linear in space and almost linear in time (technically inverse Ackermann)
- Easy to extract triangular unifier from graph
- Postpone occurrence checks to prevent traversing (potentially) large terms

```
Martelli, Montanari. An Efficient Unification Algorithm. TOPLAS, 1982
```

Error Reporting

Reporting Type Errors

Type errors

- Types are supposed to prevent us from making mistakes
- Error messages are supposed to help us fix mistakes
- This does not always work out so well
  - Wrong error location
  - Unclear error message
  - Not always clear where inferred types come from
- Users expect concise, informative messages, at the right place

Type errors = Unsatisfiable constraints

- Constraint satisfaction is a binary property
- Language designers want to influence error messages

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_1 == FUN(ty_4, ty_5)
ty_2 == INT()
ty_3 == INT()
ty_5 == INT()
ty_4 == STRING()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
["Hello!"]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3)
ty_3 == INT()
ty_4 == STRING()
ty_5 == INT()
ty_2 == INT()
ty_1 == FUN(ty_4, ty_5)

ty_1 == FUN(ty_2, ty_3)
ty_1 == FUN(ty_4, ty_5)
ty_2 == INT()
ty_3 == INT()
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
          [x]
      [_ + _]
        [f _]
   [“Hello!”]
        [_ * 2]
```

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()
ty_2 == INT()
ty_1 == FUN(ty_2, ty_3)
ty_3 == INT()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

ty_1 == FUN(ty_4, ty_5) ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_2, ty_3) ✗
ty_3 == INT()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
["Hello!"]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

ty_1 == FUN(ty_4, ty_5) ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_2, ty_3) ✗
ty_3 == INT()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

ty_1 == FUN(ty_4, ty_5) ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_2, ty_3) ✗
ty_3 == INT()

Error Example

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
[x]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

ty_1 == FUN(ty_2, ty_3) ✓
ty_1 == FUN(ty_4, ty_5) ✓
ty_2 == INT() ✓
ty_3 == INT() ✓
ty_5 == INT() ✓
ty_4 == STRING() ✗

ty_1 == FUN(ty_2, ty_3) ✓
ty_3 == INT() ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_4, ty_5) ✗

ty_1 == FUN(ty_4, ty_5) ✓
ty_4 == STRING() ✓
ty_5 == INT() ✓
ty_2 == INT() ✓
ty_1 == FUN(ty_2, ty_3) ✗
ty_3 == INT()

Order of constraint solving is relevant!

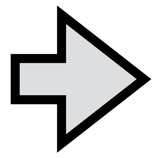
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
    [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



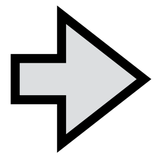
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
    [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)

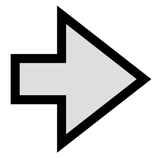
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
     [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()

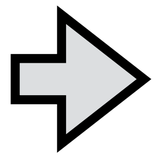
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
     [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()

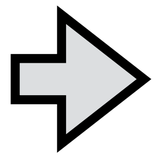
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

```
ty1 == FUN(ty2, ty3)
ty2 == INT()
ty3 == INT()
ty1 == FUN(ty4, ty5)
ty4 == STRING()
ty5 == INT()
```

```
[function f(x)]
[x + _]
[_ + _]
[f _]
[“Hello!”]
[_ * 2]
```

```
ty1 == FUN(ty2, ty3)
ty2 == INT()
ty3 == INT()
ty1 == FUN(ty4, ty5)
ty4 == STRING()
ty5 == INT()
```



```
ty1 == FUN(ty2, ty3)
ty2 == INT()

ty1 == FUN(ty4, ty5)
```

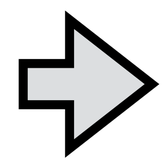
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
    [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()

ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()

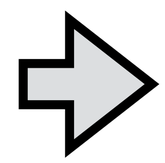
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
    [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()

ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()

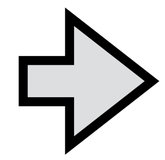
Conflict Sets

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

```
[function f(x)]
        [x + _]
        [_ + _]
          [f _]
   [“Hello!”]
          [_ * 2]
```

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()

ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()

A conflict set is an inconsistent subset of the constraint set

Conflict Sets and Errors

Conflict Sets

- Sets of constraint that are inconsistent together
- Multiple conflict sets might appear in one constraint set
- Conflict sets can intersect

How to find conflict sets?

- Find minimal sets that are in conflict
  - Combinatorial explosion (Min-SAT)
- Approximate conflict sets during constraint solving
  - Conflict set may be too large

Where to report errors?

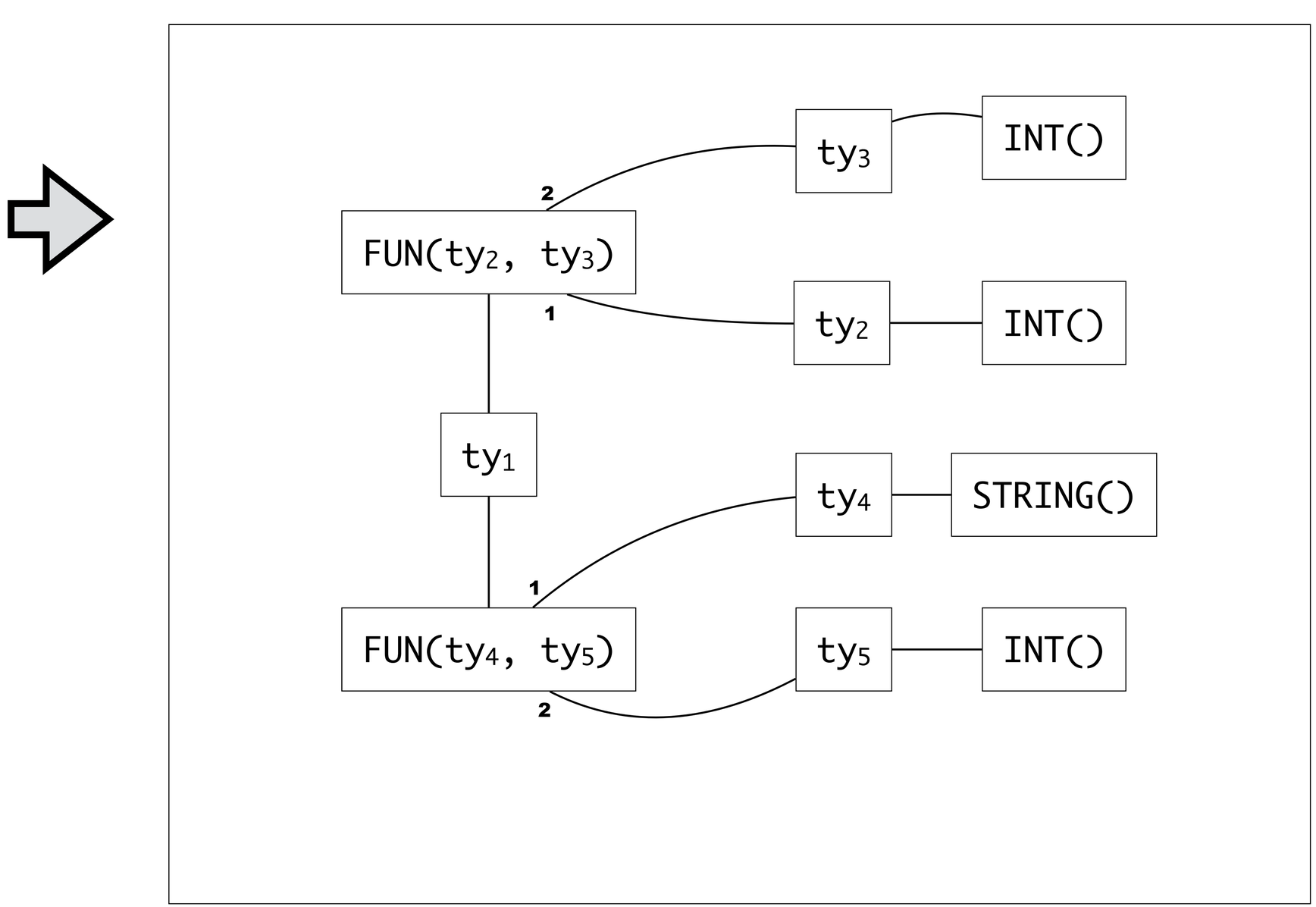
- Report the conflicting program slice (using all conflicting constraints)
- Report on a few constraints in the conflict set
  - Remove constraints until inconsistency disappears
  - Use heuristics to efficiently select constraints

```
1. let
2.   function f(x) =
3.     x + 1
4. in
5.   (f "Hello!") * 2
```

There is not always a *best* place to report the error!

Constraint Graphs

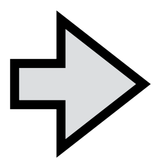
ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()

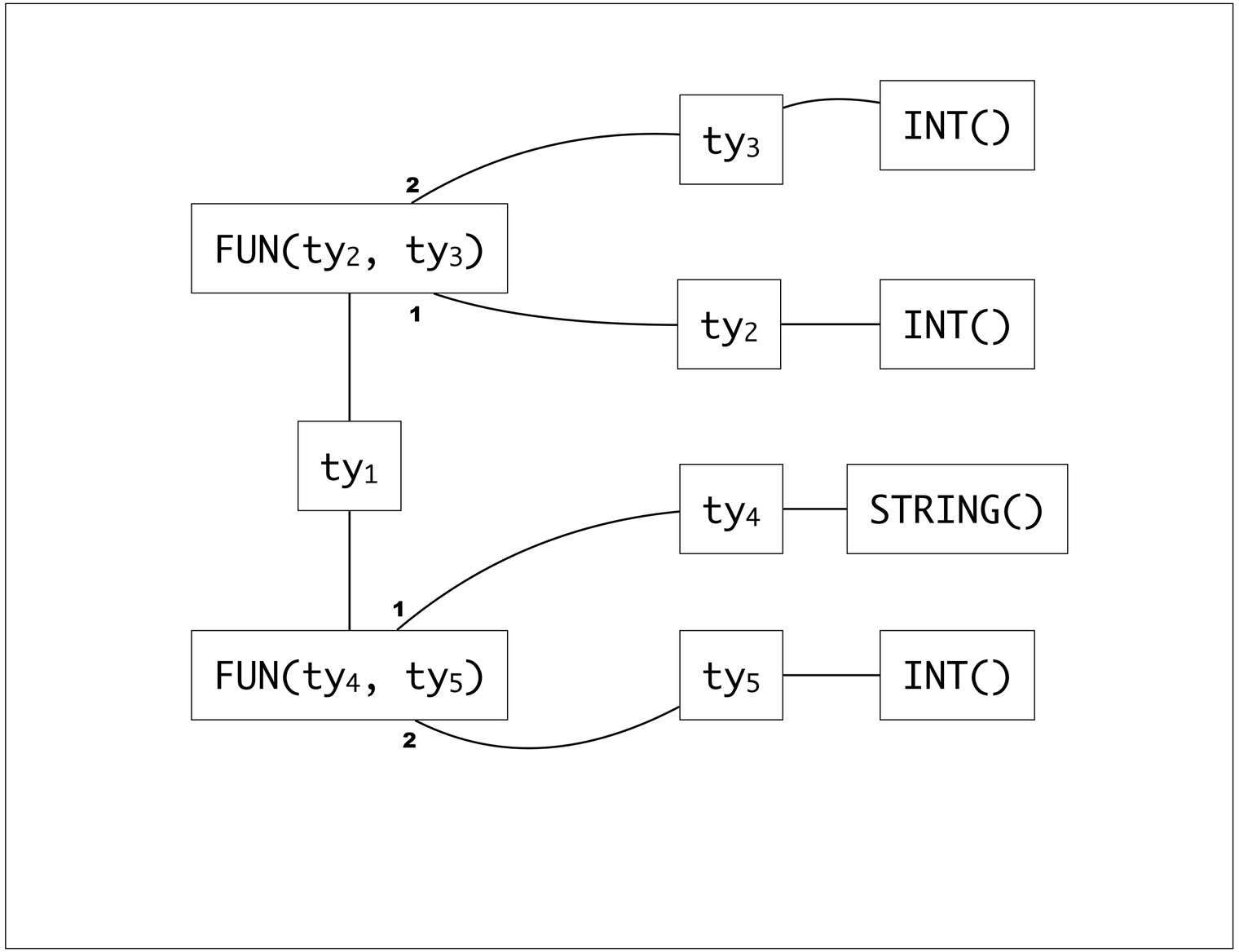


Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



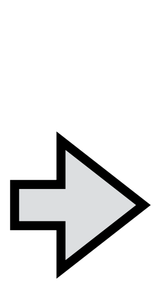


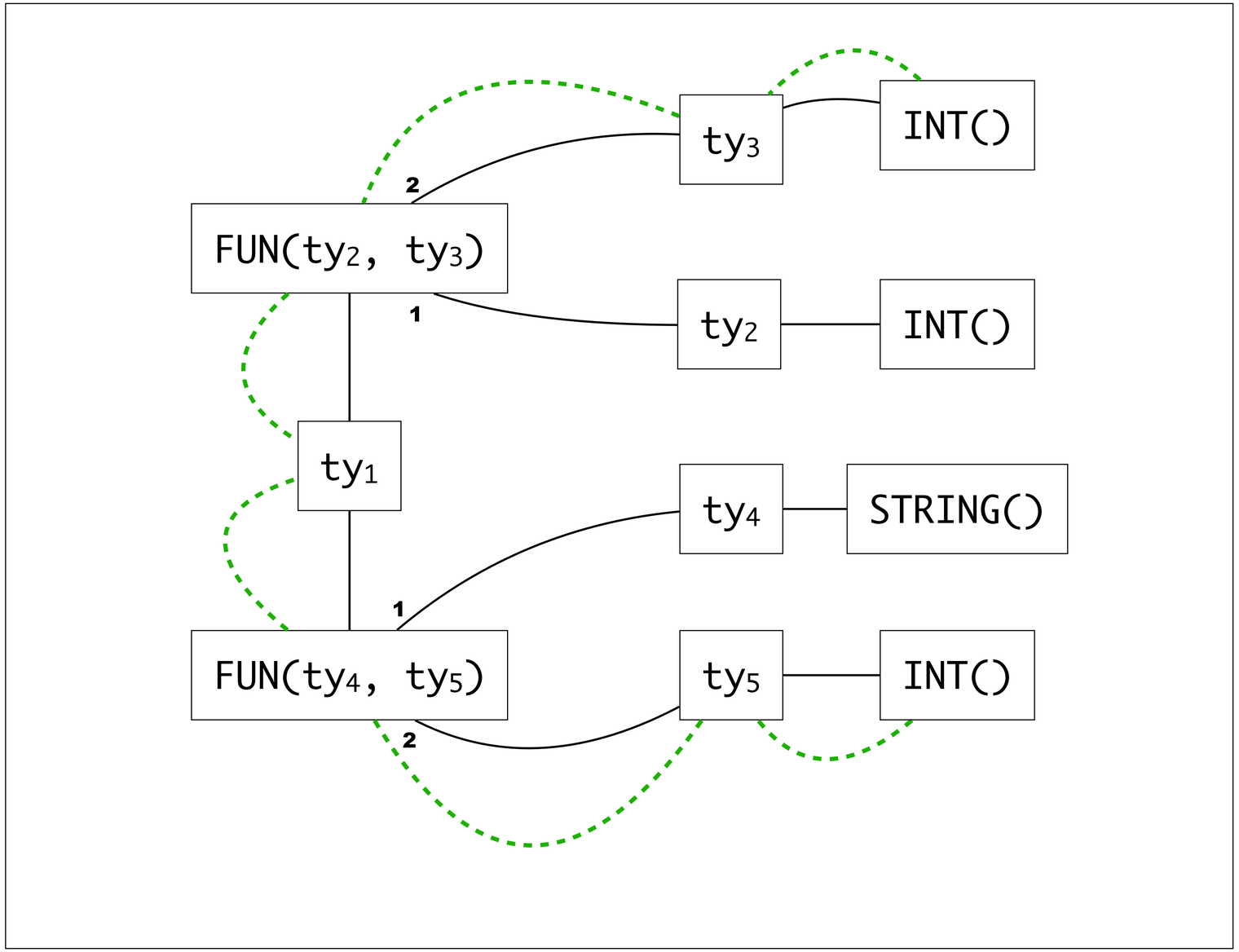
- Consider paths between non-variable terms

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



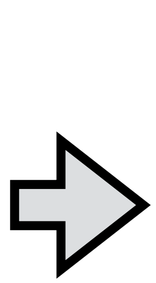


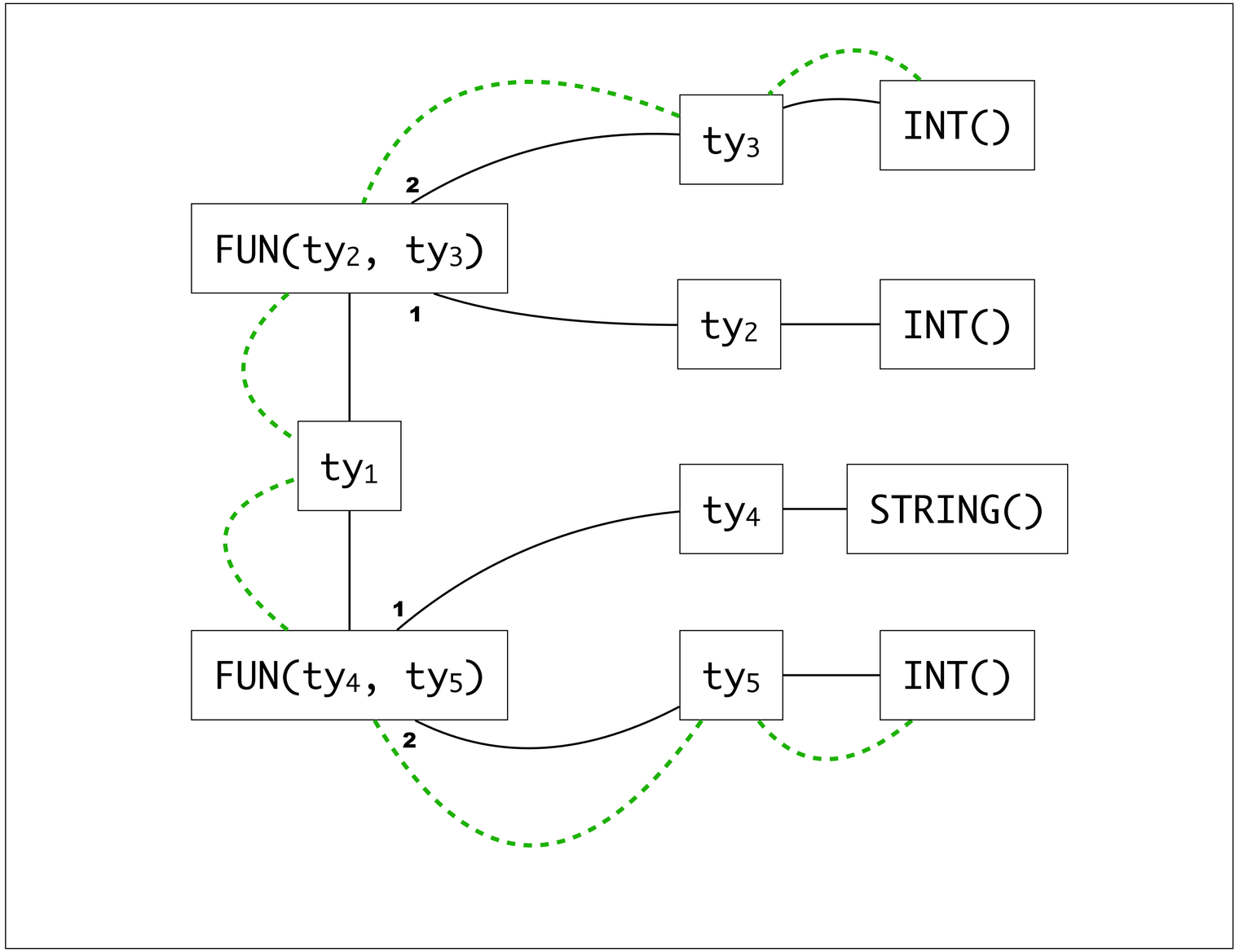
- Consider paths between non-variable terms

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



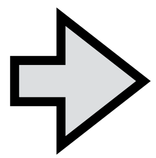


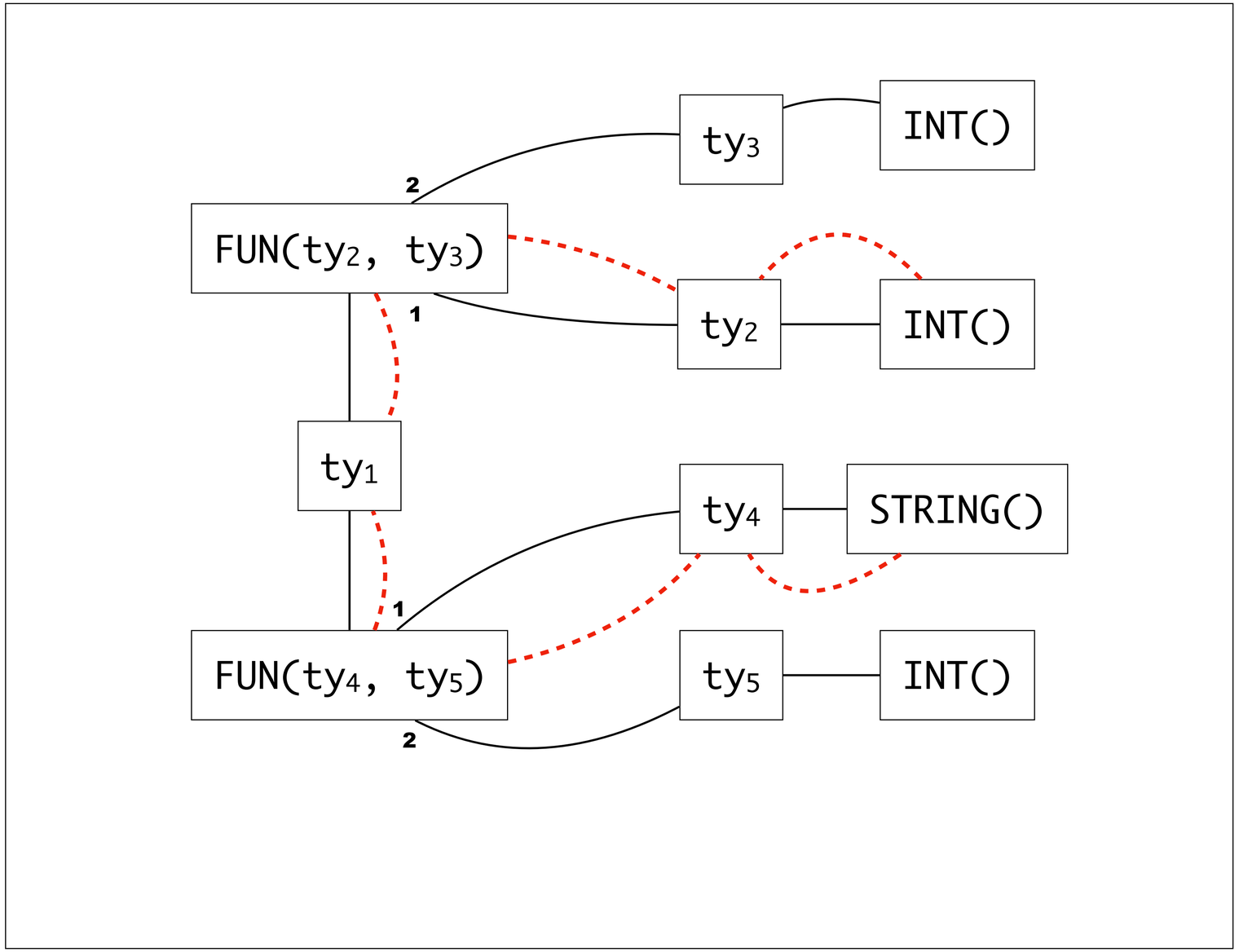
- Consider paths between non-variable terms
- Trust edges involved in many correct paths

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



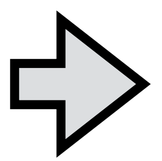


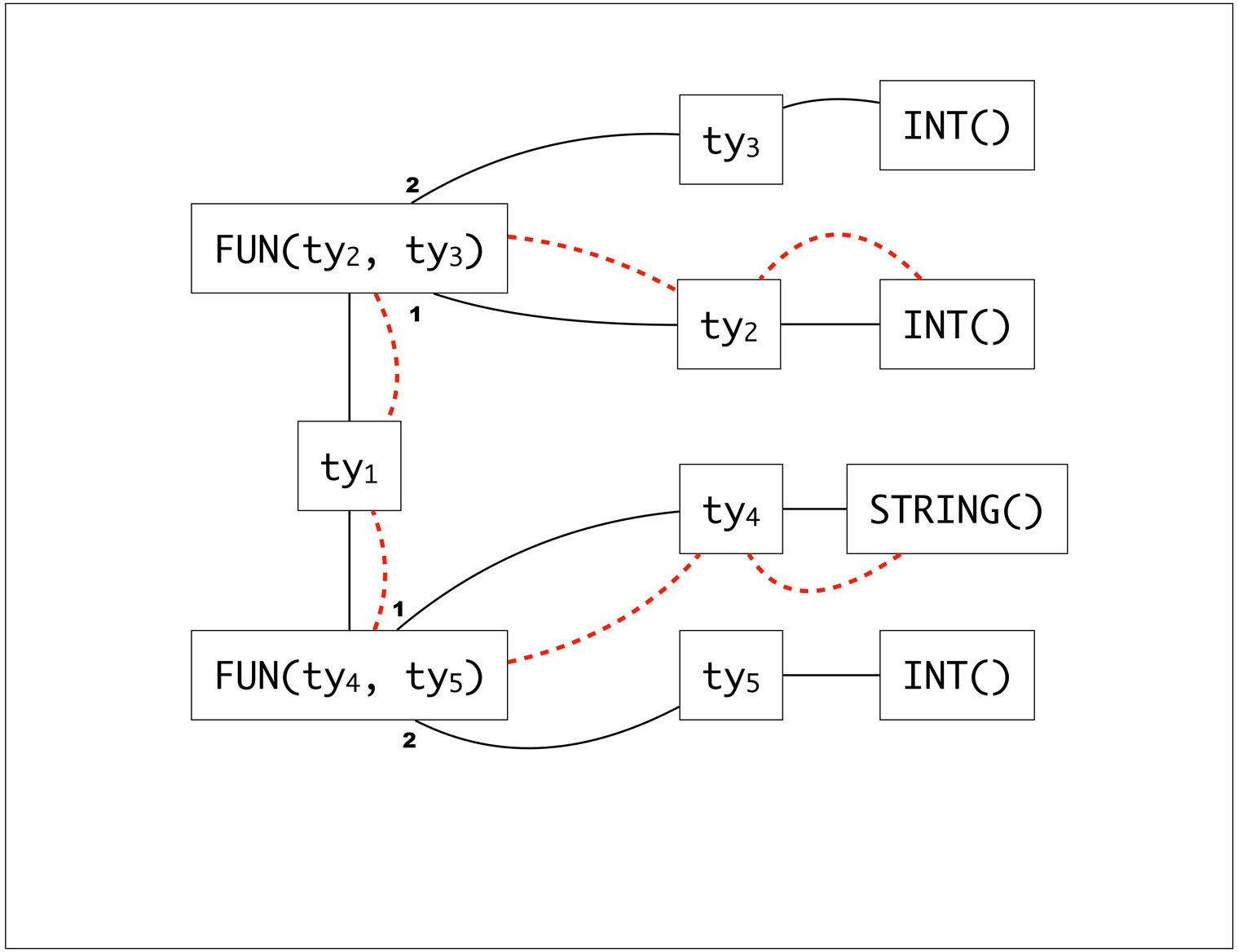
- Consider paths between non-variable terms
- Trust edges involved in many correct paths

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



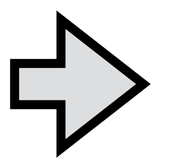


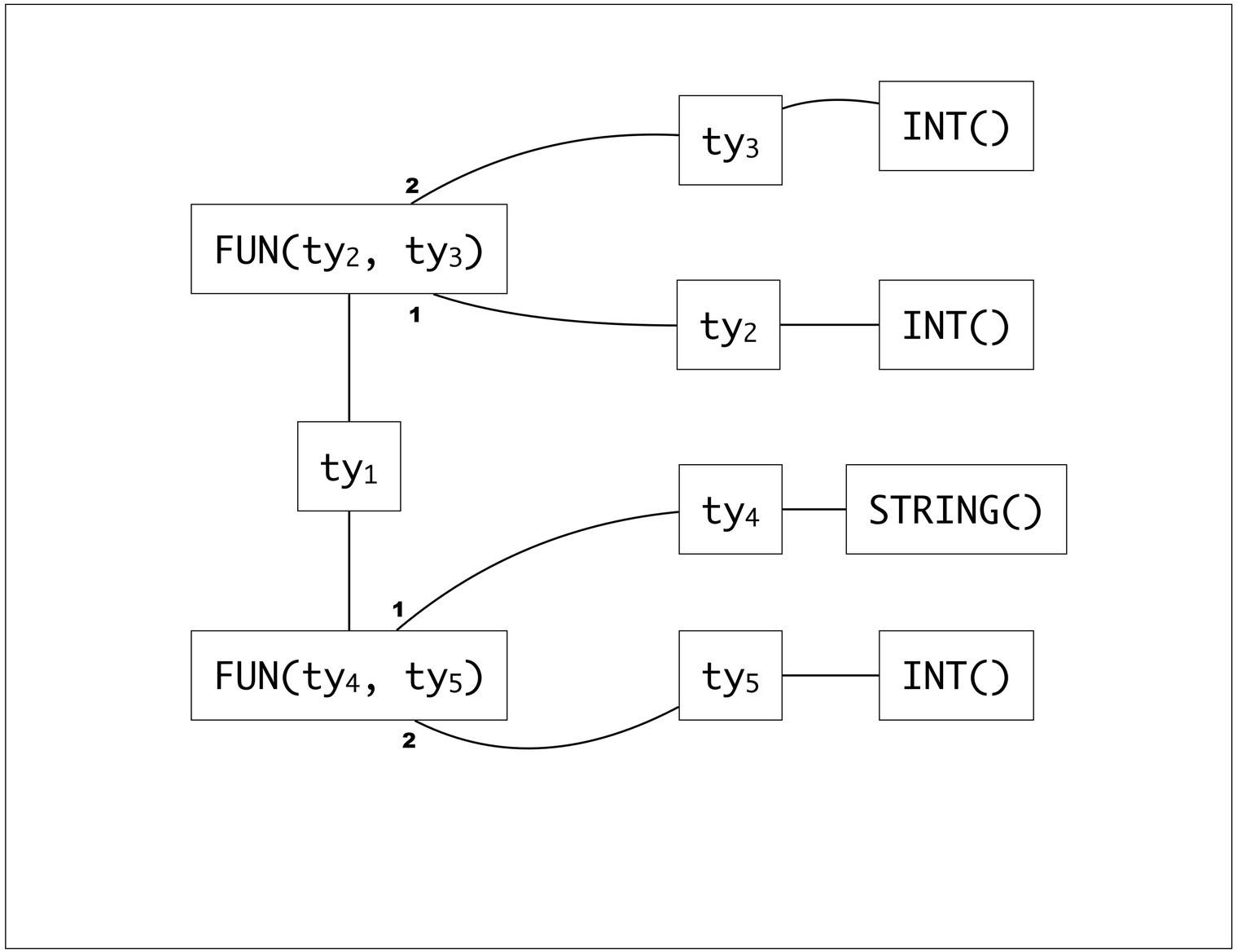
- Consider paths between non-variable terms
- Trust edges involved in many correct paths
- Suspect edges involved in many wrong paths

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Constraint Graphs

ty_1 == FUN(ty_2, ty_3)
ty_2 == INT()
ty_3 == INT()
ty_1 == FUN(ty_4, ty_5)
ty_4 == STRING()
ty_5 == INT()



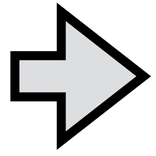


- Consider paths between non-variable terms
- Trust edges involved in many correct paths
- Suspect edges involved in many wrong paths
- Restricts constraint langauge and solver implementation

Zhang, Meyers. Toward General Diagnosis of Static Errors. POPL'14

Use approximated conflict sets

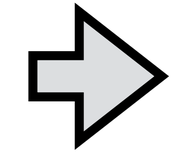
1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()

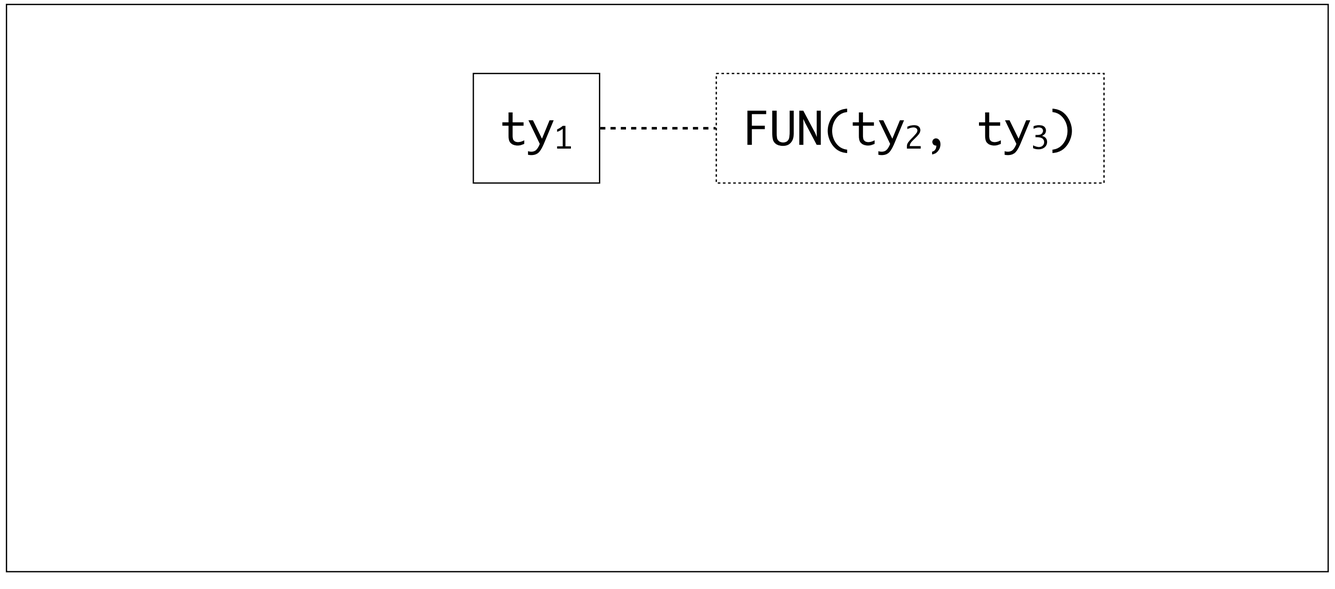


Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()

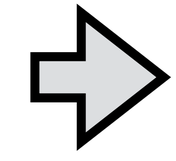


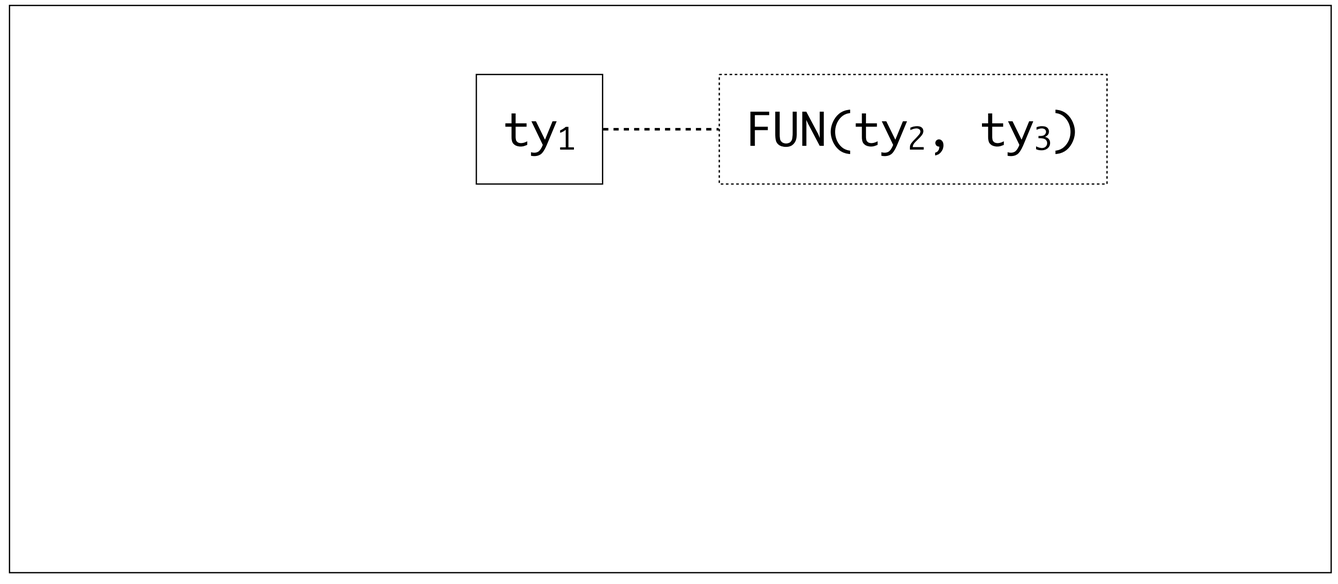


Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



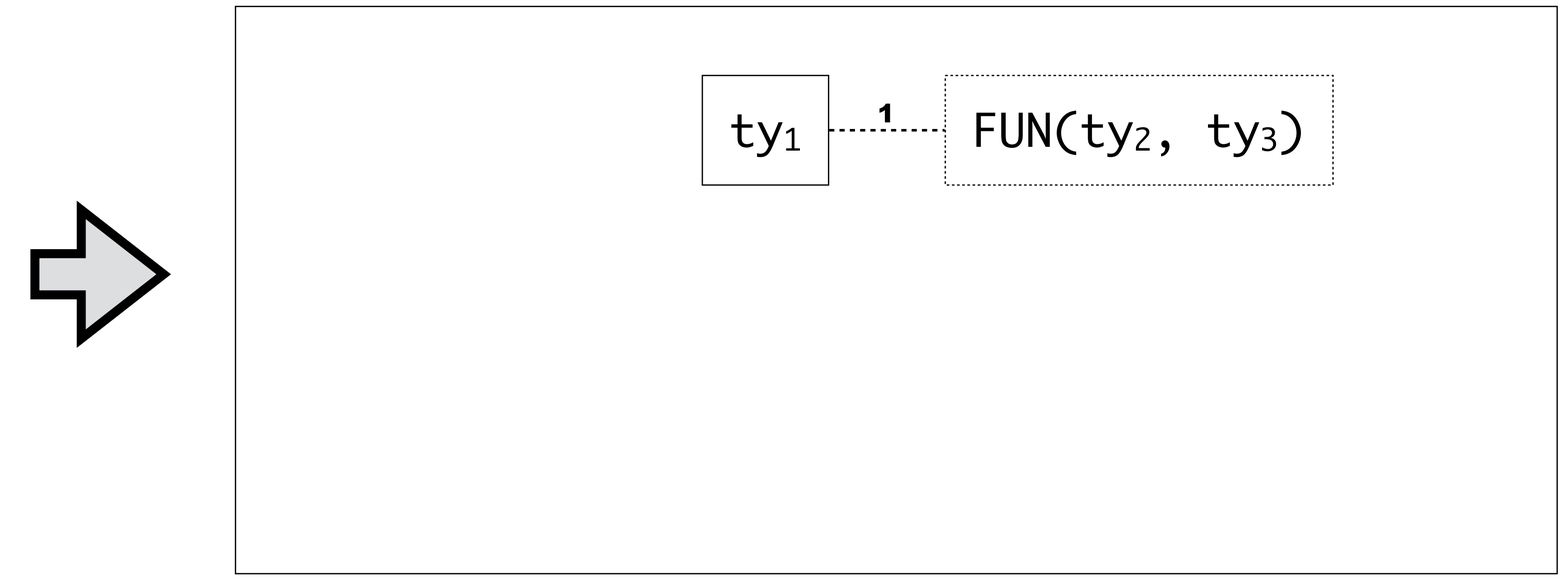


- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()

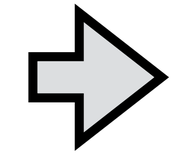


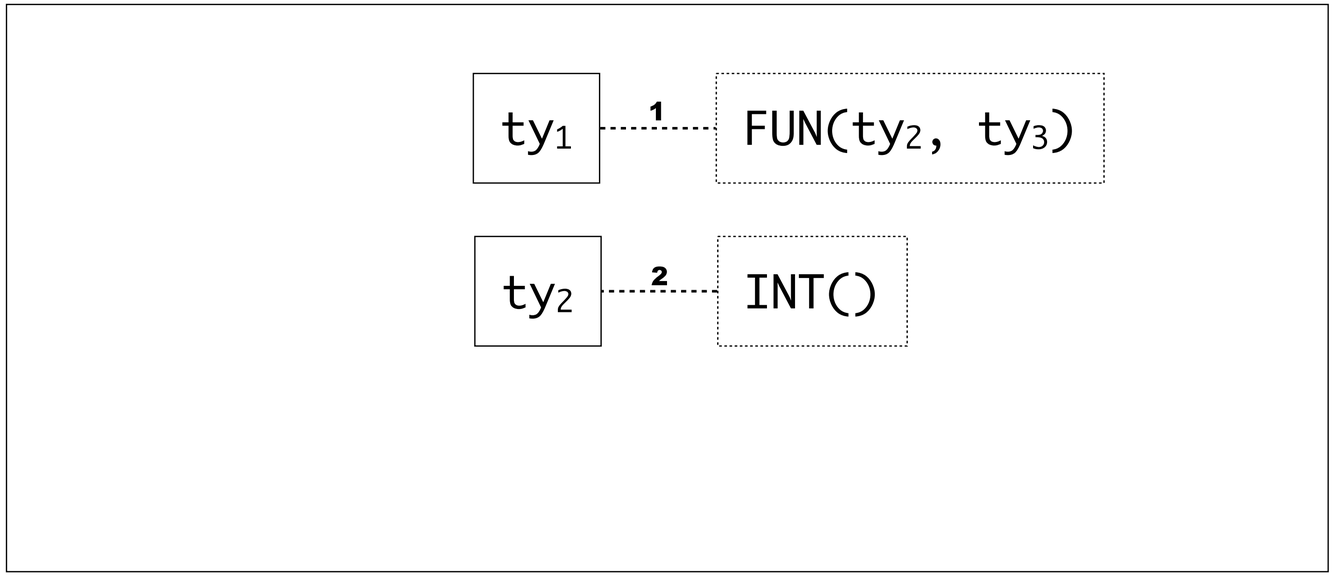
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



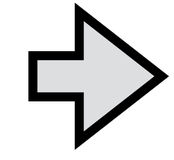


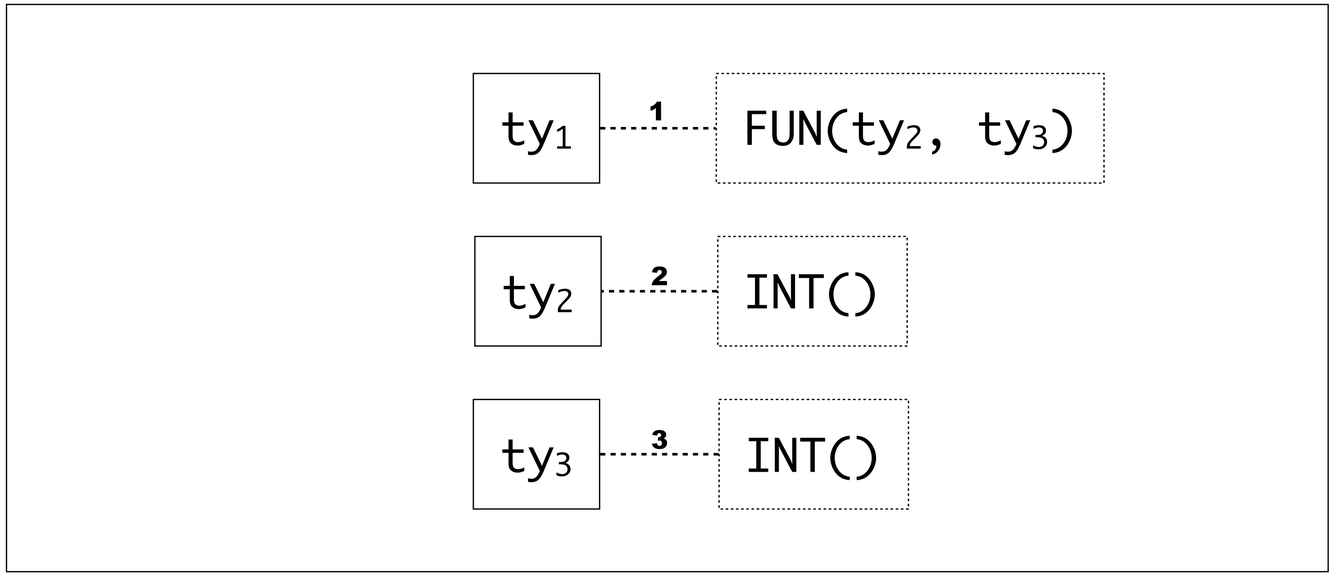
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



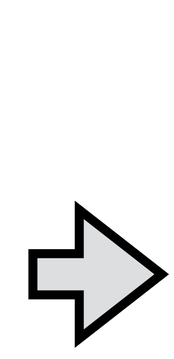


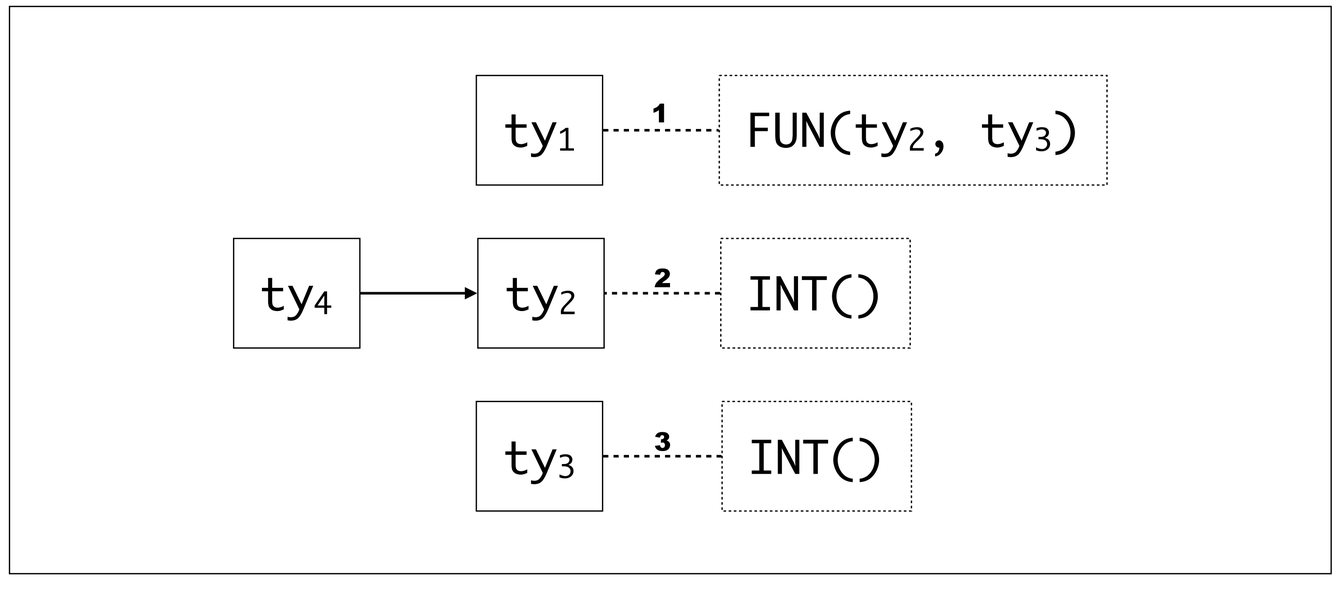
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



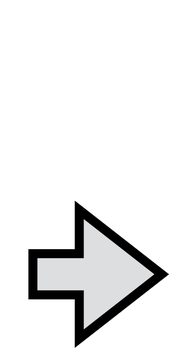


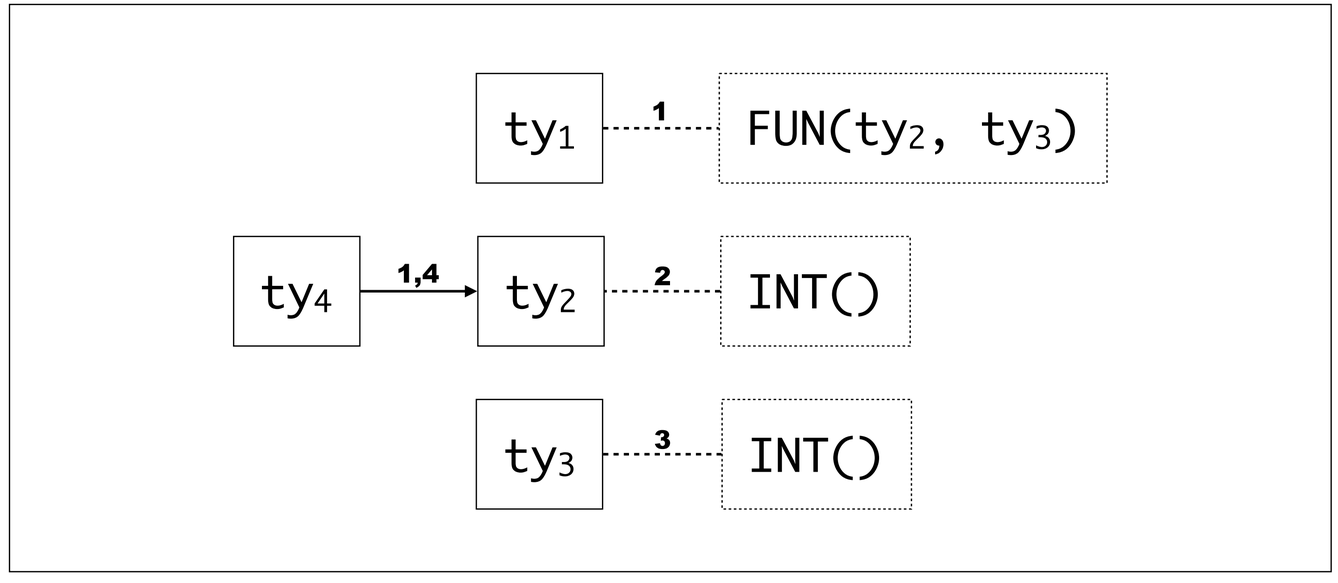
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



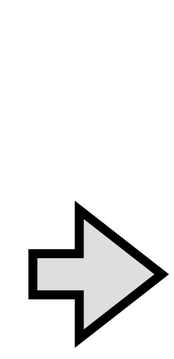


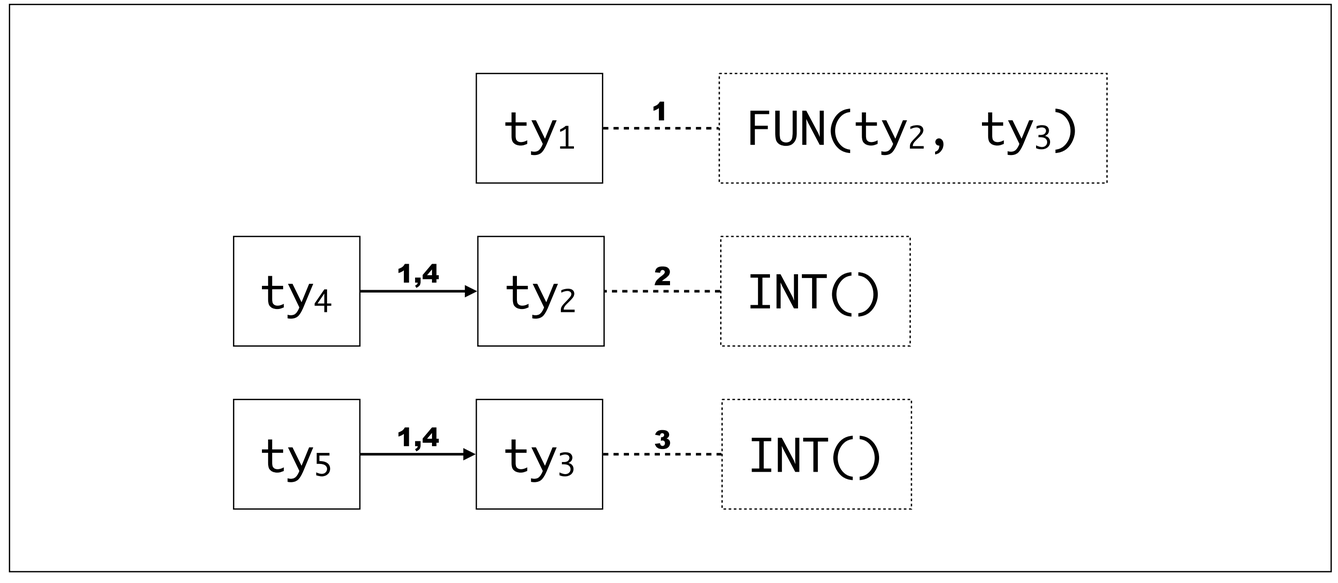
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



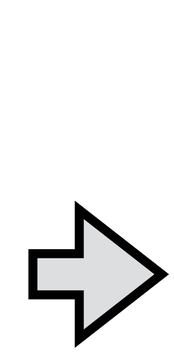


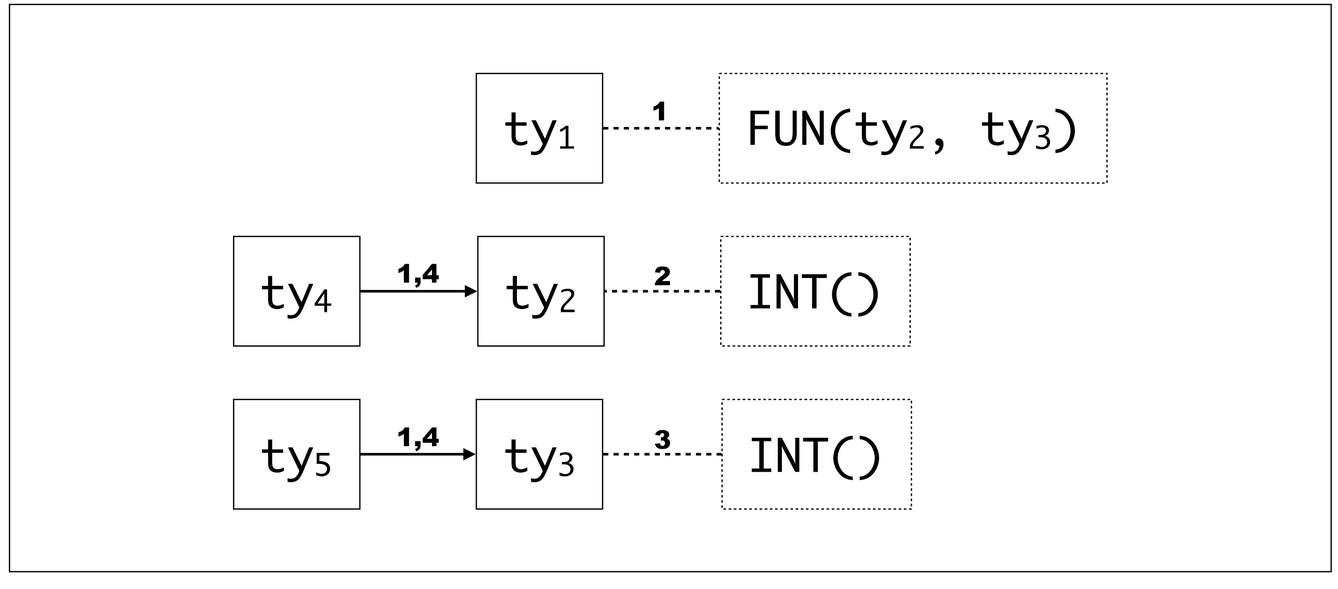
- Track constraints during unification

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()





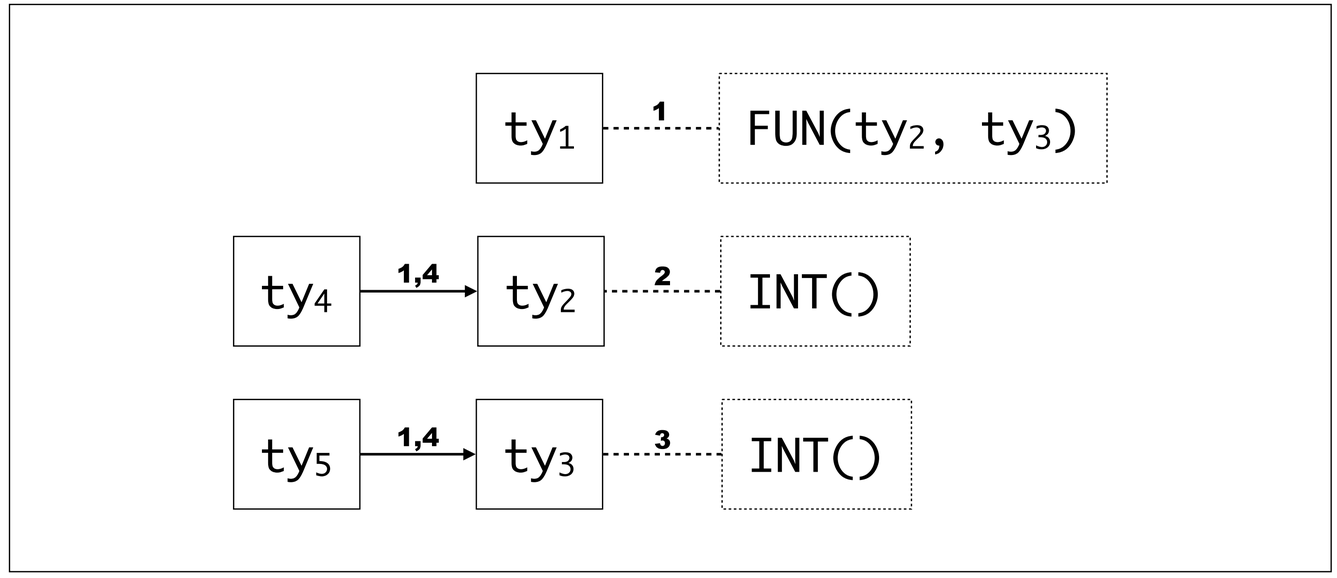
- Track constraints during unification

FIND(ty_4) = INT() because of {1,2,4}

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()



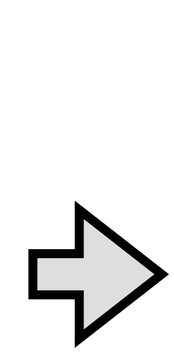
- Track constraints during unification

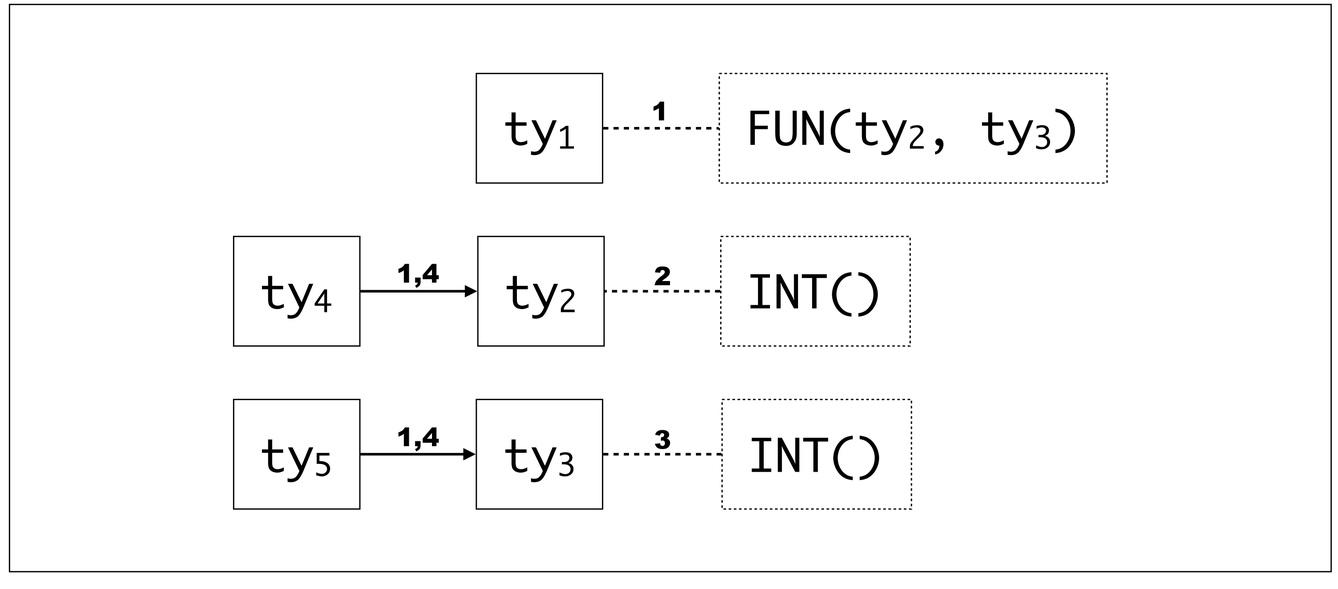
FIND(ty_4) = INT() because of {1,2,4}
constraint 5 gives conflict set {1,2,4,5}

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()





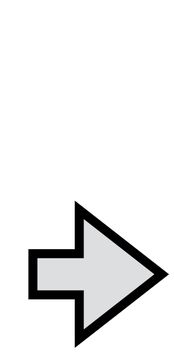
- Track constraints during unification
- Conflict set may be an over-approximation

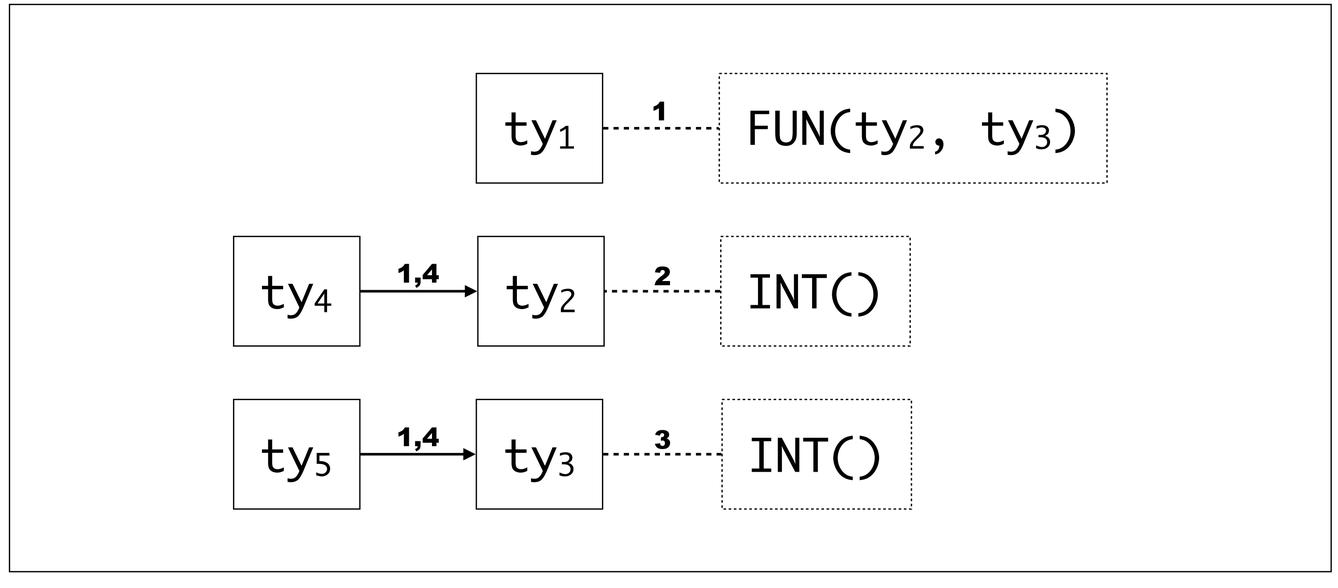
FIND(ty_4) = INT() because of {1,2,4}
constraint 5 gives conflict set {1,2,4,5}

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()





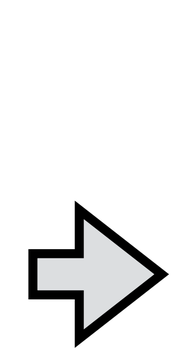
- Track constraints during unification
- Conflict set may be an over-approximation
- Iteratively drop constraints until inconsistency is gone

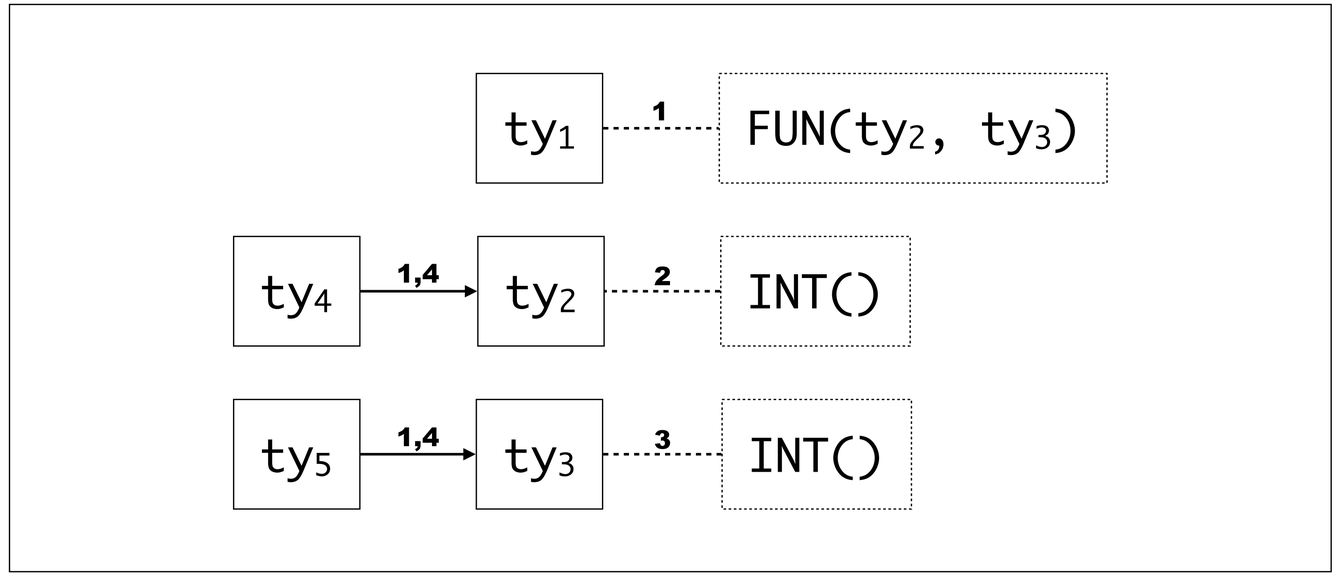
FIND(ty_4) = INT() because of {1,2,4}
constraint 5 gives conflict set {1,2,4,5}

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Use approximated conflict sets

1. ty_1 == FUN(ty_2, ty_3)
2. ty_2 == INT()
3. ty_3 == INT()
4. ty_1 == FUN(ty_4, ty_5)
5. ty_4 == STRING()
6. ty_5 == INT()





- Track constraints during unification
- Conflict set may be an over-approximation
- Iteratively drop constraints until inconsistency is gone
- Solver is used as an oracle, no requirements on its implementation

FIND(ty_4) = INT() because of {1,2,4}
constraint 5 gives conflict set {1,2,4,5}

Loncaric e.a. A practical framework for type inference error explanation. OOPSLA'16

Which constraint is to blame?

Error reporting from conflict sets

- Select constraints that are likely causes
- Few error messages is usually preferred over many
- Avoid combinatorial explosion by using heuristics

Language independent heuristics

- Select constraints shared between multiple conflict sets
- Select constraints coming from smaller program fragments
  - Function definition type comes from the (larger) body expression
  - Function use type comes from the (smaller) use expression

Language specific heuristics

- Select constraint based on weights assigned in specification
  - Assign more weight to constraints from definitions, than constraints from use
- Select constraints with explicit error messages
  - Avoids reporting on implicit equalities that arise during constraint generation

More Aspects

Requirements on the solver

- Report (approximated) conflict sets
  - Conflict sets should be over-approximations
  - The closer to the minimal conflict set, the better
- Fast failure is important
  - Solver is called repeatedly with smaller constraint sets

How to report errors?

- Reporting ‘<long function type> expected, but got <long but slightly different function type>’ can be improved
- Can we report that first argument is different, or that number of arguments is different?

Exercises

Conclusion

Summary

Unification with Union-Find

- Represent unifier as a graph
- Testing term equality becomes testing node identity
- Really fast with path compression and tree balancing

Error Reporting

- Selecting constraints to report errors on
- Use conflict sets to find cause of inconsistencies
- Report on
  - all conflicting constraints (program slice)
  - subset of constraints that removes inconsistency
- Use heuristics to select constraints to blame