Logic and Mechanized Reasoning

Unification

Marijn J.H. Heule

Carnegie Mellon University

Introduction

Generality of Unifiers

Unification Function

Termination

Introduction

Generality of Unifiers

Unification Function

Termination

Variables and Constants

Letters early in the alphabet refer to constants, e.g. a, b, c

- Constants can be seen as 0-arity functions

Small letters starting with f refer to functions, e.g. f, g, h

Letters late in the alphabet refer to variables, e.g. x, y, z

Capital letters refer to relations, e.g. P, Q, R

Motivation

Given a language with the following proven sentence:

$$\forall x, y, z.\ x < y \rightarrow x + z < y + z$$

and we try to prove

$$ab + 7 < c + 7$$

How to proceed? How can be combine them?

Motivation

Given a language with the following proven sentence:

$$\forall x, y, z.\ x < y \rightarrow x + z < y + z$$

and we try to prove

$$ab + 7 < c + 7$$

How to proceed? How can be combine them?

Substitute ab for x, c for y, and 7 for z

Motivation

Given a language with the following proven sentence:

$$\forall x, y, z.\ x < y \to x + z < y + z$$

and we try to prove

$$ab + 7 < c + 7$$

How to proceed? How can be combine them?

Substitute ab for x, c for y, and 7 for z

Note that Lean has to do this anytime you use `rw` or `apply`

Matching and Unification

Matching: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = t_i$

Matching and Unification

Matching: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = t_i$

Unification: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = \sigma t_i$

Matching and Unification

Matching: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = t_i$

Unification: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = \sigma t_i$

Example

Consider the following two expressions

$$f(x, f(x, a)) < z \quad f(b, y) < c$$

How to unify them?

Matching and Unification

Matching: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = t_i$

Unification: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = \sigma t_i$

Example

Consider the following two expressions

$$f(x, f(x, a)) < z \quad f(b, y) < c$$

How to unify them? $x \mapsto b$, $y \mapsto f(b, a)$ and $z \mapsto c$

Matching and Unification

Matching: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = t_i$

Unification: Given n pairs of terms (s_1, t_1), (s_2, t_2), ..., (s_n, t_n), find a substitution σ such that for every i: $\sigma s_i = \sigma t_i$

Example

Consider the following two expressions

$$f(x, f(x, a)) < z \quad f(b, y) < c$$

How to unify them? $x \mapsto b$, $y \mapsto f(b, a)$ and $z \mapsto c$

$$f(b, f(b, a)) < c$$

Prove Contradiction

Example

Consider the following formula

$$\forall x, z.\ R(f(x, f(x, a)), z) \land \forall y.\ \neg R(f(b, y), c)$$

Is this sentence satisfiable?

Prove Contradiction

Example

Consider the following formula

$$\forall x, z.\ R(f(x, f(x, a)), z) \wedge \forall y.\ \neg R(f(b, y), c)$$

Is this sentence satisfiable?

Apply substitution $x \mapsto b$, $y \mapsto f(b, a)$ and $z \mapsto c$

$$R(f(b, f(b, a)), c) \wedge \neg R(f(b, f(b, a)), c)$$

The above is a contradiction

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x, a)$ and $f(b, y)$

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x,a)$ and $f(b,y)$ $\qquad x \mapsto b$ and $y \mapsto a$
- $f(x,y)$ and $f(y,x)$

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x, a)$ and $f(b, y)$ $\quad$ $x \mapsto b$ and $y \mapsto a$
- $f(x, y)$ and $f(y, x)$ $\quad$ $y \mapsto x$
- $f(x, x)$ and $f(y, g(y))$

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x, a)$ and $f(b, y)$ $\quad x \mapsto b$ and $y \mapsto a$
- $f(x, y)$ and $f(y, x)$ $\quad y \mapsto x$
- $f(x, x)$ and $f(y, g(y))$ $\quad y$ and $g(y)$ can't be unified
- $f(a, x)$ and $f(y, g(y))$

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x, a)$ and $f(b, y)$ $\quad x \mapsto b$ and $y \mapsto a$
- $f(x, y)$ and $f(y, x)$ $\quad y \mapsto x$
- $f(x, x)$ and $f(y, g(y))$ $\quad y$ and $g(y)$ can't be unified
- $f(a, x)$ and $f(y, g(y))$ $\quad x \mapsto g(a)$ and $y \mapsto a$
- $f(a, x)$ and $f(x, b)$

Introduction: Examples

Hints:

- a and b can't be unified
- x and $f(x)$ can't be unified

Unify the following terms (if possible):

- $f(x,a)$ and $f(b,y)$ — $x \mapsto b$ and $y \mapsto a$
- $f(x,y)$ and $f(y,x)$ — $y \mapsto x$
- $f(x,x)$ and $f(y,g(y))$ — y and $g(y)$ can't be unified
- $f(a,x)$ and $f(y,g(y))$ — $x \mapsto g(a)$ and $y \mapsto a$
- $f(a,x)$ and $f(x,b)$ — x cannot map to a and b

Introduction

Generality of Unifiers

Unification Function

Termination

Many Unifiers

Example

Can $f(x, y)$ and $f(a, z)$ be unified?

Many Unifiers

Example

Can $f(x, y)$ and $f(a, z)$ be unified?

- $x \mapsto a$, $y \mapsto a$, and $z \mapsto a$
- $x \mapsto a$ and $y \mapsto z$
- $x \mapsto a$, $y \mapsto g(a)$, and $z \mapsto g(a)$

Many Unifiers

Example

Can $f(x,y)$ and $f(a,z)$ be unified?

- $x \mapsto a$, $y \mapsto a$, and $z \mapsto a$
- $x \mapsto a$ and $y \mapsto z$
- $x \mapsto a$, $y \mapsto g(a)$, and $z \mapsto g(a)$

Some unifiers are more useful that others

- $f(a,z)$ is more general than $f(a,a)$

Composing Substitutions

To explain what it means for one unifier to be better than another, we analyze the composition of substitutions

Composition of two substitutions σ and δ is written $\sigma\delta$

Example

To unify $f(x,z)$ and $f(g(y),z)$, consider the substitutions

- $\sigma = \{x \mapsto g(y)\}$
- $\delta = \{y \mapsto a, z \mapsto b\}$
- $\sigma\delta = \{x \mapsto g(a), z \mapsto b\}$
- σ and $\sigma\delta$ are unifiers

Generality of Unifiers

- We prefer unifiers that are as general as possible.
- A unifier σ is at least as general as unifier τ if there exists another substitution δ such that $\sigma\delta = \tau$
- σ is more general than τ if σ is at least as general as τ but not the other way around
- Intuition: σ more general than τ if τ can be obtained from through another substitution

Generality of Unifiers

- We prefer unifiers that are as general as possible.
- A unifier σ is at least as general as unifier τ if there exists another substitution δ such that $\sigma\delta = \tau$
- σ is more general than τ if σ is at least as general as τ but not the other way around
- Intuition: σ more general than τ if τ can be obtained from through another substitution

Example

Recall that $f(x, y)$ and $f(a, z)$ have (infinitely) many unifiers.

- $\sigma = \{x \mapsto a, y \mapsto z\}$
- $\tau = \{x \mapsto a, y \mapsto a, z \mapsto a\}$
- Which is more general?

Generality of Unifiers

- We prefer unifiers that are as general as possible.
- A unifier σ is at least as general as unifier τ if there exists another substitution δ such that $\sigma\delta = \tau$
- σ is more general than τ if σ is at least as general as τ but not the other way around
- Intuition: σ more general than τ if τ can be obtained from through another substitution

Example

Recall that $f(x, y)$ and $f(a, z)$ have (infinitely) many unifiers.

- $\sigma = \{x \mapsto a, y \mapsto z\}$
- $\tau = \{x \mapsto a, y \mapsto a, z \mapsto a\}$
- Which is more general?
- Let $\delta = \{y \mapsto a, z \mapsto a\}$, then $\sigma\delta = \tau$

Introduction: Most General Unifier

For every unification problem, there exists either

- a unique most general unifier (modulo renaming)
- no unifier

The most general unifier can be computed in linear time

Introduction: Most General Unifier

For every unification problem, there exists either

- a unique most general unifier (modulo renaming)
- no unifier

The most general unifier can be computed in linear time

We present an algorithm with a Lean implementation from the Handbook of Practical Logic and Automated Reasoning

- `env` is the (partial) substitution
- `eqs` is a set of pairs to unify
- `unify? env eqs` with `env` initially emtpy

Introduction

Generality of Unifiers

Unification Function

Termination

Extending a Cycle-Free Association List

Given a cycle-free association list `env` mapping variables to terms can we add the $(x \mapsto t)$ without creating a cycle?

A cycle is:

$$x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_p \longrightarrow x_0$$

It is sufficient to ensure the following:

1. There is no assignment $x \mapsto s$ in `env`
2. There is no $y \in var(t)$ such that $y \longrightarrow^* x$

Extending a Cycle-Free Association List

Given a cycle-free association list `env` mapping variables to terms can we add the $(x \mapsto t)$ without creating a cycle?

A cycle is:

$$x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_p \longrightarrow x_0$$

It is sufficient to ensure the following:

1. There is no assignment $x \mapsto s$ in `env`
2. There is no $y \in var(t)$ such that $y \longrightarrow^* x$

Proof sketch: Assume $(x \mapsto t)$ creates the cycle $z \longrightarrow x_1 \longrightarrow x \longrightarrow' y \longrightarrow \cdots \longrightarrow x_p \longrightarrow z$, then there existed a path $y \longrightarrow \cdots \longrightarrow x_p \longrightarrow z \longrightarrow x_1 \longrightarrow x$, which contradicts 2.

Trivial Check

We can add the $(x \mapsto t)$ without creating a cycle by ensuring:

1. There is no assignment $x \mapsto s$ in `env`
2. There is no $y \in var(t)$ such that $y \longrightarrow^* x$

Helper procedure to determine whether to add $(x \mapsto t)$

- Return true: $t = x$ in `env` (trivial)
- Return false: no cycle will be created
- Return none: unification not possible

Trivial Check

We can add the $(x \mapsto t)$ without creating a cycle by ensuring:

1. There is no assignment $x \mapsto s$ in `env`
2. There is no $y \in var(t)$ such that $y \longrightarrow^* x$

Helper procedure to determine whether to add $(x \mapsto t)$

- Return true: $t = x$ in `env` (trivial)
- Return false: no cycle will be created
- Return none: unification not possible

Lean: `isTriv?`

Main Unification Function

Given `eqs`, a list of pairs to unify, determine if unification succeeds and produce an association list `env` if possible

- Tail-recursive algorithm
- Front pair (x, t)
  - If $x \mapsto s$ in `env` replace (x, t) by (s, t)
  - If (x, t) is trivial, skip it and continue
  - If (x, t) creates a cycle, return failed
  - Otherwise add $(x \mapsto t)$

Main Unification Function

Given `eqs`, a list of pairs to unify, determine if unification succeeds and produce an association list `env` if possible

- Tail-recursive algorithm
- Front pair (x, t)
  - If $x \mapsto s$ in `env` replace (x, t) by (s, t)
  - If (x, t) is trivial, skip it and continue
  - If (x, t) creates a cycle, return failed
  - Otherwise add $(x \mapsto t)$
- Front pair (t, x), replace by (x, t)

Main Unification Function

Given `eqs`, a list of pairs to unify, determine if unification succeeds and produce an association list `env` if possible

- Tail-recursive algorithm
- Front pair (x, t)
  - If $x \mapsto s$ in `env` replace (x, t) by (s, t)
  - If (x, t) is trivial, skip it and continue
  - If (x, t) creates a cycle, return failed
  - Otherwise add $(x \mapsto t)$
- Front pair (t, x), replace by (x, t)
- Front pair (s, t), if both s and t are the same functions with the same number of arguments, add for each argument a pairs (s_i, t_i) to eqs

Main Unification Function

Given `eqs`, a list of pairs to unify, determine if unification succeeds and produce an association list `env` if possible

- Tail-recursive algorithm
- Front pair (x, t)
  - If $x \mapsto s$ in `env` replace (x, t) by (s, t)
  - If (x, t) is trivial, skip it and continue
  - If (x, t) creates a cycle, return failed
  - Otherwise add $(x \mapsto t)$
- Front pair (t, x), replace by (x, t)
- Front pair (s, t), if both s and t are the same functions with the same number of arguments, add for each argument a pairs (s_i, t_i) to eqs

Lean: `unify?`

Example a Successful Run of the Algorithm

Example

Unify $f(g(x), g(x))$ and $f(y, g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x), g(x)), f(y, g(a)))\}$

Example a Successful Run of the Algorithm

Example

Unify $f(g(x),g(x))$ and $f(y,g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x),g(x)),f(y,g(a)))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(g(x),y),(g(x),g(a))\}$

Example a Successful Run of the Algorithm

Example

Unify $f(g(x), g(x))$ and $f(y, g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x), g(x)), f(y, g(a)))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(g(x), y), (g(x), g(a))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(y, g(x)), (g(x), g(a))\}$

Example a Successful Run of the Algorithm

Example

Unify $f(g(x), g(x))$ and $f(y, g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x), g(x)), f(y, g(a)))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(g(x), y), (g(x), g(a))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(y, g(x)), (g(x), g(a))\}$
- $\texttt{env} = \{y \mapsto g(x)\}$, $\texttt{eqs} = \{(g(x), g(a))\}$

Example a Successful Run of the Algorithm

Example

Unify $f(g(x), g(x))$ and $f(y, g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x), g(x)), f(y, g(a)))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(g(x), y), (g(x), g(a))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(y, g(x)), (g(x), g(a))\}$
- $\texttt{env} = \{y \mapsto g(x)\}$, $\texttt{eqs} = \{(g(x), g(a))\}$
- $\texttt{env} = \{y \mapsto g(x)\}$, $\texttt{eqs} = \{(x, a)\}$

Example a Successful Run of the Algorithm

Example

Unify $f(g(x), g(x))$ and $f(y, g(a))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(g(x), g(x)), f(y, g(a)))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(g(x), y), (g(x), g(a))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(y, g(x)), (g(x), g(a))\}$
- $\texttt{env} = \{y \mapsto g(x)\}$, $\texttt{eqs} = \{(g(x), g(a))\}$
- $\texttt{env} = \{y \mapsto g(x)\}$, $\texttt{eqs} = \{(x, a)\}$
- $\texttt{env} = \{y \mapsto g(x), x \mapsto a\}$, $\texttt{eqs} = \{\}$

Example a Failed Run of the Algorithm

Example

Unify $f(x,x)$ and $f(y,g(y))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(x,x), f(y,g(y))\}$

Example a Failed Run of the Algorithm

Example

Unify $f(x,x)$ and $f(y,g(y))$

- `env` $=\{\}$, `eqs` $=\{(f(x,x),f(y,g(y))\}$
- `env` $=\{\}$, `eqs` $=\{(x,y),(x,g(y))\}$

Example a Failed Run of the Algorithm

Example

Unify $f(x,x)$ and $f(y,g(y))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(x,x), f(y,g(y))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(x,y),(x,g(y))\}$
- $\texttt{env} = \{x \mapsto y\}$, $\texttt{eqs} = \{(x,g(y))\}$

Example a Failed Run of the Algorithm

Example

Unify $f(x, x)$ and $f(y, g(y))$

- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(f(x, x), f(y, g(y))\}$
- $\texttt{env} = \{\}$, $\texttt{eqs} = \{(x, y), (x, g(y))\}$
- $\texttt{env} = \{x \mapsto y\}$, $\texttt{eqs} = \{(x, g(y))\}$
- $\texttt{env} = \{x \mapsto y\}$, $\texttt{eqs} = \{(y, g(y))\}$

Example a Failed Run of the Algorithm

Example

Unify $f(x,x)$ and $f(y,g(y))$

- `env` $= \{\}$, `eqs` $= \{(f(x,x), f(y,g(y))\}$
- `env` $= \{\}$, `eqs` $= \{(x,y), (x,g(y))\}$
- `env` $= \{x \mapsto y\}$, `eqs` $= \{(x,g(y))\}$
- `env` $= \{x \mapsto y\}$, `eqs` $= \{(y,g(y))\}$
- `isTriv? env eqs` returns none, indicating failure

The Final Step

Successful termination shows that there exists a unifier

For example, the algorithm may return:

- $\texttt{env} = \{x \mapsto y, y \mapsto z, z \mapsto w\}$
- How to turn this in the most general unifier?

The Final Step

Successful termination shows that there exists a unifier

For example, the algorithm may return:

- `env` $= \{x \mapsto y, y \mapsto z, z \mapsto w\}$
- How to turn this in the most general unifier?
- Apply the map until fixpoint
- `env` $= \{x \mapsto w, y \mapsto w, z \mapsto w\}$

What is the complexity of computing the fixpoint?

The Final Step

Successful termination shows that there exists a unifier

For example, the algorithm may return:

- $\texttt{env} = \{x \mapsto y, y \mapsto z, z \mapsto w\}$
- How to turn this in the most general unifier?
- Apply the map until fixpoint
- $\texttt{env} = \{x \mapsto w, y \mapsto w, z \mapsto w\}$

What is the complexity of computing the fixpoint?

- $\texttt{env} = \{x_0 \mapsto f(x_1, x_1), x_1 \mapsto f(x_2, x_2), x_2 \mapsto f(x_3, x_3)\}$

The Final Step

Successful termination shows that there exists a unifier

For example, the algorithm may return:

- $\texttt{env} = \{x \mapsto y, y \mapsto z, z \mapsto w\}$
- How to turn this in the most general unifier?
- Apply the map until fixpoint
- $\texttt{env} = \{x \mapsto w, y \mapsto w, z \mapsto w\}$

What is the complexity of computing the fixpoint?

- $\texttt{env} = \{x_0 \mapsto f(x_1, x_1), x_1 \mapsto f(x_2, x_2), x_2 \mapsto f(x_3, x_3)\}$
- $x_0 \mapsto f(f(f(x_3, x_3), f(x_3, x_3)), f(f(x_3, x_3), f(x_3, x_3)))$
- $x_1 \mapsto f(f(x_3, x_3), f(x_3, x_3))$
- $x_2 \mapsto f(x_3, x_3)$

The Final Step

Successful termination shows that there exists a unifier

For example, the algorithm may return:

- `env` $= \{x \mapsto y, y \mapsto z, z \mapsto w\}$
- How to turn this in the most general unifier?
- Apply the map until fixpoint
- `env` $= \{x \mapsto w, y \mapsto w, z \mapsto w\}$

What is the complexity of computing the fixpoint?

- `env` $= \{x_0 \mapsto f(x_1, x_1), x_1 \mapsto f(x_2, x_2), x_2 \mapsto f(x_3, x_3)\}$
- $x_0 \mapsto f(f(f(x_3, x_3), f(x_3, x_3)), f(f(x_3, x_3), f(x_3, x_3)))$
- $x_1 \mapsto f(f(x_3, x_3), f(x_3, x_3))$
- $x_2 \mapsto f(x_3, x_3)$

Lean: `usolve`

Introduction

Generality of Unifiers

Unification Function

Termination

Unification Termination (I)

Why does the unification algorithm terminate?

Unification Termination (I)

Why does the unification algorithm terminate?
Key observation: The number of additions to `env` is at most the number of variables in `eqs` that are not in `env`

Size of `eqs`:

- Total number of variables and functions in $(\sigma s_i,\ \sigma t_i)$ with $(s_i, t_i) \in$ eqs with σ being the fixpoint of `env`

Unification Termination (I)

Why does the unification algorithm terminate?
Key observation: The number of additions to `env` is at most the number of variables in `eqs` that are not in `env`

Size of `eqs`:

- Total number of variables and functions in $(\sigma s_i, \sigma t_i)$ with $(s_i, t_i) \in$ eqs with σ being the fixpoint of `env`
- Each step in the algorithm either decreases the size of `eqs` or can be applied finitely many times

Unification Termination (I)

Why does the unification algorithm terminate?
Key observation: The number of additions to `env` is at most the number of variables in `eqs` that are not in `env`

Size of `eqs`:

- Total number of variables and functions in $(\sigma s_i, \sigma t_i)$ with $(s_i, t_i) \in$ `eqs` with σ being the fixpoint of `env`
- Each step in the algorithm either decreases the size of `eqs` or can be applied finitely many times
- The only exception is an addition step to `env`

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i, \sigma t_i)$ with $(s_i, t_i) \in$ `eqs` with σ being the fixpoint of `env`

Steps in the function:

- If $x \mapsto s$ in `env` replace (x, t) by (s, t)

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i,\ \sigma t_i)$ with $(s_i, t_i) \in \texttt{eqs}$ with σ being the fixpoint of $\texttt{env}$

Steps in the function:

- If $x \mapsto s$ in $\texttt{env}$ replace (x, t) by (s, t)
  - Same size, but limited as $\texttt{env}$ is cycle-free
- If (x, t) is trivial, skip it and continue

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i, \sigma t_i)$ with $(s_i, t_i) \in$ `eqs` with σ being the fixpoint of `env`

Steps in the function:

- If $x \mapsto s$ in `env` replace (x, t) by (s, t)
  - Same size, but limited as `env` is cycle-free
- If (x, t) is trivial, skip it and continue
  - Reduces `eqs`
- Otherwise add $(x \mapsto t)$

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i,\ \sigma t_i)$ with $(s_i, t_i) \in \texttt{eqs}$ with σ being the fixpoint of $\texttt{env}$

Steps in the function:

- If $x \mapsto s$ in $\texttt{env}$ replace (x, t) by (s, t)
  - Same size, but limited as $\texttt{env}$ is cycle-free
- If (x, t) is trivial, skip it and continue
  - Reduces $\texttt{eqs}$
- Otherwise add $(x \mapsto t)$
  - Only increasing step, but limited
- Front pair (t, x), replace by (x, t)

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i,\ \sigma t_i)$ with $(s_i, t_i) \in \texttt{eqs}$ with σ being the fixpoint of $\texttt{env}$

Steps in the function:

- If $x \mapsto s$ in $\texttt{env}$ replace (x, t) by (s, t)
  - Same size, but limited as $\texttt{env}$ is cycle-free
- If (x, t) is trivial, skip it and continue
  - Reduces $\texttt{eqs}$
- Otherwise add $(x \mapsto t)$
  - Only increasing step, but limited
- Front pair (t, x), replace by (x, t)
  - Cannot be repeated twice
- Front pair (s, t), add the pairs (s_i, t_i) to $\texttt{eqs}$

Unification Termination (II)

Size: Total number of variables and functions in $(\sigma s_i, \sigma t_i)$ with $(s_i, t_i) \in$ eqs with σ being the fixpoint of env

Steps in the function:

- If $x \mapsto s$ in env replace (x, t) by (s, t)
  - Same size, but limited as env is cycle-free
- If (x, t) is trivial, skip it and continue
  - Reduces eqs
- Otherwise add $(x \mapsto t)$
  - Only increasing step, but limited
- Front pair (t, x), replace by (x, t)
  - Cannot be repeated twice
- Front pair (s, t), add the pairs (s_i, t_i) to eqs
  - Reduce size (removes two functions)