Typed Unification: when failure may not be wrong

João Barbosa Mário Florido Vítor Santos Costa

Department of Computer Science, Faculty of Science, University of Porto, Portugal
LIACC - Artificial Intelligence and Computer Science Laboratory, University of Porto

3^{rd} July 2023

The Universe of Terms

In logic programming, terms are:

The Universe of Terms

In logic programming, terms are:

- syntactic objects

The Universe of Terms

In logic programming, terms are:

- syntactic objects
- interpreted as semantic values that are trees

The Universe of Terms

In logic programming, terms are:

- syntactic objects
- interpreted as semantic values that are trees
- Herbrand interpretation.

The Universe of Terms

In logic programming, terms are:

- syntactic objects
- interpreted as semantic values that are trees
- Herbrand interpretation.

Unification, therefore:

The Universe of Terms

In logic programming, terms are:

- syntactic objects
- interpreted as semantic values that are trees
- Herbrand interpretation.

Unification, therefore:

- is syntactic

The Universe of Terms

In logic programming, terms are:

- syntactic objects
- interpreted as semantic values that are trees
- Herbrand interpretation.

Unification, therefore:

- is syntactic
- either returns a most general unifier (MGU) or fails.

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains.

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain
- terms are interpreted as semantic values in some domain

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain
- terms are interpreted as semantic values in some domain
- function symbols are interpreted as functions that have arguments of some domain, and output a value in some domain.

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain
- terms are interpreted as semantic values in some domain
- function symbols are interpreted as functions that have arguments of some domain, and output a value in some domain.

Types are associated with domains.

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain
- terms are interpreted as semantic values in some domain
- function symbols are interpreted as functions that have arguments of some domain, and output a value in some domain.

Types are associated with domains. We can assume:

- a function symbol that builds terms has a functional type (n input types and one output type)

Types for Logic Programming

We can suppose that the Universe is separated into disjoint domains. Then:

- each semantic value belongs to a domain
- terms are interpreted as semantic values in some domain
- function symbols are interpreted as functions that have arguments of some domain, and output a value in some domain.

Types are associated with domains. We can assume:

- a function symbol that builds terms has a functional type (n input types and one output type)
- now terms themselves can contain type errors, while in the Herbrand interpretation there is a single type (TERM), so there can be no type errors in a term.

Terms and Their Types

- Every well-typed ground term has a type.

Terms and Their Types

- Every well-typed ground term has a type.
- Unification only makes sense for values of the same domain.

Terms and Their Types

- Every well-typed ground term has a type.
- Unification only makes sense for values of the same domain.
- So we will perform unification while also checking if the terms belong to the same type (and are well-typed).

Terms and Their Types

- Every well-typed ground term has a type.
- Unification only makes sense for values of the same domain.
- So we will perform unification while also checking if the terms belong to the same type (and are well-typed).

Since not all terms are ground and, in particular, interesting unification cases include non-ground terms, we need to define a type for a non-ground term.

Non-Ground Terms and Their Types

- Every variable can be, potentially, instantiated with any term.

Non-Ground Terms and Their Types

- Every variable can be, potentially, instantiated with any term.
- Therefore, we assume each variable has a type that corresponds to everything.

Non-Ground Terms and Their Types

- Every variable can be, potentially, instantiated with any term.
- Therefore, we assume each variable has a type that corresponds to everything.
- We will give different names (type variables) to the types of different variables just to denote constraints for each one.

Non-Ground Terms and Their Types

- Every variable can be, potentially, instantiated with any term.
- Therefore, we assume each variable has a type that corresponds to everything.
- We will give different names (type variables) to the types of different variables just to denote constraints for each one.

Types for well-typed non-ground terms can be types containing type variables, which we will call polymorphic types.

Examples of a Well-Typed Term

Let $t = f(2, g(1, a, b))$ be a ground term.

Examples of a Well-Typed Term

Let $t = f(2, g(1, a, b))$ be a ground term.

- If we know that the types for the function symbols f and g are $int \times int \rightarrow atom$ and $int \times atom \times atom \rightarrow int$, respectively.

Examples of a Well-Typed Term

Let $t = f(2, g(1, a, b))$ be a ground term.

- If we know that the types for the function symbols f and g are $int \times int \rightarrow atom$ and $int \times atom \times atom \rightarrow int$, respectively.
- We can conclude that the type for term t is $atom$.

Examples of a Well-Typed Term

Let $t = f(2, g(1, a, b))$ be a ground term.

- If we know that the types for the function symbols f and g are $int \times int \rightarrow atom$ and $int \times atom \times atom \rightarrow int$, respectively.
- We can conclude that the type for term t is $atom$.
- In fact, with just information for f we could have known what the type for t could be, if it was well-typed.

Examples of a Well-Typed Term

Let $t = f(2, g(1, a, b))$ be a ground term.

- If we know that the types for the function symbols f and g are $int \times int \rightarrow atom$ and $int \times atom \times atom \rightarrow int$, respectively.
- We can conclude that the type for term t is *atom*.
- In fact, with just information for f we could have known what the type for t could be, if it was well-typed.
- We call f the principal functor of t, and the type of a well-typed term is the output type for its principal functor.

Examples of a Ill-Typed Term

Let $t = f(X, X)$ be a non-ground term.

Examples of a Ill-Typed Term

Let $t = f(X, X)$ be a non-ground term.

- If we know that the type for the function symbols f is $int \times atom \rightarrow int$.

Examples of a Ill-Typed Term

Let $t = f(X, X)$ be a non-ground term.

- If we know that the type for the function symbols f is $int \times atom \rightarrow int$.
- Then we can note that no ground instance of this term is well-typed.

Examples of a Ill-Typed Term

Let $t = f(X, X)$ be a non-ground term.

- If we know that the type for the function symbols f is $int \times atom \rightarrow int$.
- Then we can note that no ground instance of this term is well-typed.
- The type for X has to be simultaneously *int* an *atom*.

Examples of a Ill-Typed Term

Let $t = f(X, X)$ be a non-ground term.

- If we know that the type for the function symbols f is $int \times atom \rightarrow int$.
- Then we can note that no ground instance of this term is well-typed.
- The type for X has to be simultaneously *int* an *atom*.
- Therefore we say that the term is ill-typed.

What we want from Typed Unification

If we assume a type discipline for terms, now when we perform unification we can have three output values:

What we want from Typed Unification

If we assume a type discipline for terms, now when we perform unification we can have three output values:

- an MGU - if the terms unify and are well-typed

What we want from Typed Unification

If we assume a type discipline for terms, now when we perform unification we can have three output values:

- an MGU - if the terms unify and are well-typed
- false - if the terms do not unify but are well-typed (and have unifiable types)

What we want from Typed Unification

If we assume a type discipline for terms, now when we perform unification we can have three output values:

- an MGU - if the terms unify and are well-typed
- false - if the terms do not unify but are well-typed (and have unifiable types)
- wrong - we cannot simultaneously unify the terms and have them be well-typed.

What we want from Typed Unification

If we assume a type discipline for terms, now when we perform unification we can have three output values:

- an MGU - if the terms unify and are well-typed
- false - if the terms do not unify but are well-typed (and have unifiable types)
- wrong - we cannot simultaneously unify the terms and have them be well-typed.

We do not yet have a proof that our algorithm behaves correctly, but we are working on it currently.

Typed Unification Algorithm

Given two terms t_1 and t_2, we build an initial pair of sets of constraints, in the following way:

Typed Unification Algorithm

Given two terms t_1 and t_2, we build an initial pair of sets of constraints, in the following way:

- C $\rightarrow$ a set of equality constraints that initially contains only $t_1 = t_2$

Typed Unification Algorithm

Given two terms t_1 and t_2, we build an initial pair of sets of constraints, in the following way:

- C $\rightarrow$ a set of equality constraints that initially contains only $t_1 = t_2$
- S $\rightarrow$ a set of type equality constraints and membership constraints that consists on the following constraints:

Typed Unification Algorithm

Given two terms t_1 and t_2, we build an initial pair of sets of constraints, in the following way:

- C $\rightarrow$ a set of equality constraints that initially contains only $t_1 = t_2$
- S $\rightarrow$ a set of type equality constraints and membership constraints that consists on the following constraints:
  - let $t_1 = f(s_1, \ldots, s_n)$ and $t_2 = g(u_1, \ldots, u_m)$
  - let f have type $\tau_1 \times \cdots \times \tau_n \rightarrow \tau$ and g have $\tau\prime_1 \times \cdots \times \tau\prime_m \rightarrow \tau\prime$
  - $\{\tau \doteq \alpha, \tau\prime \doteq \alpha, s_1 \in \tau_1, \ldots s_n \in \tau_n, u_1 \in \tau\prime_1, \ldots, u_m \in \tau\prime_m\}$

Example

The initial set set built for terms $f(1, h(Y, a), X)$ and $g(h(Z, Z))$, considering f has type $int \times int \times int \rightarrow int$ and g has type $int \rightarrow int$ is:

Example

The initial set set built for terms $f(1, h(Y, a), X)$ and $g(h(Z, Z))$, considering f has type $int \times int \times int \rightarrow int$ and g has type $int \rightarrow int$ is:

- $C = \{f(1, h(Y, a), X) = g(h(Z, Z))\}$

Example

The initial set set built for terms $f(1, h(Y, a), X)$ and $g(h(Z, Z))$, considering f has type $int \times int \times int \rightarrow int$ and g has type $int \rightarrow int$ is:

- $C = \{f(1, h(Y, a), X) = g(h(Z, Z))\}$
- $S = \{int \doteq \alpha, int \doteq \alpha, 1 \in int, h(Y, a) \in int, X \in int, h(Z, Z) \in int\}$

Example

The initial set set built for terms $f(1, h(Y, a), X)$ and $g(h(Z, Z))$, considering f has type $int \times int \times int \rightarrow int$ and g has type $int \rightarrow int$ is:

- $C = \{f(1, h(Y, a), X) = g(h(Z, Z))\}$
- $S = \{int \doteq \alpha, int \doteq \alpha, 1 \in int, h(Y, a) \in int, X \in int, h(Z, Z) \in int\}$
- output $= (C, S)$

Algorithm Steps - 1

The algorithm applies certain rules in order.

Algorithm Steps - 1

The algorithm applies certain rules in order.
The first few rules are the following:

Algorithm Steps - 1

The algorithm applies certain rules in order.
The first few rules are the following:

1. $(C, \{f(t_1, \ldots, t_n) \in \tau\} \cup Rest) \to (C, \{t_1 \in \tau\prime_1, \ldots, t_n \in \tau\prime_n, \tau\prime_1 \doteq \tau_1, \ldots, \tau\prime_n \doteq \tau_n\} \cup Rest)$, where the type for f is $f :: \tau_1 \times \cdots \times \tau_n \to \tau$, and $\tau\prime_i$ are the types for the principal functors of t_i, respectively

Algorithm Steps - 1

The algorithm applies certain rules in order.
The first few rules are the following:

1. $(C, \{f(t_1, \ldots, t_n) \in \tau\} \cup Rest) \rightarrow (C, \{t_1 \in \tau\prime_1, \ldots, t_n \in \tau\prime_n, \tau\prime_1 \doteq \tau_1, \ldots, \tau\prime_n \doteq \tau_n\} \cup Rest)$, where the type for f is $f :: \tau_1 \times \cdots \times \tau_n \rightarrow \tau$, and $\tau\prime_i$ are the types for the principal functors of t_i, respectively
2. $(C, \{X_i \in \alpha_i\} \cup Rest) \rightarrow (C, Rest)$

Algorithm Steps - 1

The algorithm applies certain rules in order.
The first few rules are the following:

1. $(C, \{f(t_1, \ldots, t_n) \in \tau\} \cup Rest) \rightarrow (C, \{t_1 \in \tau\prime_1, \ldots, t_n \in \tau\prime_n, \tau\prime_1 \doteq \tau_1, \ldots, \tau\prime_n \doteq \tau_n\} \cup Rest)$, where the type for f is $f :: \tau_1 \times \cdots \times \tau_n \rightarrow \tau$, and $\tau\prime_i$ are the types for the principal functors of t_i, respectively
2. $(C, \{X_i \in \alpha_i\} \cup Rest) \rightarrow (C, Rest)$
3. $(C, \{c \in \tau\} \cup Rest) \rightarrow (C, Rest)$

Algorithm Steps - 1

The algorithm applies certain rules in order.
The first few rules are the following:

1. $(C, \{f(t_1, \ldots, t_n) \in \tau\} \cup Rest) \rightarrow (C, \{t_1 \in \tau\prime_1, \ldots, t_n \in \tau\prime_n, \tau\prime_1 \doteq \tau_1, \ldots, \tau\prime_n \doteq \tau_n\} \cup Rest)$, where the type for f is $f :: \tau_1 \times \cdots \times \tau_n \rightarrow \tau$, and $\tau\prime_i$ are the types for the principal functors of t_i, respectively
2. $(C, \{X_i \in \alpha_i\} \cup Rest) \rightarrow (C, Rest)$
3. $(C, \{c \in \tau\} \cup Rest) \rightarrow (C, Rest)$

These reduce the number of membership constraints to zero, while generating type equality constraints.

Algorithm Steps - 2

The next few steps are as follows:

Algorithm Steps - 2

The next few steps are as follows:

4. $(C, \{f(\tau_1, \ldots, \tau_n) \doteq f(\tau\prime_1, \ldots, \tau\prime_n)\} \cup Rest) \rightarrow (C, \{\tau_1 \doteq \tau\prime_1, \ldots, \tau_n \doteq \tau\prime_n\} \cup Rest)$
5. $(C, \{\tau \doteq \tau\} \cup Rest) \rightarrow (C, Rest)$
6. $(C, \{f(\tau_1, \ldots, \tau_n) \doteq g(\tau\prime_1, \ldots, \tau\prime_m)\} \cup Rest) \rightarrow wrong$, if $f \neq g$ or $n \neq m$
7. $(C, \{\tau \doteq \alpha\} \cup Rest) \rightarrow (C, \{\alpha \doteq \tau\} \cup Rest)$, τ is not a type variable
8. $(C, \{\alpha \doteq \tau\} \cup Rest) \rightarrow (C, \{\alpha \doteq \tau\} \cup Rest[\alpha \mapsto \tau])$, if α does not occur in τ
9. $(C, \{\alpha \doteq \tau\} \cup Rest) \rightarrow wrong$, if α occurs in τ

Algorithm Steps - 2

The next few steps are as follows:

4. $(C, \{f(\tau_1, \ldots, \tau_n) \doteq f(\tau\prime_1, \ldots, \tau\prime_n)\} \cup Rest) \rightarrow (C, \{\tau_1 \doteq \tau\prime_1, \ldots, \tau_n \doteq \tau\prime_n\} \cup Rest)$
5. $(C, \{\tau \doteq \tau\} \cup Rest) \rightarrow (C, Rest)$
6. $(C, \{f(\tau_1, \ldots, \tau_n) \doteq g(\tau\prime_1, \ldots, \tau\prime_m)\} \cup Rest) \rightarrow wrong$, if $f \neq g$ or $n \neq m$
7. $(C, \{\tau \doteq \alpha\} \cup Rest) \rightarrow (C, \{\alpha \doteq \tau\} \cup Rest)$, τ is not a type variable
8. $(C, \{\alpha \doteq \tau\} \cup Rest) \rightarrow (C, \{\alpha \doteq \tau\} \cup Rest[\alpha \mapsto \tau])$, if α does not occur in τ
9. $(C, \{\alpha \doteq \tau\} \cup Rest) \rightarrow wrong$, if α occurs in τ

These correspond to the Martelli-Montanari algorithm for unification, but on types.

Algorithm Steps - 3

The last few steps are as follows:

Algorithm Steps - 3

The last few steps are as follows:

10. $(\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup Rest, T) \rightarrow (\{t_1 = s_1, \ldots, t_n = s_n\} \cup Rest, T)$
11. $(\{t = t\} \cup Rest, T) \rightarrow (Rest, T)$
12. $(\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \cup Rest, T) \rightarrow false$, if $f \neq g$ or $n \neq m$
13. $(\{t = X\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest, T)$, t is not a variable
14. $(\{X = t\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest[X \mapsto t], T)$, if X does not occur in t
15. $(\{X = t\} \cup Rest, T) \rightarrow false$, if X occurs in t.

Algorithm Steps - 3

The last few steps are as follows:

10. $(\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup Rest, T) \rightarrow (\{t_1 = s_1, \ldots, t_n = s_n\} \cup Rest, T)$
11. $(\{t = t\} \cup Rest, T) \rightarrow (Rest, T)$
12. $(\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \cup Rest, T) \rightarrow false$, if $f \neq g$ or $n \neq m$
13. $(\{t = X\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest, T)$, t is not a variable
14. $(\{X = t\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest[X \mapsto t], T)$, if X does not occur in t
15. $(\{X = t\} \cup Rest, T) \rightarrow false$, if X occurs in t.

These also correspond to the Martelli-Montanari algorithm for unification for terms.

Details in the algorithm

- We can always apply one of the cases for the membership constraints.

Details in the algorithm

- We can always apply one of the cases for the membership constraints.
- When we fail unification on types, notice that we output wrong.

Details in the algorithm

- We can always apply one of the cases for the membership constraints.
- When we fail unification on types, notice that we output wrong.
- When we fail unification on terms, we output false.

Details in the algorithm

- We can always apply one of the cases for the membership constraints.
- When we fail unification on types, notice that we output wrong.
- When we fail unification on terms, we output false.
- Since the steps are applied in order, we can only return false if we do not output wrong, which means we were able to unify the types for both terms, and did not find a type error.

Example of Unification

Let $t_1 = f(1, f(X, 1))$ and $t_2 = f(Y, f(2, Y))$, and f have type $int \times int \rightarrow int$.

Example of Unification

Let $t_1 = f(1, f(X, 1))$ and $t_2 = f(Y, f(2, Y))$, and f have type $int \times int \rightarrow int$.

The initial tuple is: $(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{f(1, f(X, 1)) \in int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

Let $t_1 = f(1, f(X, 1))$ and $t_2 = f(Y, f(2, Y))$, and f have type $int \times int \rightarrow int$.

The initial tuple is: $(\{f(1, f(X, 1)) = f(Y, f(2, Y))\},$
$\{f(1, f(X, 1)) \in int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We can apply the rules for the membership constraints step-by-step.

Example of Unification

Let $t_1 = f(1, f(X, 1))$ and $t_2 = f(Y, f(2, Y))$, and f have type $int \times int \rightarrow int$.

The initial tuple is: $(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{f(1, f(X, 1)) \in int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We can apply the rules for the membership constraints step-by-step.

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{1 \in int, f(X, 1) \in int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{1 \in int, f(X,1) \in int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{1 \in int, f(X, 1) \in int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{X \in \alpha_X, 1 \in int, \alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{1 \in int, f(X, 1) \in int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{X \in \alpha_X, 1 \in int, \alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, f(Y, f(2, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We do the same for the other membership constraint:

Example of Unification

$(\{f(1,f(X,1)) = f(Y,f(2,Y))\}, \{1 \in int, f(X,1) \in int, int \doteq int, int \doteq int, f(Y,f(2,Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

$(\{f(1,f(X,1)) = f(Y,f(2,Y))\}, \{X \in \alpha_X, 1 \in int, \alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, f(Y,f(2,Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We do the same for the other membership constraint:

$(\{f(1,f(X,1)) = f(Y,f(2,Y))\}, \{\alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

Now we start applying the set of rules dealing with type equality.

Example of Unification

Now we start applying the set of rules dealing with type equality.

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, int \doteq int, int \doteq int,$
$int \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

Now we start applying the set of rules dealing with type equality.

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

Now we start applying the set of rules dealing with type equality.

$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, int \doteq int, int \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$

$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$

$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, \alpha \doteq int, int \doteq \alpha\})$

Example of Unification

$$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, \alpha \doteq int, int \doteq \alpha\})$$

Example of Unification

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, \alpha \doteq int, int \doteq \alpha\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha \doteq int, int \doteq int\})$

Example of Unification

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, \alpha \doteq int, int \doteq \alpha\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha \doteq int, int \doteq int\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$

Example of Unification

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha_Y \doteq int, \alpha \doteq int, int \doteq \alpha\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq int, \alpha \doteq int, int \doteq int\})$

$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$

No more rules apply, we did not halt with wrong, therefore there is no type error. We move on to the equality constraints.

Example of Unification

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X, 1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X, 1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, X = 2, 1 = Y\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X,1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, X = 2, 1 = Y\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, Y = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X, 1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, X = 2, 1 = Y\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, Y = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, 1 = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X, 1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X, 1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, X = 2, 1 = Y\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, Y = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, 1 = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{f(1, f(X,1)) = f(Y, f(2, Y))\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, f(X,1) = f(2, Y)\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{1 = Y, X = 2, 1 = Y\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, Y = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2, 1 = 1\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$$(\{Y = 1, X = 2\}, \{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

$[Y = 1, X = 2]$ is an MGU of both terms.

Example of Unification

Let $t_1 = g(1, a, h(X))$ and $t_2 = h(g(Y, b, Y))$, g have type $int \times atom \times int \rightarrow int$, and $int \rightarrow int$.

Example of Unification

Let $t_1 = g(1, a, h(X))$ and $t_2 = h(g(Y, b, Y))$, g have type $int \times atom \times int \rightarrow int$, and $int \rightarrow int$.

The initial tuple is: $(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$ $\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

Example of Unification

Let $t_1 = g(1, a, h(X))$ and $t_2 = h(g(Y, b, Y))$, g have type $int \times atom \times int \to int$, and $int \to int$.

The initial tuple is: $(\{g(1, a, h(X)) = h(g(Y, b, Y))\}, \{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We can immediately see that the terms do not unify, syntactically. But if we replace X by any integer and Y by any integer, the terms are well-typed. The algorithm should return *false*.

Example of Unification

Let $t_1 = g(1, a, h(X))$ and $t_2 = h(g(Y, b, Y))$, g have type $int \times atom \times int \rightarrow int$, and $int \rightarrow int$.

The initial tuple is: $(\{g(1, a, h(X)) = h(g(Y, b, Y))\}$, $\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$

We can immediately see that the terms do not unify, syntactically. But if we replace X by any integer and Y by any integer, the terms are well-typed. The algorithm should return *false*.

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{1 \in int, a \in atom, X \in \alpha_X, int \doteq int, atom \doteq atom, \alpha_X \doteq int,$$
$$Y \in \alpha_Y, b \in atom, \alpha_Y \doteq int, atom \doteq atom, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1,a,h(X)) = h(g(Y,b,Y))\},$$
$$\{g(1,a,h(X)) \in int, h(g(Y,b,Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1,a,h(X)) = h(g(Y,b,Y))\},$$
$$\{1 \in int, a \in atom, X \in \alpha_X, int \doteq int, atom \doteq atom, \alpha_X \doteq int,$$
$$Y \in \alpha_Y, b \in atom, \alpha_Y \doteq int, atom \doteq atom, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1,a,h(X)) = h(g(Y,b,Y))\},$$
$$\{int \doteq int, atom \doteq atom, \alpha_X \doteq int, \alpha_Y \doteq int, atom \doteq atom,$$
$$int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{g(1, a, h(X)) \in int, h(g(Y, b, Y)) \in int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{1 \in int, a \in atom, X \in \alpha_X, int \doteq int, atom \doteq atom, \alpha_X \doteq int,$$
$$Y \in \alpha_Y, b \in atom, \alpha_Y \doteq int, atom \doteq atom, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{int \doteq int, atom \doteq atom, \alpha_X \doteq int, \alpha_Y \doteq int, atom \doteq atom,$$
$$int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

Example of Unification

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{\alpha_X \doteq int, \alpha_Y \doteq int, int \doteq \alpha, int \doteq \alpha\})$$

$$(\{g(1, a, h(X)) = h(g(Y, b, Y))\},$$
$$\{\alpha_X \doteq int, \alpha_Y \doteq int, \alpha \doteq int\})$$

false

Soundness

We want to make sure that our algorithm has the expected behaviour. In particular:

Soundness

We want to make sure that our algorithm has the expected behaviour. In particular:

- We always detect ill-typed terms.

Soundness

We want to make sure that our algorithm has the expected behaviour. In particular:

- We always detect ill-typed terms.
- We return an MGU if there is a substitution for which the terms are well-typed, have the same type and are equal.

Soundness

We want to make sure that our algorithm has the expected behaviour. In particular:

- We always detect ill-typed terms.
- We return an MGU if there is a substitution for which the terms are well-typed, have the same type and are equal.
- We return wrong when there is no substitution for which the terms can have the same type and be well-typed.

Soundness

We want to make sure that our algorithm has the expected behaviour. In particular:

- We always detect ill-typed terms.
- We return an MGU if there is a substitution for which the terms are well-typed, have the same type and are equal.
- We return wrong when there is no substitution for which the terms can have the same type and be well-typed.
- We return false if there is a substitution for which the terms are well-typed and have the same type, but they cannot be unified.

Proof Progress

We state the following propositions, that lead to the soundness proof.

Proof Progress

We state the following propositions, that lead to the soundness proof.

- The algorithm terminates. (proved)

Proof Progress

We state the following propositions, that lead to the soundness proof.

- The algorithm terminates. (proved)
- If the algorithm terminates it outputs either a unifier, *false* or *wrong*. (proved)

Proof Progress

We state the following propositions, that lead to the soundness proof.

- The algorithm terminates. (proved)
- If the algorithm terminates it outputs either a unifier, *false* or *wrong*. (proved)
- If the algorithm outputs a unifier, the unifier is a MGU. (use Martelli-Montanari proof)

Proof Progress

We state the following propositions, that lead to the soundness proof.

- The algorithm terminates. (proved)
- If the algorithm terminates it outputs either a unifier, *false* or *wrong*. (proved)
- If the algorithm outputs a unifier, the unifier is a MGU. (use Martelli-Montanari proof)
- If the algorithm outputs *wrong*, then either there is a type error in one of the terms, or there is no substitution for which both terms have the same type.

Proof Progress

We state the following propositions, that lead to the soundness proof.

- The algorithm terminates. (proved)
- If the algorithm terminates it outputs either a unifier, *false* or *wrong*. (proved)
- If the algorithm outputs a unifier, the unifier is a MGU. (use Martelli-Montanari proof)
- If the algorithm outputs *wrong*, then either there is a type error in one of the terms, or there is no substitution for which both terms have the same type.
- If the algorithm outputs *false*, then either there is a substitution for which the terms have the same type, but they do not unify.

The End

Thank you!