Unification

- Unification
- Unification in Prolog

Recall previous example, where we said that Prolog unifies

\[
\text{woman}(X)
\]

with

\[
\text{woman}(\text{mia})
\]

thereby instantiating the variable \(X\) with the atom \(\text{mia}\).

Recall Prolog Terms

Simple Terms
- Constants
- Atoms

Complex Terms
- Variables
- Numbers

Unification

• Working definition:
  • Two terms unify if they are the same term or if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal.

• This means that:
  • \(\text{mia}\) and \(\text{mia}\) unify
  • \(42\) and \(42\) unify
  • \(\text{woman}(\text{mia})\) and \(\text{woman}(\text{mia})\) unify

• This also means that:
  • \(\text{vincent}\) and \(\text{mia}\) do not unify
  • \(\text{woman}(\text{mia})\) and \(\text{woman}(\text{jody})\) do not unify
Unification

- What about the terms:
  - mia and X
  - woman(Z) and woman(mia)

- What about the terms:
  - mia and X
  - woman(Z) and woman(mia)
  - loves(mia,X) and loves(X,vincent)

Instantiations

- When Prolog unifies two terms it performs all the necessary instantiations, so that the terms are equal afterwards
- This makes unification a powerful programming mechanism

Revised Definition 1/3

1. If T₁ and T₂ are constants, then T₁ and T₂ unify if they are the same atom, or the same number.

Revised Definition 2/3

1. If T₁ and T₂ are constants, then T₁ and T₂ unify if they are the same atom, or the same number.
2. If T₁ is a variable and T₂ is any type of term, then T₁ and T₂ unify, and T₁ is instantiated to T₂. (and vice versa)

Revised Definition 3/3

1. If T₁ and T₂ are constants, then T₁ and T₂ unify if they are the same atom, or the same number.
2. If T₁ is a variable and T₂ is any type of term, then T₁ and T₂ unify if T₁ = T₂ or T₁ does not occur in T₂.
3. If T₁ and T₂ are complex terms then they unify if:
   a) They have the same functor and arity, and
   b) all their corresponding arguments unify, and
   c) the variable instantiations are compatible.
Prolog unification: =/2

?- mia = mia.
yes
?-

?– mia = mia.
yes
?- mia = vincent.
no
?–

Prolog unification: =/2

?- mia = X.
X=mia
yes
?–

How will Prolog respond?

?- X=mia, X=vincent.

How will Prolog respond?

?- X=mia, X=vincent.

Example with complex terms

?- k(s(g),Y) = k(X,t(k)).

How will Prolog respond?

?- X=mia, X=vincent.

no
?–

Why? After working through the first goal, Prolog has instantiated X with mia, so that it cannot unify it with vincent anymore. Hence the second goal fails.
Example with complex terms

?- k(s(g), Y) = k(X, t(k)).
X = s(g)
Y = t(k)
yes
?- 

Example with complex terms

?- k(s(g), t(k)) = k(X, t(Y)).
X = s(g)
Y = k
yes
?- 

Example with complex terms

?- k(s(g), t(k)) = k(X, t(Y)).
X = s(g)
Y = k
yes
?- 

One more example

?- loves(X, X) = loves(marsellus, mia).

Unification Algorithm as a set of 5 rules

- Let x, y denote variables, s, t denote terms, f, g denote functionals.
- Let X = { s =? t }, where s and t are to be unified.
- **Deletion**: { t =? t } U X => X
- **Decomposition**: { f(s_1, ..., s_n) =? f(t_1, ..., t_n) } U X => { s_1 =? t_1, ..., s_n = t_n } U X.
- **Clash**: { f(s_1, ..., s_n) =? g(t_1, ..., t_m) } U X => fail
- **Occur check**: { x =? f(...) } U X => fail
- **Substitution**: { x =? t } U X => { x <- t } U X[x <- t]

Complexity of unification

- Depending on the order of rules and the representation of the unifiers, the complexity can be either linear (in terms of the size of terms) or exponential.
- Example:
  - t = g(x_1, g(x_2, g(x_3, g(x_n, a))))
  - s = g(f(x_2, x_3), g(f(x_3, x_4), ... g(a, a)))
Composition of substitutions

- A substitution is a mapping from variables to terms.
- If $\alpha$, $\beta$ and $\gamma$ are substitutions and $\gamma(t) = \beta(\alpha(t))$ for any term $t$, then we say $\gamma = \alpha \beta$, a composition of $\alpha$ and $\beta$.
- Given two terms $s$ and $t$, we say $\alpha$ is the most general unifier of $s$ and $t$, if for any unifier $\gamma$ of $s$ and $t$, there exists $\beta$ such that $\gamma = \alpha \beta$.

Prolog and unification

- Prolog does not use a standard unification algorithm
- Consider the following query:

  ?- father(X) = X.

- Do these terms unify or not?

Infinite terms

?- father(X) = X.
X=father(\text{father(\text{father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(father(}

Infinite terms

?- father(X) = X.
X=father(**)
yes
?-

Occurs Check

- A standard unification algorithm carries out an occurs check
- If it is asked to unify a variable with another term it checks whether the variable occurs in the term
- In Prolog:

  ?- unify_with_occurs_check(father(X), X).
  no

Programming with Unification

vertical(line(point(X,Y), point(X,Z)));
horizontal(line(point(X,Y), point(Z,Y))).
Programming with Unification

vertical( line(point(X,Y), point(X,Z))).
horizontal( line(point(X,Y), point(Z,Y))).

?- vertical(line(point(1,1),point(1,3))).
yes
?

?- vertical(line(point(1,1),point(1,3))).
yes
?

?- vertical(line(point(1,1),point(3,2))).
no
?

?- horizontal(line(point(2,3),Point)).
Point = point(2,3).
no
?

Exercise: unification