

### Unification

---

- Unification
- Unification in Prolog

### Unification

---

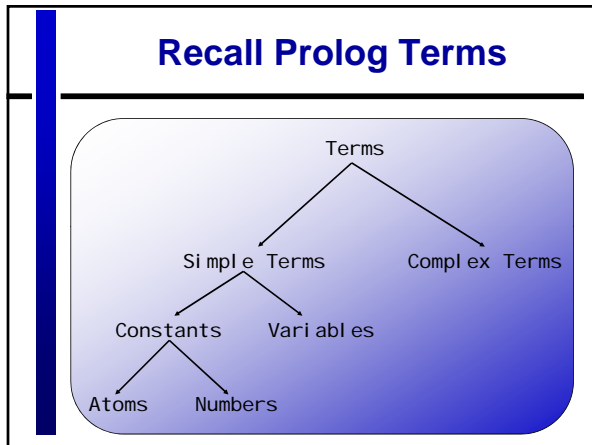
- Recall previous example, where we said that Prolog unifies

**woman(X)**

with

**woman(mia)**

thereby instantiating the variable **X** with the atom **mia**.



### 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.

### Unification

---

- This means that:
  - **mia** and **mia** unify
  - **42** and **42** unify
  - **woman(mia)** and **woman(mia)** unify
- This also means that:
  - **vincent** and **mia** do not unify
  - **woman(mia)** and **woman(jody)** do not unify

### Unification

---

- What about the terms:
  - **mia** and **X**

## Unification

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

## Unification

- 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  $T_1$  and  $T_2$  are constants, then  $T_1$  and  $T_2$  unify if they are the same atom, or the same number.

## Revised Definition 2/3

1. If  $T_1$  and  $T_2$  are constants, then  $T_1$  and  $T_2$  unify if they are the same atom, or the same number.
2. If  $T_1$  is a variable and  $T_2$  is any type of term, then  $T_1$  and  $T_2$  unify, and  $T_1$  is instantiated to  $T_2$ . (and vice versa)

## Revised Definition 3/3

1. If  $T_1$  and  $T_2$  are constants, then  $T_1$  and  $T_2$  unify if they are the same atom, or the same number.
2. If  $T_1$  is a variable and  $T_2$  is any type of term, then  $T_1$  and  $T_2$  unify if  $T_1 = T_2$  or  $T_1$  does not occur in  $T_2$ .
3. If  $T_1$  and  $T_2$  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  
?-

### Prolog unification: =/2

?- 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.  
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)).

### 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)).

### 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 functors.
- Let  $X = \{ s =? t \}$ , where  $s$  and  $t$  are to be unified.
- **Deletion:**  $\{ t =? t \} \cup X \rightarrow X$
- **Decomposition:**  $\{ f(s_1, \dots, s_n) =? f(t_1, \dots, t_n) \} \cup X \rightarrow \{ s_1 =? t_1, \dots, s_n =? t_n \} \cup X$ .
- **Clash:**  $\{ f(s_1, \dots, s_n) =? g(t_1, \dots, t_m) \} \cup X \rightarrow \text{fail}$
- **Occur check:**  $\{ x =? f(\dots x \dots) \} \cup X \rightarrow \text{fail}$
- **Substitution:**  $\{ x =? t \} \cup X \rightarrow \{ x \leftarrow t \} \cup X[x \leftarrow t]$

### 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:
  - $t = g(x_1, g(x_2, g(x_3, \dots g(x_n, a))))$
  - $s = g(f(x_2, x_2), g(f(x_3, x_3), g(f(x_4, x_4), \dots g(a, a))))$



### 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  
?-

### 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(3,2))).  
no  
?-

### Programming with Unification

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

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

?- horizontal(line(point(1,1),point(1,Y))).  
Y = 1;  
no  
?-

### Programming with Unification

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

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

?- horizontal(line(point(2,3),Point)).  
Point = point(\_554,3);  
no  
?-

### Exercise: unification