Logic Programming

Georg Moser

Department of Computer Science @ UIBK

Winter 2016

Summary of Last Lecture

Definition

- goals (aka formulas) are constants or compound terms
- goals are typically non-ground

Definitions

- a clause or rule is an universally quantified logical formula of the form

  ```
  A :- B1, B2, ..., Bn.
  ```

  where A and the B_i's are goals
- A is called the head of the clause; the B_i's are called the body
- a rule of the form `A :-` is called a fact; we write facts simply A.

Definition

a logic program is a finite set of clauses

Notation

- $A \leftarrow A_1, \ldots, A_m$ instead of A `:-` $A_1, \ldots, A_m$`.` for rules
- $\leftarrow A_1, \ldots, A_m$ instead of `?-` $A_1, \ldots, A_m$`.` for queries

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, logic foundations, unification, semantics, database and recursive programming, termination, complexity

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics (revisted), correctness proofs, meta-logical predicates, cuts non-deterministic programming, efficient programs, complexity

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, logic foundations, unification, semantics, database and recursive programming, termination, complexity

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics (revisted), correctness proofs, meta-logical predicates, cuts non-deterministic programming, efficient programs, complexity

Example

```
child_of(joseph_I, leopold_I).
child_of(karl_VI, leopold_I).
child_of(maria_theresia, karl_VI).
child_of(joseph_II, maria_theresia).
child_of(joseph_II, franz_I).
child_of(leopold_II, maria_theresia).
child_of(leopold_II, franz_I).
child_of(maria_antoinette, maria_theresia).
child_of(franz_II, leopold_II).

male(franz_I).          female(maria_theresia).
male(franz_II).         female(marie_antoinette).
male(joseph_I).
male(joseph_II).
male(kar_VI).
male(leopold_I).
male(leopold_II).

husband_wife(franz_I, maria_theresia).
```

Review of Basic Constructs

Definitions

- a fact describes a relation (predicate) between terms

```
child_of(joseph_II, maria_theresia).
```

  which reads "Joseph II is the child of Maria Theresia."
- child_of is the name of the relation
- the arity denotes the number of arguments
- predicates are also denoted as child_of/2
- fact that do not contain variables are ground

Review of Basic Constructs

Definitions

- a fact describes a relation (predicate) between terms

```
child_of(joseph_II, maria_theresia).
```

  which reads “Joseph II is the child of Maria Theresia.”
- child_of is the name of the relation
- the arity denotes the number of arguments
- predicates are also denoted as child_of/2
- fact that do not contain variables are ground

Fact

the order of the arguments is essential, hence it is important to choose meaningful names for predicates

Choosing Names

1. describe the arguments

```
typ1_typ2_typ3_typ4(Arg1, Arg2, Arg3, Arg4)
```

2. refine the name

```
person_person(X,Y).                % too coarse
child_person(Child, Person)        % better
child_parent(Child, Parent)        % perfect
```

3. indicate the relation

```
child_ofparent(Child, Parent)              % preposition
expression_improvedprogram(Exp, IExp)      % participle
expr_improved(Exp, IExp)
consists_of(X,Y)                           % verb
```

4. abbreviations

```
country_/8
```

Definition

- a query tests whether a relation holds

```
:- child_of(joseph_II, maria_theresia).
```

- queries are equivalent to use cases, as they are checked whenever the program is compiled

Definition

- a query tests whether a relation holds

```
:- child_of(joseph_II, maria_theresia).
```

- queries are equivalent to use cases, as they are checked whenever the program is compiled

Why does a Query fail?

1. the query doesn't follow from the data represented in the program; the negation of the query does not necessarily hold
2. the program is a complete representation; the negation of the query does hold

Definition

- a query tests whether a relation holds

```
:- child_of(joseph_II, maria_theresia).
```

- queries are equivalent to use cases, as they are checked whenever the program is compiled

Why does a Query fail?

1. the query doesn't follow from the data represented in the program; the negation of the query does not necessarily hold
2. the program is a complete representation; the negation of the query does hold

Fact

Horn logic cannot distinguish between these options

Some Background in Logic

Fact

a rule

```
mother_of(Mum, Child) :-
    child_of(Child, Mum),
    female(Mum).
```

represents a logical formula:

$$\forall x_{Mum} \forall x_{Child} \left(\text{Child\_of}(x_{Child}, x_{Mum}) \wedge \text{Female}(x_{Mum}) \rightarrow \right.$$
$$\left. \rightarrow \text{Mother\_of}(x_{Mum}, x_{Child}) \right)$$

Some Background in Logic

Fact

a rule

```
mother_of(Mum, Child) :-
    child_of(Child, Mum),
    female(Mum).
```

represents a logical formula:

$$\forall x_{Mum} \forall x_{Child} \left(\text{Child\_of}(x_{Child}, x_{Mum}) \wedge \text{Female}(x_{Mum}) \rightarrow \right.$$
$$\left. \rightarrow \text{Mother\_of}(x_{Mum}, x_{Child}) \right)$$

Definition

formulas of this form are called Horn formulas (or Horn Clauses); thus a logic program is a set of Horn formulas

Computation is Inference

Fact

let P be a program and G a goal; a computation of G from P is the verification of a logical consequence: $P \models G$

Computation is Inference

Fact

let P be a program and G a goal; a computation of G from P is the verification of an inference: $P \vdash G$

Computation is Inference

Fact

let P be a program and G a goal; a computation of G from P is the verification of an inference: $P \vdash G$

Fact (revisited)

Horn logic cannot distinguish whether or not P represents the specification completely

Computation is Inference

Fact

let P be a program and G a goal; a computation of G from P is the verification of an inference: $P \vdash G$

Fact (revisited)

Horn logic cannot distinguish whether or not P represents the specification completely

Why only Horn formulas?

consider the "program":

$$\forall x\,(\mathrm{Even}(x) \vee \mathrm{Odd}(x))$$

then Even(1) or Odd(1) follows as consequence; that is, the program semantic is non-deterministic

Definition

a negative query verifies that the goal fails

```
:/- child_of(joseph_II, friedrich_II).
```

Definition

a negative query verifies that the goal fails

```
:/- child_of(joseph_II, friedrich_II).
```

Definition

a general query with variables provide answer substitutions

```
:- child_of(Child, maria_theresia).
:- child_of(_Child, maria_theresia).
:/- child_of(Child,Child).
```

NB: occurring variables are existentially quantified (inside negation)

Definition

a negative query verifies that the goal fails

```
:/- child_of(joseph_II, friedrich_II).
```

Definition

a general query with variables provide answer substitutions

```
:- child_of(Child, maria_theresia).
:- child_of(_Child, maria_theresia).
:/- child_of(Child,Child).
```

NB: occurring variables are existentially quantified (inside negation)

Definition

a complex query combines several goals and typically make use of shared variables

```
:- child_of(joseph_II, Mum), female(Mum).
```

How to Read a Program

- procedurally: look at the inference steps
- declarative: look at the consequence relation

How to Read a Program

- procedurally: look at the inference steps
- declarative: look at the consequence relation

Tower of Hanoi in Prolog

```
hanoi(0,_,_,_).
hanoi(N,X,Y,Z) :-
    N > 0, M is N-1,
    hanoi(M,X,Z,Y),
    move(N,X,Y),
    hanoi(M,Z,Y,X).

move(D,X,Y) :-
    write('move disk '), write(D),
    write(' from '), write(X),
    write(' to '), write(Y), nl.

?- hanoi(4,a,c,b).
```

Recursive Rules

Example

```
grandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), parent(Person,Descendant).

greatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), grandpartent(Person,Descendant).

greatgreatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), greatgrandpartent(Person,Descendant).
```

Recursive Rules

Example

```
grandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), parent(Person,Descendant).

greatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), grandpartent(Person,Descendant).

greatgreatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), greatgrandpartent(Person,Descendant).
 ⋮
```

Recursive Rules

Example

```
grandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), parent(Person,Descendant).

greatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), grandpartent(Person,Descendant).

greatgreatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), greatgrandpartent(Person,Descendant).
  ⋮
```

Example

```
ancestor(Ancestor,Descendant) :-
  parent(Ancestor,Person), ancestor(Person,Descendant).
```

Recursive Rules

Example

```
grandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), parent(Person,Descendant).

greatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), grandpartent(Person,Descendant).

greatgreatgrandpartent(Ancestor,Descendant) :-
  parent(Ancestor,Person), greatgrandpartent(Person,Descendant).
  ⋮
```

Example

```
ancestor(Ancestor,Descendant) :-
  parent(Ancestor,Person), ancestor(Person,Descendant).

ancestor(Ancestor,Descendent) :- parent(Ancestor,Descendent).
```

Definition

- a rule consists of a head and a body, separated by ":-"

```
mother_of(Mum, Child) :-
  child_of(Child, Mum),
  female(Mum).
```

- a rule is recursive, if the body contains the predicate in the head

Definition

- a rule consists of a head and a body, separated by ":-"

```
mother_of(Mum, Child) :-
    child_of(Child, Mum),
    female(Mum).
```

- a rule is recursive, if the body contains the predicate in the head

Definitions

- we distinguish between the set of solutions of a query and the sequence of solutions
- the sequence may contain redundant solutions
- redundant solutions may be due to existential variables

Example

```
% recursive rule
  married_with(Husband, Wife) :-
    husband_wife(Husband, Wife).
  married_with(PersonA, PersonB) :-
    married_with(PersonB, PersonA).

% non-recursive rule
  married_with(Husband, Wife) :-
    husband_wife(Husband, Wife).
  married_with(Wife, Husband) :-
    husband_wife(Husband, Wife).
```

Example

```
ancestor_of(Ancestor, Descendant) :-
    child_of(Descendant, Ancestor).
ancestor_of(Ancestor, Descendant) :-
    child_of(Person, Ancestor),
    ancestor_of(Person, Descendant).

:/- ancestor_of(X,X).
```

Example

```
ancestor_of(Ancestor, Descendant) :-
    child_of(Descendant, Ancestor).
ancestor_of(Ancestor, Descendant) :-
    child_of(Person, Ancestor),
    ancestor_of(Person, Descendant).

:/- ancestor_of(X,X).
```

Example

```
ancestor_of_2(Ancestor, Descendant) :-
    child_of(Descendant, Ancestor).
ancestor_of_2(Ancestor, Descendant) :-
    ancestor_of_2(Person, Descendant),
    child_of(Person, Ancestor).

:/- ancestor_of_2(X,X).
```

Definition

composition of substitutions

$$\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$$

and

$$\sigma = \{Y_1 \mapsto s_1, \ldots, Y_k \mapsto s_k\}$$

is substitution

$$\theta\sigma = \{X_1 \mapsto t_1\sigma, \ldots, X_n \mapsto t_n\sigma\} \cup \{Y_i \mapsto s_i \mid Y_i \notin \{X_1, \ldots, X_n\}\}$$

Definition

composition of substitutions

$$\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$$

and

$$\sigma = \{Y_1 \mapsto s_1, \ldots, Y_k \mapsto s_k\}$$

is substitution

$$\theta\sigma = \{X_1 \mapsto t_1\sigma, \ldots, X_n \mapsto t_n\sigma\} \cup \{Y_i \mapsto s_i \mid Y_i \notin \{X_1, \ldots, X_n\}\}$$

Example

$\theta = \{X \mapsto g(Y, Z), Y \mapsto a\}$

$\sigma = \{X \mapsto f(Y), Z \mapsto f(X)\}$

Definition

composition of substitutions

$$\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$$

and

$$\sigma = \{Y_1 \mapsto s_1, \ldots, Y_k \mapsto s_k\}$$

is substitution

$$\theta\sigma = \{X_1 \mapsto t_1\sigma, \ldots, X_n \mapsto t_n\sigma\} \cup \{Y_i \mapsto s_i \mid Y_i \notin \{X_1, \ldots, X_n\}\}$$

Example

$\theta = \{X \mapsto g(Y,Z), Y \mapsto a\}$ $\quad \theta\sigma = \{X \mapsto g(Y, f(X)), Y \mapsto a, Z \mapsto f(X)\}$

$\sigma = \{X \mapsto f(Y), Z \mapsto f(X)\}$

Definition

composition of substitutions

$$\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$$

and

$$\sigma = \{Y_1 \mapsto s_1, \ldots, Y_k \mapsto s_k\}$$

is substitution

$$\theta\sigma = \{X_1 \mapsto t_1\sigma, \ldots, X_n \mapsto t_n\sigma\} \cup \{Y_i \mapsto s_i \mid Y_i \notin \{X_1, \ldots, X_n\}\}$$

Example

$$\theta = \{X \mapsto g(Y, Z), Y \mapsto a\} \qquad \theta\sigma = \{X \mapsto g(Y, f(X)), Y \mapsto a, Z \mapsto f(X)\}$$

$$\sigma = \{X \mapsto f(Y), Z \mapsto f(X)\} \qquad \sigma\theta = \{X \mapsto f(a), Z \mapsto f(g(Y, Z)), Y \mapsto a\}$$

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$

Definition

- substitution θ is at least as general as substitution σ if $\exists \mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable:

$\{X \mapsto h(a), Y \mapsto a, Z \mapsto h(a), U \mapsto a\}$

$\{X \mapsto h(U), Y \mapsto U, Z \mapsto h(U)\}$

$\{X \mapsto h(g(U)), Y \mapsto g(U), Z \mapsto h(g(U)), U \mapsto g(U)\}$

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable:

$\{X \mapsto h(a), Y \mapsto a, Z \mapsto h(a), U \mapsto a\}$

$\{X \mapsto h(U), Y \mapsto U, Z \mapsto h(U)\}$ mgu

$\{X \mapsto h(g(U)), Y \mapsto g(U), Z \mapsto h(g(U)), U \mapsto g(U)\}$

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable:

$\{X \mapsto h(a), Y \mapsto a, Z \mapsto h(a), U \mapsto a\}$	$\{U \mapsto a\}$
$\{X \mapsto h(U), Y \mapsto U, Z \mapsto h(U)\}$	mgu
$\{X \mapsto h(g(U)), Y \mapsto g(U), Z \mapsto h(g(U)), U \mapsto g(U)\}$	$\{U \mapsto g(U)\}$

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable:

$\{X \mapsto h(a), Y \mapsto a, Z \mapsto h(a), U \mapsto a\}$	$\{U \mapsto a\}$
$\{X \mapsto h(U), Y \mapsto U, Z \mapsto h(U)\}$	mgu
$\{X \mapsto h(g(U)), Y \mapsto g(U), Z \mapsto h(g(U)), U \mapsto g(U)\}$	$\{U \mapsto g(U)\}$

Theorem

- *unifiable terms have mgu*

Definition

- substitution θ is at least as general as substitution σ if $\exists\mu\ \theta\mu = \sigma$
- unifier of set S of terms is substitution θ such that $\forall s, t \in S\ s\theta = t\theta$
- most general unifier (mgu) is at least as general as any other unifier

Example

terms $f(X, g(Y), X)$ and $f(Z, g(U), h(U))$ are unifiable:

$\{X \mapsto h(a), Y \mapsto a, Z \mapsto h(a), U \mapsto a\}$	$\{U \mapsto a\}$
$\{X \mapsto h(U), Y \mapsto U, Z \mapsto h(U)\}$	mgu
$\{X \mapsto h(g(U)), Y \mapsto g(U), Z \mapsto h(g(U)), U \mapsto g(U)\}$	$\{U \mapsto g(U)\}$

Theorem

- *unifiable terms have mgu*
- $\exists$ *algorithm to compute mgu*

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n$ is called an equality problem

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n$ is called an equality problem
- if $E = X_1 \stackrel{?}{=} v_1, \ldots, X_n \stackrel{?}{=} v_n$, with X_i pairwise distinct and $X_i \notin \mathcal{V}\mathrm{ar}(v_j)$ for all i, j, then E is in solved form

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \dots, u_n \stackrel{?}{=} v_n$ is called an equality problem
- if $E = X_1 \stackrel{?}{=} v_1, \dots, X_n \stackrel{?}{=} v_n$, with X_i pairwise distinct and $X_i \notin \mathcal{V}\mathrm{ar}(v_j)$ for all i, j, then E is in solved form
- let $E = X_1 \stackrel{?}{=} v_1, \dots, X_n \stackrel{?}{=} v_n$ be a equality problem in solved form E induces substitution $\sigma_E = \{X_1 \mapsto v_1, \dots, X_n \mapsto v_n\}$

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n$ is called an equality problem
- if $E = X_1 \stackrel{?}{=} v_1, \ldots, X_n \stackrel{?}{=} v_n$, with X_i pairwise distinct and $X_i \notin \mathcal{V}\mathrm{ar}(v_j)$ for all i, j, then E is in solved form
- let $E = X_1 \stackrel{?}{=} v_1, \ldots, X_n \stackrel{?}{=} v_n$ be a equality problem in solved form E induces substitution $\sigma_E = \{X_1 \mapsto v_1, \ldots, X_n \mapsto v_n\}$

Unification Algorithm

$$u \stackrel{?}{=} u, E \Rightarrow E$$

$$f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n), E \Rightarrow s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n, E$$

$$f(s_1, \ldots, s_n) \stackrel{?}{=} g(t_1, \ldots, t_n), E \Rightarrow \bot \quad f \neq g$$

$$X \stackrel{?}{=} t, E \Rightarrow X \stackrel{?}{=} t, E\{X \mapsto t\} \quad X \in \mathcal{V}\mathrm{ar}(E), X \notin \mathcal{V}\mathrm{ar}(t)$$

$$X \stackrel{?}{=} t, E \Rightarrow \bot \quad X \neq t, X \in \mathcal{V}\mathrm{ar}(t)$$

$$t \stackrel{?}{=} X, E \Rightarrow X \stackrel{?}{=} t, E \quad t \notin \mathcal{V}$$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U))$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U))$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$$f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U)) \;\Rightarrow\; X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), X \stackrel{?}{=} h(U)$$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$$f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U)) \Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), X \stackrel{?}{=} h(U)$$

$$\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), Z \stackrel{?}{=} h(U)$$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$$
\begin{aligned}
f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U)) &\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), X \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), Z \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, Y \stackrel{?}{=} U, Z \stackrel{?}{=} h(U)
\end{aligned}
$$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$$
\begin{aligned}
f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U)) &\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), X \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), Z \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, Y \stackrel{?}{=} U, Z \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} h(U), Y \stackrel{?}{=} U, Z \stackrel{?}{=} h(U)
\end{aligned}
$$

Theorem

1. *equality problems E is unifiable iff the unification algorithm stops with a solved form*
2. *if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is mgu of E*

Example

$$
\begin{aligned}
f(X, g(Y), X) \stackrel{?}{=} f(Z, g(U), h(U)) &\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), X \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, g(Y) \stackrel{?}{=} g(U), Z \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} Z, Y \stackrel{?}{=} U, Z \stackrel{?}{=} h(U) \\
&\Rightarrow X \stackrel{?}{=} h(U), Y \stackrel{?}{=} U, Z \stackrel{?}{=} h(U) \quad \text{mgu}
\end{aligned}
$$