Automated Theorem Proving

Georg Moser

Institute of Computer Science @ UIBK

Winter 2015

Summary of Last Lecture

Gilmore's Prover in Pseudo-Code

```
begin {
  contr := false;
  n := 0;
  while (not contr) do {
    D' := DNF(C'_n);
    contr := all constituents of D'
             contain complementary literals;
    n := n + 1;
   }
 }
```

Disadvantages

- generation of all $\mathcal{C}'_n$
- transformation to DNF
- did not yield actual proofs of simple (predicate logic) formulas

Outline of the Lecture

Early Approaches in Automated Reasoning

Herbrand's theorem for dummies, Gilmore's prover, method of Davis and Putnam

Starting Points

resolution, tableau provers, Skolemisation, ordered resolution, redundancy and deletion

Automated Reasoning with Equality

paramodulation, ordered completion and proof orders, superposition

Applications of Automated Reasoning

Neuman-Stubblebinde Key Exchange Protocol, Robbins problem

Outline of the Lecture

Early Approaches in Automated Reasoning

Herbrand's theorem for dummies, Gilmore's prover, method of Davis and Putnam

Starting Points

resolution, tableau provers, Skolemisation, ordered resolution, redundancy and deletion

Automated Reasoning with Equality

paramodulation, ordered completion and proof orders, superposition

Applications of Automated Reasoning

Neuman-Stubblebinde Key Exchange Protocol, Robbins problem

Definitions

- a clause C is called reduced, if every literal occurs at most once in C
- a clause set $\mathcal{C}$ is called reduced for tautologies, if every clause in $\mathcal{C}$ is reduced and does not contain complementary literals

Definitions

- a clause C is called reduced, if every literal occurs at most once in C
- a clause set $\mathcal{C}$ is called reduced for tautologies, if every clause in $\mathcal{C}$ is reduced and does not contain complementary literals

Definition (tautology rule)

delete all clauses containing complementary literals

Definitions

- a clause C is called reduced, if every literal occurs at most once in C
- a clause set $\mathcal{C}$ is called reduced for tautologies, if every clause in $\mathcal{C}$ is reduced and does not contain complementary literals

Definition (tautology rule)

delete all clauses containing complementary literals

let $\mathcal{C}'$ be ground and reduced for tautologies

Definitions

- a clause C is called reduced, if every literal occurs at most once in C
- a clause set $\mathcal{C}$ is called reduced for tautologies, if every clause in $\mathcal{C}$ is reduced and does not contain complementary literals

Definition (tautology rule)

delete all clauses containing complementary literals

let $\mathcal{C}'$ be ground and reduced for tautologies

Definition (one-literal rule)

let $C \in \mathcal{C}'$ and suppose

1. C consists of just one literal L
2. remove all clauses $D \in \mathcal{C}'$ such that L occurs in D
3. remove $\neg L$ from all remaining clauses in $\mathcal{C}'$

Definition (pure literal rule)

let $\mathcal{D}' \subseteq \mathcal{C}'$ such that

1. $\exists$ literal L that appears in all clauses in $\mathcal{D}'$
2. $\neg L$ doesn't appear in $\mathcal{C}'$
3. replace $\mathcal{C}'$ by $\mathcal{C}' \setminus \mathcal{D}'$

Definition (pure literal rule)

let $\mathcal{D}' \subseteq \mathcal{C}'$ such that

1. $\exists$ literal L that appears in all clauses in $\mathcal{D}'$
2. $\neg L$ doesn't appear in $\mathcal{C}'$
3. replace $\mathcal{C}'$ by $\mathcal{C}' \setminus \mathcal{D}'$

Definition (splitting rule)

suppose the clause set $\mathcal{C}'$ can be written as
$\mathcal{C}' = \{A_1, \dots, A_n, B_1, \dots, B_m\} \cup \mathcal{D}$ where

1. $\exists$ literal L, such that neither L nor $\neg L$ occurs in $\mathcal{D}$
2. L occurs in any A_i (but in no B_j); A_i' is the result of removing L
3. $\neg L$ occurs in any B_j (but in no A_i) B_j' is the result of removing $\neg L$
4. rule consists in splitting $\mathcal{C}'$ into $\mathcal{C}_1' = \{A_1', \dots, A_n'\} \cup \mathcal{D}$ and $\mathcal{C}_2' = \{B_1', \dots, B_m'\} \cup \mathcal{D}$

The Method of Davis and Putnam (for Ground Clauses)

Fact

the method encompasses the above defined four rules

- *tautology rule*
- *one-literal rule*
- *pure literal rule*
- *splitting rule*

The Method of Davis and Putnam (for Ground Clauses)

Fact

the method encompasses the above defined four rules

- *tautology rule*
- *one-literal rule*
- *pure literal rule*
- *splitting rule*

Theorem

1. *the rules of the DPLL-method are correct*
2. *that is, if $\mathcal{D}$ is a set of ground clauses and either $\mathcal{D}'$ or $\mathcal{D}_1$ and $\mathcal{D}_2$ are obtained by the above rules, then $\mathcal{D}$ is satisfiable if $\mathcal{D}'$ ($\mathcal{D}_1$ or $\mathcal{D}_2$) is satisfiable*

let $\mathcal{C}'$ be a set of reduced ground clauses

let $\mathcal{C}'$ be a set of reduced ground clauses

Definition (DPLL-tree)

- T consists only of the root, labelled by $\mathcal{C}'$
- let N be a node in T, labelled by $\mathcal{D}$; then N is either a
  1. leaf node,
  2. N has one successor N', labelled by $\mathcal{D}'$, where $\mathcal{D}'$ is obtained as the application of tautology, one-literal, pure literal rule to $\mathcal{D}$, or
  3. N has two successors N_1, N_2 labelled by the clause sets obtained by an application of the split rule to $\mathcal{D}$

let $\mathcal{C}'$ be a set of reduced ground clauses

Definition (DPLL-tree)

- T consists only of the root, labelled by $\mathcal{C}'$
- let N be a node in T, labelled by $\mathcal{D}$; then N is either a
  1. leaf node,
  2. N has one successor N', labelled by $\mathcal{D}'$, where $\mathcal{D}'$ is obtained as the application of tautology, one-literal, pure literal rule to $\mathcal{D}$, or
  3. N has two successors N_1, N_2 labelled by the clause sets obtained by an application of the split rule to $\mathcal{D}$

Definition (DPLL-decision tree)

a DPLL-tree is a decision tree for $\mathcal{C}'$ if

1. all leafs are labelled by the empty clause $\square$, or
2. $\exists$ leaf labelled by the empty clause set $\varnothing$

Theorem (Soundness)

- *let* $\mathcal{C}'$ *be a reduced set of ground clauses and let* T *be a decision tree proving satisfiability or unsatisfiability for* $\mathcal{C}'$
- *then* $\mathcal{C}'$ *is satisfiable or unsatisfiable, respectively*

Theorem (Soundness)

- *let $\mathcal{C}'$ be a reduced set of ground clauses and let T be a decision tree proving satisfiability or unsatisfiability for $\mathcal{C}'$*
- *then $\mathcal{C}'$ is satisfiable or unsatisfiable, respectively*

Definition (DPLL Method)

DPLL(a) remove multiple occurrences of literals in $\mathcal{C}'$ to obtain a reduced clause set $\mathcal{D}_1$

DPLL(b) apply the tautology rule exhaustively to $\mathcal{D}_1$ to obtain a reduced clause set $\mathcal{D}_2$ that is reduced for tautologies

DPLL(c) construct a decision tree for $\mathcal{D}_2$.

Theorem (Strong (or Constructive) Completeness)

- *let* $\mathcal{C}'$ *be as above and let* T *be a DPLL-tree for* $\mathcal{C}'$
- *then* T *can be extended to a decision tree for* $\mathcal{C}'$

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

1. $\ell = 0$: $\mathcal{C}'$ is either empty or contains $\square$, T is already a decision tree

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

1. $\ell = 0$: $\mathcal{C}'$ is either empty or contains $\Box$, T is already a decision tree
2. $\ell > 0$: we distinguish
   - T consists only of the root, labelled by $\mathcal{C}'$
   - T contains more than one node

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

1. $\ell = 0$: $\mathcal{C}'$ is either empty or contains $\square$, T is already a decision tree
2. $\ell > 0$: we distinguish
   - T consists only of the root, labelled by $\mathcal{C}'$
     we employ a one-literal, pure literal rule, or a splitting rule; extend T such that the successors nodes are labelled with smaller clause sets; induction hypothesis becomes applicable
   - T contains more than one node

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

1. $\ell = 0$: $\mathcal{C}'$ is either empty or contains $\Box$, T is already a decision tree
2. $\ell > 0$: we distinguish
   - T consists only of the root, labelled by $\mathcal{C}'$
     we employ a one-literal, pure literal rule, or a splitting rule; extend T such that the successors nodes are labelled with smaller clause sets; induction hypothesis becomes applicable
   - T contains more than one node
     let $\mathcal{D}_1, \dots, \mathcal{D}_n$ denote all leaf nodes of T; for at least one of these nodes we can employ one-literal, pure literal rule, or a splitting rule; then we argue as in the first sub-case

Theorem (Strong (or Constructive) Completeness)

- *let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$*
- *then T can be extended to a decision tree for $\mathcal{C}'$*

Proof.

by induction on the number ℓ of atoms in $\mathcal{C}'$

1. $\ell = 0$: $\mathcal{C}'$ is either empty or contains $\Box$, T is already a decision tree
2. $\ell > 0$: we distinguish
   - T consists only of the root, labelled by $\mathcal{C}'$
     we employ a one-literal, pure literal rule, or a splitting rule; extend T such that the successors nodes are labelled with smaller clause sets; induction hypothesis becomes applicable
   - T contains more than one node
     let $\mathcal{D}_1, \dots, \mathcal{D}_n$ denote all leaf nodes of T; for at least one of these nodes we can employ one-literal, pure literal rule, or a splitting rule; then we argue as in the first sub-case

∎

The Method of Davis and Putnam (for First-Order Logic)

Method of Davis and Putnam in Pseudo-Code

```
if 𝒞 does not contain function symbols
then apply DPLL(a)-DPLL(c) on 𝒞'_0
else {
  n := 0;
  contr := false;
  while (¬ contr) do {
    apply DPLL(a)-DPLL(c) on 𝒞'_n;
    if the decision tree proves unsatisfiability,
    then contr := true
    else contr := false;
    n := n + 1;
  }}
```

The Language of Clause Logic (with Equality)

Definition

- individual constants
  $k_0, k_1, \dots, k_j, \dots$ denoted c, d, etc.
- function constants with i arguments
  $f_0^i, f_1^i, \dots, f_j^i, \dots$ denoted f, g, h, etc.
- predicate constants with i arguments
  $R_0^i, R_1^i, \dots, R_j^i, \dots$ denoted P, Q, R, etc.
- variables, collected in $\mathcal{V}$
  $x_0, x_1, \dots, x_j, \dots$ denoted x, y, z, etc.

The Language of Clause Logic (with Equality)

Definition

- individual constants
  $k_0, k_1, \dots, k_j, \dots$ denoted c, d, etc.
- function constants with i arguments
  $f_0^i, f_1^i, \dots, f_j^i, \dots$ denoted f, g, h, etc.
- predicate constants with i arguments
  $R_0^i, R_1^i, \dots, R_j^i, \dots$ denoted P, Q, R, etc.
- variables, collected in $\mathcal{V}$
  $x_0, x_1, \dots, x_j, \dots$ denoted x, y, z, etc.

Definition

- propositional connectives $\neg$, $\vee$
- equality sign $=$

Definition

1. $P(t_1, \ldots, t_n)$ is called an atomic formula if $t_1, \ldots, t_n$ are terms, P a predicate constant

Definition

1. $P(t_1, \dots, t_n)$ is called an atomic formula if $t_1, \dots, t_n$ are terms, P a predicate constant
2. a literal is an atomic formula or its negation

Definition

1. $P(t_1, \ldots, t_n)$ is called an atomic formula if $t_1, \ldots, t_n$ are terms, P a predicate constant
2. a literal is an atomic formula or its negation
3. a clause is disjunction of literals

Definition

1. $P(t_1, \ldots, t_n)$ is called an atomic formula if $t_1, \ldots, t_n$ are terms, P a predicate constant
2. a literal is an atomic formula or its negation
3. a clause is disjunction of literals

Theorem

$\forall$ *first-order sentence* F, $\exists$ *set of clauses* $\mathcal{C} = \{C_1, \ldots, C_m\}$

$$F \approx \forall x_1 \cdots \forall x_n (C_1 \wedge \cdots \wedge C_m)$$

Definition

1. $P(t_1, \ldots, t_n)$ is called an atomic formula if $t_1, \ldots, t_n$ are terms, P a predicate constant
2. a literal is an atomic formula or its negation
3. a clause is disjunction of literals

Theorem

$\forall$ first-order sentence F, $\exists$ set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$

$$F \approx \forall x_1 \cdots \forall x_n (C_1 \wedge \cdots \wedge C_m)$$

Proof.

- let F be a sentence (in standard first-order language)
- there exists $G \approx F$ such that

$$G = \forall x_1 \cdots \forall x_n (H_1(x_1, \ldots, x_n) \wedge \cdots \wedge H_m(x_1, \ldots, x_n))$$

- each H_i $(i = 1, \ldots, m)$ is a disjunction of literals, hence a clause ∎

Definition

1. $\square$ is a clause
2. literals are clauses
3. if C, D are clauses, then $C \vee D$ is a clause

Definition

1. $\Box$ is a clause
2. literals are clauses
3. if C, D are clauses, then $C \lor D$ is a clause

Convention

we use (i) the equivalences $A \equiv \neg\neg A$, A atomic formula, that (ii) disjunction $\lor$ is associative and commutative, and (iii) $\Box \lor \Box = \Box$, and $C \lor \Box = \Box \lor C = C$

Definition

1. $\Box$ is a clause
2. literals are clauses
3. if C, D are clauses, then $C \vee D$ is a clause

Convention

we use (i) the equivalences $A \equiv \neg\neg A$, A atomic formula, that (ii) disjunction $\vee$ is associative and commutative, and (iii) $\Box \vee \Box = \Box$, and $C \vee \Box = \Box \vee C = C$

Definition

- let $\mathcal{T}$ denote the set of terms in our language
- $\mathcal{V}\mathrm{ar}(E)$ denotes set of variables occurring in E
- a substitution σ is a mapping $\mathcal{V} \to \mathcal{T}$
  such that $\sigma(x) = x$, for almost all x
- we write $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$; empty subst. denoted by ϵ

Most General Unifier

application of a substitution σ to expression E is denoted as $E\sigma$; $E\sigma$ is called an instance of E

Definition

- $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, $\tau = \{y_1 \mapsto r_1, \dots, y_1 \mapsto r_m\}$
- composition of σ and τ denoted as $\sigma\tau$:

$$\{x_1 \mapsto t_1\tau, \dots, x_n \mapsto t_n\tau\} \cup \{y_i \mapsto r_i \mid \text{for all } j = 1, \dots, n,\ y_i \neq x_j\}$$

- σ is more general than a substitution τ, if there exists a substitution ρ such that $\sigma\rho = \tau$

Most General Unifier

application of a substitution σ to expression E is denoted as $E\sigma$; $E\sigma$ is called an instance of E

Definition

- $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, $\tau = \{y_1 \mapsto r_1, \dots, y_1 \mapsto r_m\}$
- composition of σ and τ denoted as $\sigma\tau$:

$$\{x_1 \mapsto t_1\tau, \dots, x_n \mapsto t_n\tau\} \cup \{y_i \mapsto r_i \mid \text{for all } j = 1, \dots, n,\ y_i \neq x_j\}$$

- σ is more general than a substitution τ, if there exists a substitution ρ such that $\sigma\rho = \tau$ $E\tau$ is instance of $E\sigma$

Most General Unifier

application of a substitution σ to expression E is denoted as $E\sigma$; $E\sigma$ is called an instance of E

Definition

- $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, $\tau = \{y_1 \mapsto r_1, \dots, y_1 \mapsto r_m\}$
- composition of σ and τ denoted as $\sigma\tau$:

$$\{x_1 \mapsto t_1\tau, \dots, x_n \mapsto t_n\tau\} \cup \{y_i \mapsto r_i \mid \text{for all } j = 1, \dots, n,\ y_i \neq x_j\}$$

- σ is more general than a substitution τ, if there exists a substitution ρ such that $\sigma\rho = \tau$ $\quad$ $E\tau$ is instance of $E\sigma$

Definition

- a substitution σ such that $E\sigma = F\sigma$ is unifier of E, F
  generalises to sets U of expressions ($=$ terms or atomic formulas)
- unifier σ is most general if σ is more general than any other unifier

Example

consider $U = \{\mathsf{P}(x, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(x', y')\}$

- $\sigma = \{x \mapsto 0, y \mapsto 0, x' \mapsto 0, y' \mapsto \mathsf{f}(0)\}$ is a unifier of U
- $\tau = \{y \mapsto x, x' \mapsto x, y' \mapsto \mathsf{f}(x)\}$ is most general

Example

consider $U = \{\mathsf{P}(x, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(x', y')\}$

- $\sigma = \{x \mapsto 0, y \mapsto 0, x' \mapsto 0, y' \mapsto \mathsf{f}(0)\}$ is a unifier of U
- $\tau = \{y \mapsto x, x' \mapsto x, y' \mapsto \mathsf{f}(x)\}$ is most general

Definition

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

Example

consider $U = \{\mathsf{P}(x, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(x', y')\}$

- $\sigma = \{x \mapsto 0, y \mapsto 0, x' \mapsto 0, y' \mapsto \mathsf{f}(0)\}$ is a unifier of U
- $\tau = \{y \mapsto x, x' \mapsto x, y' \mapsto \mathsf{f}(x)\}$ is most general

Definition

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

Example

consider $U = \{\mathsf{P}(x, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(x', y')\}$

- $\sigma = \{x \mapsto 0, y \mapsto 0, x' \mapsto 0, y' \mapsto \mathsf{f}(0)\}$ is a unifier of U
- $\tau = \{y \mapsto x, x' \mapsto x, y' \mapsto \mathsf{f}(x)\}$ is most general

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n$ is called an equality problem
- unifier of E is the unifier of $\{u_1 = v_1, \ldots, u_n = v_n\}$
- If $E = x_1 \stackrel{?}{=} v_1, \ldots, x_n \stackrel{?}{=} v_n$, with x_i pairwise distinct and $x_i \notin \mathcal{V}\mathrm{ar}(v_j)$, then E is in solved form

Example

consider $U = \{\mathsf{P}(x, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(x', y')\}$

- $\sigma = \{x \mapsto 0, y \mapsto 0, x' \mapsto 0, y' \mapsto \mathsf{f}(0)\}$ is a unifier of U
- $\tau = \{y \mapsto x, x' \mapsto x, y' \mapsto \mathsf{f}(x)\}$ is most general

Definition

- sequence $E = u_1 \stackrel{?}{=} v_1, \dots, u_n \stackrel{?}{=} v_n$ is called an equality problem
- unifier of E is the unifier of $\{u_1 = v_1, \dots, u_n = v_n\}$
- If $E = x_1 \stackrel{?}{=} v_1, \dots, x_n \stackrel{?}{=} v_n$, with x_i pairwise distinct and $x_i \notin \mathcal{V}\mathrm{ar}(v_j)$, then E is in solved form

Example

U becomes $\quad \mathsf{P}(x, \mathsf{f}(x)) \stackrel{?}{=} \mathsf{P}(y, \mathsf{f}(x)), \mathsf{P}(y, \mathsf{f}(x)) \stackrel{?}{=} \mathsf{P}(x', y')$

τ becomes $\quad y \stackrel{?}{=} x, x' \stackrel{?}{=} x, y' \stackrel{?}{=} \mathsf{f}(x)$

Unification Algorithm

Unification Algorithm

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

Unification Algorithm

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

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

Unification Algorithm

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

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

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

Unification Algorithm

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

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

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

$$x \stackrel{?}{=} v, E \Rightarrow x \stackrel{?}{=} v, E\{x \mapsto v\} \qquad x \in \mathcal{V}\mathrm{ar}(E), x \notin \mathcal{V}\mathrm{ar}(v)$$

Unification Algorithm

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

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

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

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

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

Unification Algorithm

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

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

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

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

$$x \stackrel{?}{=} v, E \Rightarrow \perp \quad x \neq v, x \in \mathcal{V}\mathrm{ar}(v)$$

$$v \stackrel{?}{=} x, E \Rightarrow x \stackrel{?}{=} v, E \quad v \notin \mathcal{V}$$

Unification Algorithm

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

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

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

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

$$x \stackrel{?}{=} v, E \Rightarrow \perp \quad x \neq v, x \in \mathcal{V}\mathrm{ar}(v)$$

$$v \stackrel{?}{=} x, E \Rightarrow x \stackrel{?}{=} v, E \quad v \notin \mathcal{V}$$

Example

$$\mathrm{f}(x, \mathrm{g}(y), x) \stackrel{?}{=} \mathrm{f}(z, \mathrm{g}(x'), \mathrm{h}(x')) \Rightarrow x \stackrel{?}{=} z, \mathrm{g}(y) \stackrel{?}{=} \mathrm{g}(x'), x \stackrel{?}{=} \mathrm{h}(x')$$

Unification Algorithm

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

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

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

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

$$x \stackrel{?}{=} v, E \Rightarrow \perp \quad x \neq v, x \in \mathcal{V}\mathrm{ar}(v)$$

$$v \stackrel{?}{=} x, E \Rightarrow x \stackrel{?}{=} v, E \quad v \notin \mathcal{V}$$

Example

$$\begin{aligned} \mathsf{f}(x, \mathsf{g}(y), x) \stackrel{?}{=} \mathsf{f}(z, \mathsf{g}(x'), \mathsf{h}(x')) &\Rightarrow x \stackrel{?}{=} z, \mathsf{g}(y) \stackrel{?}{=} \mathsf{g}(x'), x \stackrel{?}{=} \mathsf{h}(x') \\ &\Rightarrow x \stackrel{?}{=} z, \mathsf{g}(y) \stackrel{?}{=} \mathsf{g}(x'), z \stackrel{?}{=} \mathsf{h}(x') \end{aligned}$$

Unification Algorithm

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

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

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

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

$$x \stackrel{?}{=} v, E \Rightarrow \perp \quad x \neq v, x \in \mathcal{V}\mathrm{ar}(v)$$

$$v \stackrel{?}{=} x, E \Rightarrow x \stackrel{?}{=} v, E \quad v \notin \mathcal{V}$$

Example

$$\begin{aligned} \mathsf{f}(x, \mathsf{g}(y), x) \stackrel{?}{=} \mathsf{f}(z, \mathsf{g}(x'), \mathsf{h}(x')) &\Rightarrow x \stackrel{?}{=} z, \mathsf{g}(y) \stackrel{?}{=} \mathsf{g}(x'), x \stackrel{?}{=} \mathsf{h}(x') \\ &\Rightarrow x \stackrel{?}{=} z, \mathsf{g}(y) \stackrel{?}{=} \mathsf{g}(x'), z \stackrel{?}{=} \mathsf{h}(x') \\ &\Rightarrow x \stackrel{?}{=} z, y \stackrel{?}{=} x', z \stackrel{?}{=} \mathsf{h}(x') \end{aligned}$$

Unification Algorithm

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

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

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

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

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

$$v \stackrel{?}{=} x, E \Rightarrow x \stackrel{?}{=} v, E \quad v \notin \mathcal{V}$$

Example

$$\begin{aligned} \mathrm{f}(x, \mathrm{g}(y), x) \stackrel{?}{=} \mathrm{f}(z, \mathrm{g}(x'), \mathrm{h}(x')) &\Rightarrow x \stackrel{?}{=} z, \mathrm{g}(y) \stackrel{?}{=} \mathrm{g}(x'), x \stackrel{?}{=} \mathrm{h}(x') \\ &\Rightarrow x \stackrel{?}{=} z, \mathrm{g}(y) \stackrel{?}{=} \mathrm{g}(x'), z \stackrel{?}{=} \mathrm{h}(x') \\ &\Rightarrow x \stackrel{?}{=} z, y \stackrel{?}{=} x', z \stackrel{?}{=} \mathrm{h}(x') \\ &\Rightarrow x \stackrel{?}{=} \mathrm{h}(x'), y \stackrel{?}{=} x', z \stackrel{?}{=} \mathrm{h}(x') \end{aligned}$$

Definition

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

let $E = x_1 \stackrel{?}{=} v_1, \dots, x_n \stackrel{?}{=} v_n$ be a equality problem in solved form
E induces substitution $\sigma_E = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$

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 a most general unifier (mgu for short) of E;*

Definition

let $E = x_1 \stackrel{?}{=} v_1, \dots, x_n \stackrel{?}{=} v_n$ be a equality problem in solved form
E induces substitution $\sigma_E = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$

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 a most general unifier (mgu for short) of E;*

Proof.

in proof, we verify the following three facts:

Definition

let $E = x_1 \stackrel{?}{=} v_1, \dots, x_n \stackrel{?}{=} v_n$ be a equality problem in solved form
E induces substitution $\sigma_E = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$

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 a most general unifier (mgu for short) of E;*

Proof.

in proof, we verify the following three facts:

- if $E \Rightarrow E'$, then σ is a unifier of E iff σ is a unifier of E'

Definition

let $E = x_1 \stackrel{?}{=} v_1, \dots, x_n \stackrel{?}{=} v_n$ be a equality problem in solved form
E induces substitution $\sigma_E = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$

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 a most general unifier (mgu for short) of E;*

Proof.

in proof, we verify the following three facts:

- if $E \Rightarrow E'$, then σ is a unifier of E iff σ is a unifier of E'
- if $E \Rightarrow^* \bot$, then E is not unifiable

Definition

let $E = x_1 \stackrel{?}{=} v_1, \ldots, x_n \stackrel{?}{=} v_n$ be a equality problem in solved form
E induces substitution $\sigma_E = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$

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 a most general unifier (mgu for short) of E;*

Proof.

in proof, we verify the following three facts:

- if $E \Rightarrow E'$, then σ is a unifier of E iff σ is a unifier of E'
- if $E \Rightarrow^* \bot$, then E is not unifiable
- if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is a mgu of E ∎