3.1 Skolem Normal Form

Freitag, 17. April 2015 08:30

Today: 2 lectures

Monday: exercise course instead of lecture

exercise sheets:

- first sheet due on Monday
- second sheet: on the web, due next Friday
- groups of two or three
- students looking for exercise partners: meet in between the 2 lectures in AH 1

$I \models \varphi$ — I satisfies formula φ

$\Phi \models \varphi$ — Φ entails φ

(Φ: program clauses; φ: query)

means: for every interpretation I:

$I \models \Phi$ implies $I \models \varphi$

Ex: Example LP

Query: ?- motherOf(X, susanne).

This means that we have to check:

$\Phi \models \exists X \;$ motherOf(X, susanne)

(Φ: prog. clauses)

This indeed holds:

$I \models \Phi$

$\wedge \; I \models$ motherOf(renate, susanne)

substitution

$\rightsquigarrow I \models motherOf(X, susanne)\,[X/renate]$

$\rightsquigarrow I[\![X / I(renate)]\!] \models motherOf(X, susanne)$

by the subst. lemma 2.2.3 (a)

$\rightsquigarrow I \models \exists X\ motherOf(X, susanne)$

How does Prolog perform proofs of the form

$\Phi \models \varphi$?

3. Resolution

Problem: Entailment is defined semantically

Not suitable for automation (one would have to check all possible interpretations).

Solution: Check entailment syntactically

Define a calculus with syntactic rules that define when a formula φ can be deduced from Φ.

entailment (semantic)   deduction (syntactic)

Calculus is sound iff deduction $\Rightarrow$ entailment

(i.e. if φ is deduced from Φ, then $\Phi \models \varphi$)

Calculus is complete iff entailment $\Rightarrow$ deduction

Calculus is complete iff entailment $\Rightarrow$ deduction
(i.e., if $\Phi \models \varphi$, then φ can be deduced from Φ).

Unfortunately, entailment in predicate logic is undecidable: There is no program which always terminates and which finds out for any Φ, φ whether $\Phi \models \varphi$.
$\Rightarrow$ there is no automatable, always terminating calculus that is sound + complete.

But: Entailment is semi-decidable
$\Rightarrow$ there is a program such that for every Φ, φ:

prog. terminates with "Yes" iff $\Phi \models \varphi$
(But if $\Phi \not\models \varphi$, then the prog. might not terminate).

We will now introduce such a sound + complete calculus which terminates if $\Phi \models \varphi$, but which might not terminate if $\Phi \not\models \varphi$.

Resolution Calculus: sound, complete, automatable, ~~terminating~~

Plan

- First introduce a simpler calculus

(also sound, complete, automatable).

- Then refine this calculus step by step to increase efficiency.

Idea of the resolution calculus:

express entailment problems

as unsatisfiability problems.

<u>Lemma 3.0.1.</u> (Entailment vs. Unsatisfiability)

Let $\varphi_1, \ldots, \varphi_k, \varphi \in \mathcal{F}(\Sigma, \Delta, \mathcal{V})$.

Then $\{\varphi_1, \ldots, \varphi_k\} \models \varphi$ iff

$\varphi_1 \wedge \ldots \wedge \varphi_k \wedge \neg\varphi$ is unsatisfiable.

<u>Proof:</u> $\{\varphi_1, \ldots, \varphi_k\} \models \varphi$

$\iff$ for all int. $\mathcal{I}$: $\mathcal{I} \models \{\varphi_1, \ldots, \varphi_k\}$ implies $\mathcal{I} \models \varphi$

$\iff$ there is no $\mathcal{I}$ with

$\mathcal{I} \models \{\varphi_1, \ldots, \varphi_k\}$ and $\mathcal{I} \models \neg\varphi$

$\iff$ $\varphi_1 \wedge \ldots \wedge \varphi_k \wedge \neg\varphi$ is unsatisfiable. $\square$

Goal: Check unsatisfiability of formulas automatically.

Since this is undecidable, but semi-decidable:

Develop a technique which always finds out unsatisfiability, but may not terminate for satisfiable formulas.

3.1. Skolem Normal Form

First step to check whether a formula φ is unsatisfiable: transform φ into a normal form:

1. prenex normal form

$$\forall x_1 \exists x_2 \exists x_3 \forall x_4 \quad \psi$$

quantifier-free

2. Skolem normal form

$$\forall x_1 \forall x_2 \ldots \forall x_n \quad \psi$$

no variables except $x_1, .., x_n$

<u>Def 3.1.1.</u> (Prenex NF)

A formula φ is in <u>prenex normal form</u> iff it has the form $Q_1 x_1 \ldots Q_n x_n \psi$ where $Q_1, .., Q_n \in \{\forall, \exists\}$ and ψ is quantifier-free.

<u>Thm 3.1.2.</u> (Transformation to prenex NF)

For every formula φ, one can automatically generate an equivalent formula φ' in prenex normal form.

<u>Proof</u>: An algorithm for this transformation works as follows:

First replace sub-formulas $\varphi_1 \leftrightarrow \varphi_2$

by $(\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$.

Then replace sub-formulas $\varphi_1 \to \varphi_2$

by $\neg \varphi_1 \vee \varphi_2$.

Then use the following alg. PRENEX(φ):

- if φ is quantifier-free then return φ

Then use the following alg. PRENEX(φ):

- if φ is quantifier-free, then return φ
- if $\varphi = \neg \varphi_1$, then compute
  PRENEX(φ_1) $= Q_1 X_1 \dots Q_n X_n\ \psi_1$.
  Return $\overline{Q_1} X_1 \dots \overline{Q_n} X_n\ \neg \psi_1$,
  where $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

$\neg \forall X\, p(X) \Rightarrow \exists X\, \neg p(X)$

- if $\varphi = \varphi_1 \circ \varphi_2$ where $\circ \in \{\wedge, \vee\}$, then compute
  PRENEX(φ_1) $= Q_1 X_1 \dots Q_n X_n\ \psi_1$
  PRENEX(φ_2) $= R_1 Y_1 \dots R_m Y_m\ \psi_2$
  By renaming bound variables, we can ensure that $X_1, \dots, X_n$ do not occur in $R_1 Y_1 \dots R_m Y_m\ \psi_2$ and $Y_1, \dots, Y_m$ do not occur in $Q_1 X_1 \dots Q_n X_n\ \psi_1$.
  Then return:
  $Q X_1 \dots Q_n X_n R_1 Y_1 \dots R_m Y_m\ (\psi_1 \circ \psi_2)$
- if $\varphi = QX\ \varphi_1$ with $Q \in \{\forall, \exists\}$,
  then compute PRENEX(φ_1) $= Q_1 X_1 \dots Q_n X_n\ \psi$.
  By renaming bound variables, we ensure that $X_1, \dots, X_n$ are different from X.
  Then return $QX\, Q_1 X_1 \dots Q_n X_n\ \psi$. $\boxtimes$

Ex. 3.1.3 Transform the following formula to prenex NF:

$$\neg \exists X (\mathit{married}(X,Y) \wedge \underbrace{\neg \exists Y\, \mathit{motherOf}(X,Y)}_{\underbrace{\forall Y \neg \mathit{motherOf}(X,Y)}_{\forall Z \neg \mathit{motherOf}(X,Z)}})$$

$$\forall Z\, \neg motherOf(X,Z)$$

$$\neg \exists X \; \forall Z (married(X,Y) \wedge \neg motherOf(X,Z))$$

$$\forall X \, \exists Z \, \neg (married(X,Y) \wedge \neg motherOf(X,Z))$$

Ex 3.14 Consider our example LP and the query ?- motherOf(X, susanne).

We want to prove

$$motherOf(renate, sus) \models \exists X \; motherOf(X, susanne)$$

To this end, we have to show unsatisfiability of

$$motherOf(ren, sus) \wedge \neg \exists X \, motherOf(X, sus).$$

First, this formula is transformed to prenex NF:

$$\forall X \, (motherOf(ren, sus) \wedge \neg motherOf(X, sus))$$

Def 3.15 (Skolem NF)

A formula φ is in Skolem normal form iff it is closed (i.e., it has no free variables) and it has the form $\forall X_1, \ldots, X_n \; \psi$ where ψ is quantifier-free.

Goal: obtain Skolem NF automatically

Solution: first transform to prenex NF, then remove free variables and $\exists$

There exist formulas φ where there is no equivalent formula φ' in Skolem NF.

Ex: female(X)

$\exists X$ female(X)

But: for every formula φ there exists a "satisfiability-equivalent" formula in Skolem NF.

Thm 3.1.6 (Transf. in Skolem NF)

For every formula φ, one can automatically construct a formula φ' in Skolem normal form such that φ is satisfiable iff φ' is satisfiable.

Proof: First, transform φ to prenex NF as in Thm 3.1.2. This results in φ_1.

Let $X_1, \ldots, X_n$ be the free variables of φ_1.

Then transform φ_1 into

$$\underbrace{\exists X_1, \ldots, X_n \quad \varphi_1}_{\varphi_2}.$$

← This is not equivalent to φ_1, but satisfiability-equivalent.

Finally, remove $\exists$ from φ_2 (φ_2 is closed and in prenex NF).

We remove $\exists$ from the outside to the inside:

If φ_2 has the form $\forall X_1, \ldots X_n \; \exists Y \; \psi$, then replace it by $\forall X_1,\dots,X_n \; \psi[Y/f(X_1,\dots,X_n)]$.

fresh fct. symbol of arity n

This is repeated until all $\exists$ have been removed.

The resulting formula is satisfiability-equivalent to the original formula (follows from substitution lemma). $\boxed{\text{Th}}$

Ex 3.17 In Ex 3.13 we obtained the following formula in prenex NF:

$$\forall X\, \exists Z\, \neg(\text{married}(X,Y) \vee \neg \text{motherOf}(X,Z))$$

$$\Downarrow$$

$$\exists Y\, \forall X\, \exists Z\, \neg(\text{married}(X,Y) \vee \neg \text{motherOf}(X,Z))$$

$$\Downarrow$$

$$\forall X\, \exists Z\, \neg(\text{married}(X,a) \vee \neg \text{motherOf}(X,Z))$$

$$\Downarrow$$

$$\forall X\, \neg(\text{married}(X,a) \vee \neg \text{motherOf}(X,f(X)))$$