

Lecture 14: Theorem Proving 2

10:30 AM, Mar 12, 2009

Contents

1 Overview	1
2 Conversion to Normal Form	1
3 Unification: Definition and Procedure	4
4 Sample Proof-by-Refutation	6

1 Overview

At this point, we have all the machinery in place to automate theorem proving in first-order logic. More specifically, our goal is solve the logical entailment problem: $KB \models A$? At a high-level, our strategy is as follows:

1. Add $A \rightarrow \perp$ to KB.
2. Convert KB to normal form.
3. Using unification and resolution, search for a refutation: i.e., a proof of $\top \rightarrow \perp$.

The rest of this lecture details these steps and traces through an example.

2 Conversion to Normal Form

The following algorithm converts an arbitrary formula of first-order logic into a set of quantifier-free formulas in normal form.

1. Eliminate implications.
 - Rewrite $\phi \rightarrow \psi$ as $\neg\phi \vee \psi$.
 - Rewrite $\phi \leftarrow \psi$ as $\neg\psi \vee \phi$.
2. Move negations inwards.
 - Use DeMorgan's laws:
 - Rewrite $\neg(\phi \vee \psi)$ as $\neg\phi \wedge \neg\psi$.
 - Rewrite $\neg(\phi \wedge \psi)$ as $\neg\phi \vee \neg\psi$.
 - Rewrite $\neg\forall x \phi$ as $\exists x \neg\phi$.

- Rewrite $\neg\exists x \phi$ as $\forall x \neg\phi$.
 - Use the law of double negation:
 - Rewrite $\neg\neg\phi$ as ϕ .
3. Standardize variables apart. If the same variable name appears within the scope of multiple quantifiers, replace these multiple occurrences of the same variable name with distinct variable names. For example, rewrite $\forall x P(x) \vee \forall x Q(x)$ as $\forall x P(x) \vee \forall y Q(y)$. Changing the names of bound variables does not change the meaning of formulas.
 4. Move universal quantifiers left.
 - Rewrite $(\forall x \phi) \wedge \psi$ as $\forall x (\phi \wedge \psi)$.
 - Rewrite $\phi \wedge (\forall x \psi)$ as $\forall x (\phi \wedge \psi)$.
 - Rewrite $(\forall x \phi) \vee \psi$ as $\forall x (\phi \vee \psi)$.
 - Rewrite $\phi \vee (\forall x \psi)$ as $\forall x (\phi \vee \psi)$.

Since variables are standardized apart, this step is sound. Otherwise, if we rewrite $\forall x P(x) \vee \forall x Q(x)$ as $\forall x P(x) \vee Q(x)$, the meaning of the formula changes. For example, let $P = \text{BLACK}$ and $Q = \text{WHITE}$. The latter formula is true of piano keys, all of which are either black or white. But the original formula is not true of piano keys, since it is neither the case that all piano keys are black nor that all piano keys are white.

At this point the formula is in **prenex** normal form.

5. *Skolemize* to eliminate existential quantifiers. Rewrite quantified formulas of the form

$$\forall x_1, \dots, x_n \exists y \phi(x_i, y)$$

as

$$\forall x_1, \dots, x_n \phi(x_i, \text{SK}_m(x_1, \dots, x_n))$$

where SK_m is a new function symbol that does not appear elsewhere in the database. The subscript m denotes the m th Skolemization, thereby ensuring that SK_m has not appeared previously. For example, the Skolemization of this formula:

$$\forall x, y (x < y) \rightarrow \exists z (x < z) \wedge (z < y)$$

is this formula:

$$\forall x, y (x < y) \rightarrow (x < \text{SK}(x, y)) \wedge (\text{SK}(x, y) < y)$$

where $\text{SK}(x, y)$ denotes, for example, the function that averages (the denotations of) x and y . The following example is an incorrect Skolemization. The premise states “everyone has a mother;” the conclusion states “ SK_1 is everyone’s mother.”

$$\frac{\forall x \exists y \text{MOTHER_OF}(x) = y}{\forall x \text{MOTHER_OF}(x) = \text{SK}_1}$$

The following is a correct Skolemization: it states that “ $\text{SK}_1(x)$ is x ’s mother.”

$$\frac{\forall x \exists y \text{MOTHER_OF}(x) = y}{\forall x \text{MOTHER_OF}(x) = \text{SK}_1(x)}$$

Lemma: Given any first-order formula $\forall x_1, \dots, x_n \exists y \psi$ in prenex normal form, there exists a corresponding Skolemized formula $\forall x_1, \dots, x_n \psi|_{\{\text{SK}(x_1, \dots, x_n)/y\}}$ *s.t.* the two are **equisatisfiable**, meaning either both have a model or neither does (though the models need not coincide).

Remark: A formula (in prenex normal form) and its Skolemization are not logically equivalent.

Exercise: Let ϕ be an atomic formula, and let SK be a function symbol not occurring in ϕ . Show that $\forall x \phi|_{\{\text{SK}(x)/y\}} \models \forall x \exists y \phi$, but $\forall x \exists y \phi \not\models \forall x \phi|_{\{\text{SK}(x)/y\}}$.

6. Drop the prefix: *i.e.*, eliminate universal quantifiers. The convention after this point is that all variables are universally quantified.
7. Distribute \vee over \wedge .
 - Rewrite $\phi \vee (\psi \wedge \tau)$ as $(\phi \vee \psi) \wedge (\phi \vee \tau)$.
 - Rewrite $(\phi \wedge \psi) \vee \tau$ as $(\phi \vee \tau) \wedge (\psi \vee \tau)$.

(Note: The knowledge base is now in conjunctive normal form (CNF).)

8. Split conjunctions: split up formulas like $\phi \wedge \psi$ into two separate entries in the knowledge base, namely ϕ and ψ .
9. Flatten nested disjunctions: flatten formulas of the form $((\phi \vee \psi) \vee \chi)$ into $\phi \vee \psi \vee \chi$. (This is often done along the way.)
10. Eliminate negations by reintroducing implications.
 - Rewrite $\neg\phi \vee \psi$ as $\phi \rightarrow \psi$.
 - Rewrite $\neg\phi$ as $\phi \rightarrow \perp$.
 - Rewrite ϕ as $\top \rightarrow \phi$.

Example: [Ginsberg 1993] If a house is big and old, then it is a lot of work to maintain, unless it comes with a housecleaner and doesn't have a garden.

We express this sentence of first-order logic using the predicates BIG, OLD, WORK, CLEANS, and GARDEN, as well as the "type" predicates ISA_HOUSE, ISA_MAN, and ISA_GARDEN:

$$\begin{aligned} &\forall h \text{ (ISA_HOUSE}(h) \wedge \text{BIG}(h) \wedge \text{OLD}(h)) \rightarrow \text{WORK}(h) \vee \\ &(\exists m \text{ ISA_MAN}(m) \wedge \text{CLEANS}(m, h) \wedge \neg \exists g \text{ ISA_GARDEN}(g) \wedge \text{GARDEN}(g, h)) \end{aligned}$$

To demonstrate how to convert a sentence of first-order logic such as this one to normal form, we work with the following simplification which does not include type predicates:

$$\forall h \text{ BIG}(h) \wedge \text{OLD}(h) \rightarrow \text{WORK}(h) \vee (\exists m \text{ CLEANS}(m, h) \wedge \neg \exists g \text{ GARDEN}(g, h))$$

1. Eliminate implications:

$$\forall h \neg (\text{BIG}(h) \wedge \text{OLD}(h)) \vee \text{WORK}(h) \vee (\exists m \text{ CLEANS}(m, h) \wedge \neg \exists g \text{ GARDEN}(g, h))$$

2. Move negations inwards:

$$\forall h \neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee (\exists m \text{CLEANS}(m, h) \wedge \forall g \neg \text{GARDEN}(g, h))$$

3. Standardize variables apart: If the formula were,

$$\forall x \neg \text{BIG}(x) \vee \neg \text{OLD}(x) \vee \text{WORK}(x) \vee (\exists y \text{CLEANS}(y, x) \wedge \forall y \neg \text{GARDEN}(y, x))$$

we could rewrite it as it is written in the previous step.

4. Skolemize to eliminate existential quantifiers:

$$\forall h \neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee (\text{CLEANS}(\text{SK_CLEANS}(h), h) \wedge \forall g \neg \text{GARDEN}(g, h))$$

5. Move universal quantifiers left:

$$\forall h, g \neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee (\text{CLEANS}(\text{SK_CLEANS}(h), h) \wedge \neg \text{GARDEN}(g, h))$$

6. Drop the prefix:

$$\neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee (\text{CLEANS}(\text{SK_CLEANS}(h), h) \wedge \neg \text{GARDEN}(g, h))$$

7. Distribute \vee over \wedge :

$$\begin{aligned} &(\neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee \text{CLEANS}(\text{SK_CLEANS}(h), h)) \wedge \\ &(\neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee \neg \text{GARDEN}(g, h)) \end{aligned}$$

8. Split conjunctions:

$$\begin{aligned} &\neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee \text{CLEANS}(\text{SK_CLEANS}(h), h) \\ &\neg \text{BIG}(h) \vee \neg \text{OLD}(h) \vee \text{WORK}(h) \vee \neg \text{GARDEN}(g, h) \end{aligned}$$

9. Flatten nested disjunctions. (We've been doing that along the way.)

10. Eliminate negations by reintroducing implications:

$$\begin{aligned} &\text{BIG}(h) \wedge \text{OLD}(h) \rightarrow \text{WORK}(h) \vee \text{CLEANS}(\text{SK_CLEANS}(h), h) \\ &\text{BIG}(h) \wedge \text{OLD}(h) \wedge \text{GARDEN}(g, h) \rightarrow \text{WORK}(h) \end{aligned}$$

3 Unification: Definition and Procedure

An **expression** is a term or a formula of first-order logic. Given two expressions e_1, e_2 , a **unifier** of e_1 and e_2 is a substitution σ of terms to variables *s.t.* $e_1|_\sigma = e_2|_\sigma$. Given two substitutions σ and τ that unify expressions e_1 and e_2 , σ is more general than τ iff there exists θ *s.t.* $e_1|_\tau = e_1|_{\sigma\theta} = e_2|_{\sigma\theta} = e_2|_\tau$. Intuitively, σ is more general than τ iff σ is less constraining than τ iff σ binds fewer variables to terms than τ . As an example, consider the expressions $g(f(x))$ and $g(y)$. The substitution $\tau = \{1/x, f(1)/y\}$ unifies these terms forming $g(f(1))$. More generally, $\sigma = \{f(x)/y\}$ unifies these terms, with $\theta = \{1/x\}$, as follows: $g(f(x))|_\tau = g(f(x))|_{\sigma\theta} = g(f(1)) = g(y)|_{\sigma\theta} = g(y)|_\tau$.

Given two expressions e_1 and e_2 , the most general unification (MGU) algorithm returns the most general substitution σ *s.t.* $e_1|_\sigma = e_2|_\sigma$. The MGU algorithm pattern matches as follows:

- if e_1 and e_2 agree everywhere, return σ
else if neither e_1 nor e_2 are variables, FAIL
- if e_1 is a variable, substitute e_2 for e_1 , and recur
else if e_2 is a variable, substitute e_1 for e_2 , and recur

MGU(e_1, e_2, σ)	
Inputs	expressions e_1 and e_2 , substitution σ
Output	most general unifier σ
<ol style="list-style-type: none"> 1. $(u, v) = \text{DISAGREE}(e_1, e_2)$ if (u, v) is empty, return σ 2. if u is a variable <ol style="list-style-type: none"> (a) if $\text{OCCURS}(u, v)$, FAIL (b) else $\text{MGU}(e'_1, e'_2, \sigma')$, where $\sigma' = \sigma \cup \{v/u\}$, $e'_1 = e_1 _{\sigma'}$, and $e'_2 = e_2 _{\sigma'}$ 3. else if v is a variable <ol style="list-style-type: none"> (a) if $\text{OCCURS}(v, u)$, FAIL (b) else $\text{MGU}(e'_1, e'_2, \sigma')$, where $\sigma' = \sigma \cup \{u/v\}$, $e'_1 = e_1 _{\sigma'}$, and $e'_2 = e_2 _{\sigma'}$ 4. else if neither u nor v are variables, FAIL 	

Table 1: Unification Algorithm.

DISAGREE(e_1, e_2)	
Inputs	expressions e_1 and e_2
Output	first pair of subexpressions of e_1 and e_2 that disagree
<ol style="list-style-type: none"> 1. if either of e_1 or e_2 is a variable or symbol <ol style="list-style-type: none"> (a) if $e_1 \neq e_2$, return (e_1, e_2) (b) else return $()$ 2. else recursively check if subexpressions of e_1 and e_2 disagree: return $\text{DISAGREE}(\text{FIRST}(e_1), \text{FIRST}(e_2))$ or $\text{DISAGREE}(\text{REST}(e_1), \text{REST}(e_2))$ 	

Table 2: Disagreement Subroutine.

Unification relies on a subroutine that finds the first disagreement between two expressions by recursively comparing the expressions' constituents (see Table 2). Sound implementations of unification also make use of a further subroutine that checks whether one expression occurs within another (see Table 3).

Exercise: Compute the most general unifier or state why no unifier exists, for the following pairs of expressions:

- $R(f(x, x), a)$ and $R(f(y, f(y, a)), a)$

OCCURS(x, t)	
Inputs	variable x and term t
Output	if x occurs in t , then return true if x does not occur in t , then return false
<ol style="list-style-type: none"> 1. if t is a symbol, return false 2. if t is a variable, return $x = t$ 3. else recursively check whether x is a subterm of t: return OCCURS($x, \text{FIRST}(t)$) or OCCURS($x, \text{REST}(t)$) 	

Table 3: Occurs Check Subroutine.

Solution: No unifier exists, since occurs check fails on $y \in f(y, a)$.

- $f(f(y, x), x)$ and $f(f(v, f(b, v)), f(u, a))$

Solution: The most general unifier $\sigma = \{f(b, a)/x, a/y, a/v, b/u\}$, since $f(f(y, x), x)|_\sigma = f(f(a, f(b, a)), f(b, a)) = f(f(v, f(b, v)), f(u, a))|_\sigma$.

4 Sample Proof-by-Refutation

Example: Today is a day, and further, it is one on which we have class. If we have class on a day, then that day is either Tuesday or Thursday. If it is Tuesday, then the students can relax (students work hard on weekends). If the students are relaxing, then their professor can relax. The professor can also relax if it is Thursday, since s/he does not teach again until the following Tuesday. Noah is a student. Amy is a professor. (In fact, Amy is Noah's professor.) Can anyone relax today?

To express this scenario in the language of first-order logic, we introduce the following alphabet: $\mathcal{A} = \{\text{TODAY, YESTERDAY, TOMORROW, } \dots, \text{SUNDAY, MONDAY, TUESDAY, } \dots, \text{NOAH, AMY, } \dots, \text{PROF_OF}(\cdot), C, D, T, H, S, P, R\}$.

The days in our alphabet are intended to represent themselves. Similarly, the names are intended to represent the people being named. The function PROF_OF(\cdot) takes as input a student and returns that student's professor. And the predicates included in our alphabet can be understood as follows:

- The first four predicates represent relations that relate objects like today, tomorrow, and yesterday, as well as days of the week.
 1. $C(d)$ is a unary predicate representing the relation "we have class on day d ".
 2. $D(d)$ is a unary predicate representing the relation " d is a day of the week".
 3. $T(d)$ is a unary predicate representing the relation " d is a Tuesday".
 4. $H(d)$ is a unary predicate representing the relation " d is a Thursday".
- The next two predicates represent relations that relate people:
 1. $S(x)$ is a unary predicate representing the relation " x is a student".

2. $P(x)$ is a unary predicate representing the relation “ x is a professor”.

- The last predicate, $R(x, d)$, relates days and people, and indicates whether or not “person x can relax on day d ”.

Using this alphabet, the following formulas express our knowledge base in first-order logic (in normal form). The final sentence represents the negation of the query, “Can anyone relax today?”.

1. $\top \rightarrow D(\text{TODAY})$
2. $\top \rightarrow C(\text{TODAY})$
3. $D(d_1) \wedge C(d_1) \rightarrow T(d_1) \vee H(d_1)$
4. $T(d_2) \wedge S(x_2) \rightarrow R(x_2, d_2)$
5. $D(d_3) \wedge S(x_3) \wedge R(x_3, d_3) \rightarrow R(\text{PROF_OF}(x_3), d_3)$
6. $H(x_4) \wedge P(d_4) \rightarrow R(x_4, d_4)$
7. $\top \rightarrow P(\text{AMY})$
8. $\top \rightarrow S(\text{NOAH})$
9. $R(x, d) \rightarrow \perp$

The following proof-by-refutation shows someone (namely, Noah’s professor) can relax today.

$\frac{D(d_1) \wedge C(d_1) \rightarrow T(d_1) \vee H(d_1) \quad \top \rightarrow D(\text{TODAY})}{C(\text{TODAY}) \rightarrow T(\text{TODAY}) \vee H(\text{TODAY})} \quad \{\text{TODAY}/d_1\} \quad \top \rightarrow C(\text{TODAY})$
$\top \rightarrow T(\text{TODAY}) \vee H(\text{TODAY})$

$\frac{\vdots}{\top \rightarrow T(\text{TODAY}) \vee H(\text{TODAY})} \quad \frac{T(d_2) \wedge S(x_2) \rightarrow R(x_2, d_2)}{S(x_2) \rightarrow R(x_2, \text{TODAY}) \vee H(\text{TODAY})} \quad \{\text{TODAY}/d_2\} \quad \frac{\top \rightarrow D(\text{TODAY}) \quad D(x_3) \wedge S(x_3) \wedge R(x_3, d_3) \rightarrow R(\text{PROF_OF}(x_3), d_3)}{S(x_3) \wedge R(x_3, d_3) \rightarrow R(\text{PROF_OF}(x_3), \text{TODAY})}$
$S(x_2) \rightarrow R(\text{PROF_OF}(x_2), \text{TODAY}) \vee H(\text{TODAY})$

$\frac{\vdots}{S(x_2) \rightarrow R(\text{PROF_OF}(x_2), \text{TODAY}) \vee H(\text{TODAY})} \quad \top \rightarrow S(\text{NOAH}) \quad \{\text{NOAH}/x_2\} \quad R(x, d) \rightarrow \perp$
$\top \rightarrow R(\text{PROF_OF}(\text{NOAH}), \text{TODAY}) \vee H(\text{TODAY}) \quad \top \rightarrow H(\text{TODAY})$

$$\begin{array}{c}
 \vdots \\
 \hline
 \top \rightarrow H(\text{TODAY}) \quad H(x_4) \wedge P(d_4) \rightarrow R(x_4, d_4) \quad \{\text{TODAY}/d_4\} \quad \top \rightarrow P(\text{PROF_OF}(\text{NOAH})) \\
 \hline
 P(x_4) \rightarrow R(x_4, \text{TODAY}) \quad \top \rightarrow R(\text{PROF_OF}(\text{NOAH}), \text{TODAY}) \quad \{\text{PROF_OF}(\text{NOAH})/x_4\} \quad R(x, d) \rightarrow \perp \\
 \hline
 \top \rightarrow \perp
 \end{array}$$