知识表示与推理 (KRR)

- 知识表示与推理概述
- 一阶逻辑 (谓词逻辑): 文法和语义
- 基于归结的推理过程

*Slides based on those of Hector Levesque and Sheila McIlraith

What is KRR?

智能体所相信的命题的符号编码及其操作以生成智能体所相信的但未显示表示的命题的表示

An example

- 显示表示的信念：
  $GradStu(Ann)$, $GradStu(Bob)$, $\forall x(GradStu(x) \rightarrow Student(x))$
- 隐式表示的信念：
  $Student(Ann)$, $Student(Bob)$,
  $\forall x(\neg Student(x) \rightarrow \neg GradStu(x))$

Why KRR?

我们需要知识来完成很多任务，如回答问题

Could a crocodile run a steeplechase? [Levesque 88]

- Yes
- No

- steeplechase: 障碍赛跑
- 思考过程：鳄鱼腿短，障碍高，因而答案是 No

Yet another example

考虑一个关于材料的问题

> The large ball crashed right through the table because it was made of XYZZY. What was made of XYZZY?
> - the large ball
> - the table

假设你了解了关于 XYZZY 的一些事实：

1. 它是 Dow 化学公司的产品。
2. 它通常是白色的，但也有绿色和蓝色的。
3. 它 98% 是空气，因此重量轻，浮力大
4. 它最初是由一位瑞典发明家发现的。

在什么时候答案不再只是猜测?

Why KRR?

- KR 假设: 任何人工智能系统都是基于知识的
    - 人工智能在很大程度上涉及构建基于知识的系统
    - 有些，在一定程度上，例如，游戏，视觉等。
    - 有些，在更小的程度上，如语音，运动控制等。
- 基于知识的系统: 具有有以下性质的结构的系统
    - 可以被解释为命题
    - 决定系统的行为

  这样的结构被称为知识库 (knowledge base, KB)

Two examples

Example 1

```
printColour(snow) :- !, write("It's white.").
printColour(grass) :- !, write("It's green.").
printColour(sky) :- !, write("It's yellow.").
printColour(X) :- write("Beats me.").
```

Example 2

```
printColour(X) :- colour(X,Y), !,
        write("It's "), write(Y), write(".").
printColour(X) :- write("Beats me.").

colour(snow,white).
colour(sky,yellow).
colour(X,Y) :- madeof(X,Z), colour(Z,Y).
madeof(grass,vegetation).
colour(vegetation,green).
```

基于知识的系统的优势

- 最适合开放式任务
- 可解释性和可扩展性
- 认知可渗透性，即动作依赖于信念，包括隐式表示的信念

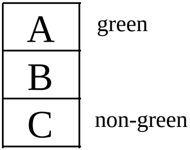
KRR and logic

Logic is the main tool for KRR, because logic studies

- How to formally represent agent's beliefs
- Given the explicitly represented beliefs, what are the implicitly represented beliefs

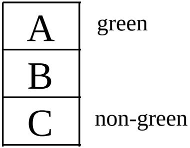
There are many kinds of logics. In this course, we will use first-order logic (FOL) as the tool for KRR

A blocks world example



- Given the scene, human can easily draw the conclusion "there is a green block directly on top of a non-green block"
- How can a machine do the same?

Formalization in FOL



- $S = \{On(a, b), On(b, c), Green(a), \neg Green(c)\}$
- $\alpha = \exists x \exists y [Green(x) \wedge \neg Green(y) \wedge On(x, y)]$
- S logically entails (逻辑蕴涵) α

An example

- Tony, Mike, and John belong to the Alpine Club.
- Every member of the Alpine Club who is not a skier is a mountain climber.
- Mountain climbers do not like rain, and anyone who does not like snow is not a skier.
- Mike dislikes whatever Tony likes, and likes whatever Tony dislikes.
- Tony likes rain and snow.
- Is there a member of the Alpine Club who is a mountain climber but not a skier?

An example (cont'd)

- Intelligence is needed to answer the question
- Can we make machines answer the question?
- A possible approach
    - First, translate the sentences and question into FOL formulas
        - This is semantic parsing, *i.e.*, automaticly translate natural language sentences into formal expressions
        - Despite substantial research, semantic parsing remains challenging
    - Second, check if the formula of the question is logically entailed by the formulas of the sentences
        - We will show that there are ways to automate this step

Alphabet

- Individuals (constants or 0-ary functions):
  - tony, mike, john
  - rain, snow
- Types (unary predicates):
  - $A(x)$ means that x belongs to Alpine Club
  - $S(x)$ means that x is a skier
  - $C(x)$ means that x is a mountain climber
- Relationships (binary predicates):
  - $L(x, y)$ means that x likes y

Basic facts

- Tony, Mike, and John belong to the Alpine Club.
  $A(tony), A(mike), A(john)$
- Tony likes rain and snow.
  $L(tony, rain), L(tony, snow)$

Complex facts

- Every member of the Alpine Club who is not a skier is a mountain climber.
  $\forall x(A(x) \wedge \neg S(x) \rightarrow C(x))$
- Mountain climbers do not like rain, and anyone who does not like snow is not a skier.
  $\forall x(C(x) \rightarrow \neg L(x, rain))$
  $\forall x(\neg L(x, snow) \rightarrow \neg S(x))$
- Mike dislikes whatever Tony likes, and likes whatever Tony dislikes.
  $\forall x(L(tony, x) \rightarrow \neg L(mike, x))$
  $\forall x(\neg L(tony, x) \rightarrow L(mike, x))$
- Is there a member of the Alpine Club who is a mountain climber but not a skier?
  $\exists x(A(x) \wedge C(x) \wedge \neg S(x))$

Outline

- 知识表示与推理概述
- 一阶逻辑 (谓词逻辑): 文法和语义
- 基于归结的推理过程

字母表 (Alphabet)

逻辑符号 (固定的含义和用法):

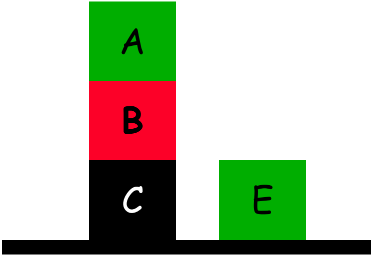
- 标点符号: (,),,,.
- 连接词 (Connectives) 和量词 (quantifiers): $=, \neg, \wedge, \vee, \forall, \exists$
- 变量 (Variables): $x, x_1, x_2, \ldots, x', x'', \ldots, y, \ldots, z, \ldots$

非逻辑符号 (含义和用法依赖于论域):

- 谓词 (Predicate) 符号
  - 维数 (arity): 参数的个数
  - 0 元谓词: 命题符号
- 函数 (Function) 符号
  - 0 元函数: 常量符号

A blocks world example

Environment



Language (Syntax)

- Constants: a,b,c,e
- Functions:
  - No function
- Predicates:
  - on: binary
  - above: binary
  - clear: unary
  - ontable: unary

Terms (项)

- Every variable is a term
- If $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term

Formulas

- If $t_1, \ldots, t_n$ are terms and P is a predicate symbol of arity n, then $P(t_1, \ldots, t_n)$ is an atomic formula
- If t_1 and t_2 are terms, then $(t_1 = t_2)$ is an atomic formula
- If α and β are formulas, and v is a variable, then $\neg\alpha, (\alpha \land \beta), (\alpha \lor \beta), \exists v.\alpha, \forall v.\alpha$ are formulas

Notation

- Occasionally add or omit (,)
- Use [,] and $\{,\}$
- Abbreviation: $(\alpha \rightarrow \beta)$ for $(\neg\alpha \vee \beta)$
- Abbreviation: $(\alpha \leftrightarrow \beta)$ for $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
- Predicates: mixed case capitalized, *e.g.*, Person, OlderThan
- Functions (and constants): mixed case uncapitalized, *e.g.*, john, father,

Variable scope

- Free and bound occurrences of variables: An occurrence of variable x in A is bound if it is in a subformula of A of the form $\forall x B$ or $\exists x B$. Otherwise the occurrence is free.
- *e.g.*, $P(x) \wedge \exists x[P(x) \vee Q(x)]$
- A sentence: formula with no free variables
- Substitution (代入): $\alpha[v/t]$ means α with all free occurrences of the v replaced by term t
- In general, $\alpha[v_1/t_1, \ldots, v_n/t_n]$

Interpretations

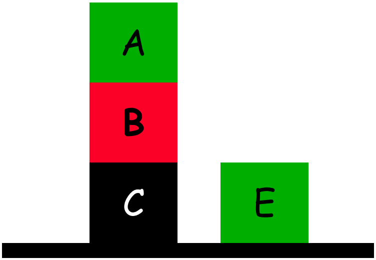
An interpretation (解释) is a pair $M = \langle D, I \rangle$

- D is the domain, can be any non-empty set
- I is a mapping from the set of predicate and function symbols
- If P is a predicate symbol of arity n, $I(P)$ is an n-ary relation over D, *i.e.*, $I(P) \subseteq D^n$
    - If p is a 0-ary predicate symbol, *i.e.*, a propositional symbol, $I(p) \in \{true, false\}$
- If f is a function symbol of arity n, $I(f)$ is an n-ary function over D, *i.e.*, $I(f) : D^n \to D$
    - If c is a 0-ary function symbol, *i.e.*, a constant symbol, $I(c) \in D$

Blocks world example

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}$, $\Phi(b) = \underline{B}$, $\Phi(c) = \underline{C}$, $\Phi(e) = \underline{E}$.
- $\Psi(on) = \{(\underline{A},\underline{B}),(\underline{B},\underline{C})\}$
- $\Psi(above) = \{(\underline{A},\underline{B}),(\underline{B},\underline{C}),(\underline{A},\underline{C})\}$
- $\Psi(clear)=\{\underline{A},\underline{E}\}$
- $\Psi(ontable)=\{\underline{C},\underline{E}\}$

Environment



Note: Φ and Ψ are *I*

Denotation (指称) of terms

- Terms denote elements of the domain
- A variable assignment μ is a mapping from the set of variables to the domain D
- $\|v\|_{M,\mu} = \mu(v)$
- $\|f(t_1, \ldots, t_n)\|_{M,\mu} = I(f)(\|t_1\|_{M,\mu}, \ldots, \|t_n\|_{M,\mu})$

Satisfaction: atomic formulas

$M, \mu \models \alpha$ is read "M, μ satisfies (满足) α"

- $M, \mu \models P(t_1, \ldots, t_n)$ iff $\langle \|t_1\|_{M,\mu}, \ldots, \|t_n\|_{M,\mu} \rangle \in I(P)$
- $M, \mu \models (t_1 = t_2)$ iff $\|t_1\|_{M,\mu} = \|t_2\|_{M,\mu}$

Satisfaction: propositional connectives

- $M, \mu \models \neg\alpha$ iff $M, \mu \not\models \alpha$
- $M, \mu \models (\alpha \wedge \beta)$ iff $M, \mu \models \alpha$ and $M, \mu \models \beta$
- $M, \mu \models (\alpha \vee \beta)$ iff $M, \mu \models \alpha$ or $M, \mu \models \beta$

Satisfaction: quantifiers

$\mu[v \mapsto d]$ denotes a variable assignment just like μ, except that it maps v to d

- $M, \mu \models \exists v.\alpha$ iff for some $d \in D$, $M, \mu[v \mapsto d] \models \alpha$
- $M, \mu \models \forall v.\alpha$ iff for all $d \in D$, $M, \mu[v \mapsto d] \models \alpha$

Let α be a sentence. Then whether $M, \mu \models \alpha$ is independent of μ. Thus we simply write $M \models \alpha$

Blocks world example

- D = {<u>A</u>, <u>B</u>, <u>C</u>, <u>E</u>}
- Φ(a) = <u>A</u>, Φ(b) = <u>B</u>, Φ(c) = <u>C</u>, Φ(e) = <u>E</u>.
- Ψ(on) = {(<u>A</u>,<u>B</u>),(<u>B</u>,<u>C</u>)}
- Ψ(above) = {(<u>A</u>,<u>B</u>),(<u>B</u>,<u>C</u>),(<u>A</u>,<u>C</u>)}
- Ψ(clear)={<u>A</u>,<u>E</u>}
- Ψ(ontable)={<u>C</u>,<u>E</u>}

∀X,Y. on(X,Y)→above(X,Y)

✓X=<u>A</u>, Y=<u>B</u>

✓X=<u>C</u>, Y=<u>A</u>

✓ ...

∀X,Y. above(X,Y)→on(X,Y)

✓X=<u>A</u>, Y=<u>B</u>

× X=<u>A</u>, Y=<u>C</u>

Blocks world example

- $D = \{\underline{A}, \underline{B}, \underline{C}, \underline{E}\}$
- $\Phi(a) = \underline{A}$, $\Phi(b) = \underline{B}$, $\Phi(c) = \underline{C}$, $\Phi(e) = \underline{E}$.
- $\Psi(on) = \{(\underline{A},\underline{B}),(\underline{B},\underline{C})\}$
- $\Psi(above) = \{(\underline{A},\underline{B}),(\underline{B},\underline{C}),(\underline{A},\underline{C})\}$
- $\Psi(clear)=\{\underline{A},\underline{E}\}$
- $\Psi(ontable)=\{\underline{C},\underline{E}\}$

$\forall X \exists Y. (clear(X) \vee on(Y,X))$

- ✓ $X=\underline{A}$
- ✓ $X=\underline{C}$, $Y=\underline{B}$
- ✓ ...

$\exists Y \forall X.(clear(X) \vee on(Y,X))$

- × $Y=\underline{A}$? No! $(X=\underline{C})$
- × $Y=\underline{C}$? No! $(X=\underline{B})$
- × $Y=\underline{E}$? No! $(X=\underline{B})$
- × $Y=\underline{B}$? No! $(X=\underline{B})$

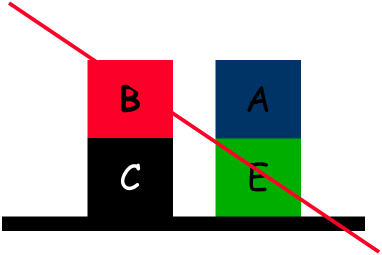
Satisfiability (可满足性)

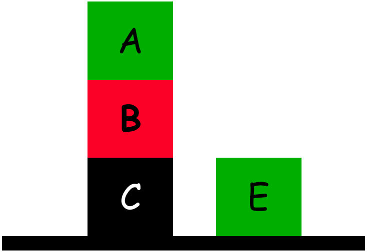
- Let S be a set of sentences
- $M \models S$, read M satisfies S, if for every $\alpha \in M$, $M \models \alpha$
- If $M \models S$, we say M is a model of S
- We say that S is satisfiable if there is M s.t. $M \models S$, and
- *e.g.*, is $\{\forall x(P(x) \to Q(x)), P(a), \neg Q(a)\}$ satisfiable?

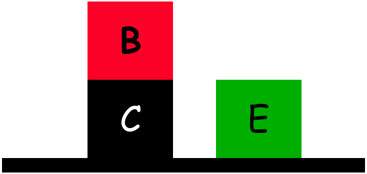
Blocks world example

KB

1. on(b,c)
2. clear(e)



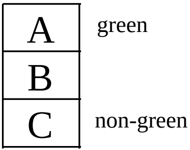




Logical entailment (逻辑蕴涵)

- $S \models \alpha$ iff for every M, if $M \models S$ then $M \models \alpha$
- $S \models \alpha$ is read: S entails α or α is a logical consequence (逻辑推论) of S
- A special case: $\emptyset \models \alpha$, simply written $\models \alpha$, read "α is valid" (有效的)
- Note that $\{\alpha_1, \ldots, \alpha_n\} \models \alpha$ iff $\alpha_1 \land \ldots \land \alpha_n \to \alpha$ is valid iff $\alpha_1 \land \ldots \land \alpha_n \land \neg\alpha$ is unsatisfiable
- Alpine Club example
  - Let KB be the set of sentences, and α be the question
  - We want to know if $KB \models \alpha$?

Blocks world example cont'd



- $S = \{On(a, b), On(b, c), Green(a), \neg Green(c)\}$
- $\alpha = \exists x \exists y [Green(x) \wedge \neg Green(y) \wedge On(x, y)]$
- We prove that $S \models \alpha$

Logical entailment: examples

- $\forall x A \vee \forall x B \models \forall x(A \vee B)$
- Does $\forall x(A \vee B) \models \forall x A \vee \forall x B$
- $\exists x(A \wedge B) \models \exists x A \wedge \exists x B$
- Does $\exists x A \wedge \exists x B \models \exists x(A \wedge B)$?
- $\exists y \forall x A \models \forall x \exists y A$
- Does $\forall x \exists y A \models \exists y \forall x A$?

The only way to prove that $KB \not\models \alpha$ is to give an interpretation satisfying KB but not α.

Alpine Club example cont'd

- Suppose that we had been told that Mike likes whatever Tony dislikes, but we had not been told that Mike dislikes whatever Tony likes.
- Can we still claim that there is a member of the Alpine Club who is a mountain climber but not a skier?
- No. We give an interpretation which satisfies the modified KB but not f as follows: Let $D = \{T, M, J, R, S\}$. Let $I(tony) = T, I(mike) = M, I(john) = J, I(rain) = R, I(snow) = S$. Let $I(A) = \{T, M, J\}, I(S) = \{T, M, J\}, I(C) = \emptyset$, $I(L) = \{(T, R), (T, S), (T, T), (M, M), (M, S), (M, J), (J, S)\}$.

Logical entailment and knowledge-based systems

- Start with KB representing explicit beliefs, usually what the agent has been told or has learned
- Implicit beliefs: $\{\alpha \mid KB \models \alpha\}$
- Actions depend on implicit beliefs, rather than explicit beliefs

Inference (推理) procedure

- We want a mechanical procedure to check if $KB \models \alpha$
- Called an inference procedure
- Sound (合理的) if whenever it says yes, then $KB \models \alpha$
- Complete (完备的) if whenever $KB \models \alpha$, then it says yes

Inference rules commonly used by human

- 假言推理: $\phi \rightarrow \psi, \phi \models \psi$
- 假言三段论: $\phi \rightarrow \psi, \psi \rightarrow \eta \models \phi \rightarrow \eta$
- 析取三段论: $\phi \vee \psi, \neg\phi \models \psi$
- 全称实例化: $\forall x P(x) \models P(c)$
- 全称一般化: $P(c) \models \exists x P(x)$
- ...

Resolution-based Inference procedure

- Resolution (归结) is a rule of inference
- Resolution-based inference procedure: refutation (反演)
- We begin with the propositional case
- Then proceed to the first-order case

Clausal form

- A literal (文字) is an atomic formula or its negation, *e.g.*, $p, \neg p$
- A clause (子句) is a disjunction (析取) of literals, written as the set of literals
  - *e.g.*, $p \vee \neg r \vee s$, written $(p, \neg r, s)$
- A special case: empty clause $()$, representing false
- A formula is a conjunction (合取) of clauses, written as the set of clauses

Resolution rule of inference

- From the two clauses $\{p\} \cup c_1$ and $\{\neg p\} \cup c_2$, infer the clause $c_1 \cup c_2$
- $c_1 \cup c_2$ is called the resolvent of input clauses wrt the atom p
- *e.g.*, (p) and $(\neg p)$ resolve to $()$,
  (w, r, q) and $(w, s, \neg r)$ resolve to (w, q, s) wrt r
- **Proposition.** $\{p\} \cup c_1, \{\neg p\} \cup c_2 \models c_1 \cup c_2$
  Proof:

Derivation (推导)

A derivation of a clause c from a set S of clauses is a sequence $c_1, c_2, \ldots, c_n$ of clauses, where $c_n = c$, and for each c_i, either

- $c_i \in S$, or
- c_i is a resolvent of two earlier clauses in the derivation

We write $S \vdash c$ if there is a derivation of c from S

Soundness of derivations

- **Theorem.** If $S \vdash c$, then $S \models c$
  Proof:
  - Let $c_1, c_2, \ldots, c_n$ be a derivation of c from S
  - We prove by induction on i that for all $1 \leq i \leq n$, $S \models c_i$.
- However, the converse does not hold in general
  e.g., $(p) \models (p, q)$, but $(p) \not\vdash (p, q)$

Soundness and completeness of refutations

Theorem. $S \vdash ()$ iff $S \models ()$ iff S is unsatisfiable

We will not prove the completeness part

Resolution-based inference procedure: refutation

$KB \models \alpha$ iff $KB \land \neg\alpha$ is unsatisfiable

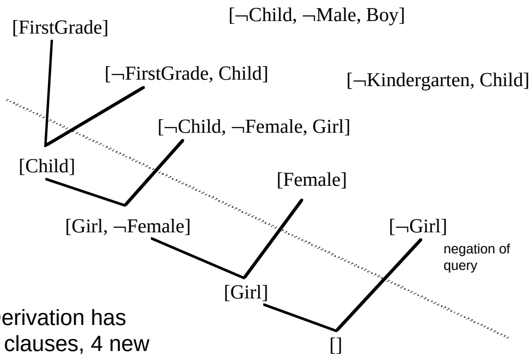
Thus to check if $KB \models \alpha$,

- put KB and $\neg\alpha$ into clausal form to get S,
- check if $S \vdash ()$

KB

FirstGrade
FirstGrade ⊃ Child
Child ∧ Male ⊃ Boy
Kindergarten ⊃ Child
Child ∧ Female ⊃ Girl
Female

Show that KB |= Girl



Derivation has
9 clauses, 4 new

Note ⊃ is another representation of →

Refutation example 2

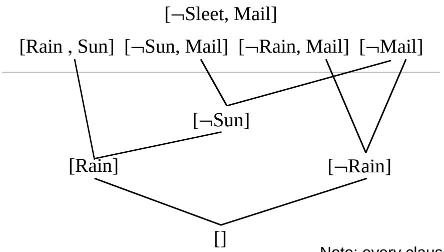
KB

(Rain $\vee$ Sun)

(Sun $\supset$ Mail)

((Rain $\vee$ Sleet) $\supset$ Mail)

Show KB $\models$ Mail



Note: every clause not in S has 2 parents

Similarly KB $\not\models$ Rain

Can enumerate all resolvents given ¬Rain, and [] will not be generated

The first-order case

We need

- A way of converting KB and f (the query) into clausal form
- A way of doing resolution even when we have variables.
  This needs unification (合一)

Conversion to Clausal Form: 8 steps

1. Eliminate Implications.
2. Move Negations inwards (and simplify $\neg\neg$).
3. Standardize Variables.
4. Skolemize.
5. Convert to Prenex (前缀) Form.
6. Distribute disjunctions over conjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.

Skolemization

Consider $\exists y. Elephant(y) \wedge Friendly(y)$

- This asserts that there is some individual that is both an elephant and friendly.
- To remove the existential, we invent a name for this individual, say a. This is a new constant symbol not equal to any previous constant symbols:
  $Elephant(a) \wedge Friendly(a)$
- This is saying the same thing, since we do not know anything about the new constant a.
- It is essential that the introduced symbol a is new. Else we might say more than the existential formula.

Skolemization

Now consider $\forall x \exists y. Loves(x, y)$.

- This formula claims that for every x there is some y that x loves (perhaps a different y for each x).
- Replacing the existential by a new constant won't work: $\forall x. Loves(x, a)$, because this asserts that there is a particular individual a loved by every x.
- To properly convert existential quantifiers scoped by universal quantifiers we must use functions not just constants.
- In this case x scopes y, so we must replace y by a function of x: $\forall x. Loves(x, g(x))$, where g is a new function symbol.
- This formula asserts that for every x there is some individual (given by $g(x)$) that x loves. $g(x)$ can be different for each x.

Skolemization examples

- $\forall x, y, z \exists w. R(x, y, z, w) \implies \forall x, y, z. R(x, y, z, h_1(x, y, z))$
- $\forall x, y \exists w. R(x, y, g(w)) \implies \forall x, y. R(x, y, g(h_2(x, y)))$
- $\forall x, y \exists w \forall z. R(x, y, w) \wedge Q(z, w) \implies$
  $\forall x, y, z. R(x, y, h_3(x, y)) \wedge Q(z, h_3(x, y))$

A conversion example

$$\forall x\{P(x) \to [\forall y(P(y) \to P(f(x,y))) \wedge \neg\forall y(\neg Q(x,y) \wedge P(y))]\}$$

1. Eliminate implications using $A \to B \Leftrightarrow \neg A \vee B$

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x,y))) \wedge \neg\forall y(\neg Q(x,y) \wedge P(y))]\}$$

Move negations inwards

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x,y))) \wedge \neg\forall y(\neg Q(x,y) \wedge P(y))]\}$$

2. Move negations inwards using
   - $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$, $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
   - $\neg\exists x.A \Leftrightarrow \forall x.\neg A$, $\neg\forall x.A \Leftrightarrow \exists x.\neg A$, $\neg\neg A \Leftrightarrow A$

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x,y))) \wedge \exists y(Q(x,y) \vee \neg P(y))]\}$$

Standardize Variables

$$\forall x\{\neg P(x) \lor [\forall y(\neg P(y) \lor P(f(x,y))) \land \exists y(Q(x,y) \lor \neg P(y))]\}$$

3. Standardize Variables (Rename variables so that each quantified variable is unique)

$$\forall x\{\neg P(x) \lor [\forall y(\neg P(y) \lor P(f(x,y))) \land \exists z(Q(x,z) \lor \neg P(z))]\}$$

Skolemize

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x, y))) \wedge \exists z(Q(x, z) \vee \neg P(z))]\}$$

4. Skolemize (Remove existential quantifiers by introducing new function symbols)

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x, y))) \wedge (Q(x, g(x)) \vee \neg P(g(x)))]\}$$

Convert to prenex form

$$\forall x\{\neg P(x) \vee [\forall y(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \vee \neg P(g(x)))]\}$$

5. Convert to prenex form. (Bring all quantifiers to the front – only universals, each with different name)

$$\forall x \forall y\{\neg P(x) \vee [(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \vee \neg P(g(x)))]\}$$

注意：教材中先求前束范式再 Skolem 化，这样引入的函数可能更复杂，比如这里是 $g(x,y)$

Disjunctions over conjunctions

$$\forall x \forall y \{\neg P(x) \vee [(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \vee \neg P(g(x)))]\}$$

6. Disjunctions over conjunctions using
$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

$$\begin{aligned} \forall x \forall y \{ & (\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge \\ & (\neg P(x) \vee Q(x,g(x)) \vee \neg P(g(x)))\} \end{aligned}$$

Convert to Clauses

$$\forall x \forall y \{(\neg P(x) \vee \neg P(y) \vee P(f(x, y))) \wedge \\ (\neg P(x) \vee Q(x, g(x)) \vee \neg P(g(x)))\}$$

8. Convert to Clauses (remove quantifiers and break apart conjunctions).

a) $\neg P(x) \vee \neg P(y) \vee P(f(x, y))$
b) $\neg P(x) \vee Q(x, g(x)) \vee \neg P(g(x))$

Unification

- Can the clauses $(P(john), Q(fred), R(x))$ and $(\neg P(y), R(susan), R(y))$ be resolved?
- Once reduced to clausal form, all remaining variables are universally quantified.
  $(P(john), Q(fred), R(john))$, $(P(john), Q(fred), R(fred))$, $\ldots$
- So there is a specialization of $(P(john), Q(fred), R(x))$ that can be resolved with a specialization of $(\neg P(y), R(susan), R(y))$
- In particular, $(P(john), Q(fred), R(john))$ can be resolved with $(\neg P(john), R(susan), R(john))$, producing $(Q(fred), R(john), R(susan))$

Unification

- We want to be able to match conflicting literals, even when they have variables.
- This matching process automatically determines whether or not there is a specialization that matches.
- But, we don't want to over specialize!

Unification

- Consider $(\neg P(x), S(x), Q(fred))$ and $(P(y), R(y))$
- We need to unify $P(x)$ and $P(y)$. How do we do this?
- Possible resolvants:
  - $(S(john), Q(fred), R(john))\{x = john, y = john\}$
  - $(S(sally), Q(fred), R(sally))\{x = sally, y = sally\}$
  - $(S(x), Q(fred), R(x))\{y = x\}$
- The last resolvant is most-general, the other two are specializations. We want the most general clause for use in future resolution steps.
- To define the most-general unifier, we need to define substitutions.

Substitution (置换)

- A key component of unification is substitution.
- A substitution is a finite set of equations of the form $V = t$ where V is a variable and t is a term not containing V. (t might contain other variables).
- We can apply a substitution $\sigma = \{V_1 = t_1, \ldots, V_n = t_n\}$ to a formula f to obtain a new formula $f\sigma$ by simultaneously replacing every variable V_i by term t_i.
- *e.g.*, $P(x, g(y, z))\{x = y, y = f(a)\} \Longrightarrow P(y, g(f(a), z))$
- Note that the substitutions are not applied sequentially, *i.e.*, the first y is not subsequently replaced by $f(a)$.

Composition of substitutions (置换的复合)

- We can compose two substitutions θ and σ to obtain a new substitution $\theta\sigma$
- Let $\theta = \{x_1 = s_1, x_2 = s_2, \ldots, x_m = s_m\}$, $\sigma = \{y_1 = t_1, y_2 = t_2, \ldots, y_k = t_k\}$
- Step 1. Get $S = \{x_1 = s_1\sigma, x_2 = s_2\sigma, \ldots, x_m = s_m\sigma, y_1 = t_1, y_2 = t_2, \ldots, y_k = t_k\}$
- Step 2. Delete any identities, *i.e.*, equations of the form $V = V$.
- Step 3. Delete any equation $y_i = s_i$ where y_i is equal to one of the x_j in θ. Because $y_i = s_i$ is overridden

Composition example

- Let $\theta = \{x = f(y), y = z\}$, $\sigma = \{x = a, y = b, z = y\}$
- Step 1. Get $S = \{x = f(b), y = y, x = a, y = b, z = y\}$
- Step 2. Delete $y = y$.
- Step 3. Delete $x = a$.
- The result is $S = \{x = f(b), y = b, z = y\}$

Note on substitutions

- The empty substitution $\epsilon = \{\}$ is also a substitution, and we have $\theta\epsilon = \theta$.
- More importantly, substitutions when applied to formulas are associative (结合的): $(f\theta)\sigma = f(\theta\sigma)$
- Composition is simply a way of converting the sequential application of a series of substitutions to a single substitution.

Unifiers

- A unifier (合一项) of two formulas f and g is a substitution σ that makes f and g syntactically identical.
- Note that not all formulas can be unified – substitutions only affect variables.
- *e.g.*, $P(f(x), a)$ and $P(y, f(w))$ cannot be unified, as there is no way of making $a = f(w)$ with a substitution.

MGU

A substitution σ of two formulas f and g is a Most General Unifier (MGU) if

- σ is a unifier.
- For every other unifier θ of f and g there must exist a third substitution λ such that $\theta = \sigma\lambda$.

This says that every other unifier is "more specialized" than σ.

The MGU of a pair of formulas f and g is unique up to renaming.

MGU example

- $P(f(x), z)$ and $P(y, a)$
- $\sigma = \{y = f(a), x = a, z = a\}$ is a unifier, but not an MGU
- $\theta = \{y = f(x), z = a\}$ is an MGU
- $\sigma = \theta\lambda$, where $\lambda = \{x = a\}$

Computing MGUs

- The MGU is the "least specialized" way of making atomic formulas with variables match.
- We can compute MGUs.
- Intuitively we line up (对齐) the two formulas and find the first sub-expression where they disagree.
- The pair of subexpressions where they first disagree is called the disagreement set (差异集).
- The algorithm works by successively fixing disagreement sets (逐项修正差异集) until the two formulas become syntactically identical.

Computing MGUs

Given two atomic formulas f and g

1. $\sigma = \{\}$; $S = \{f, g\}$
2. If S contains an identical pair of formulas, stop and return σ as the MGU of f and g.
3. Else find the disagreement set $D = \{e_1, e_2\}$ of S
4. If $e_1 = V$ a variable, and $e_2 = t$ a term not containing V (or vice-versa) then let $\sigma = \sigma\{V = t\}$; $S = S\{V = t\}$; Goto 2
5. Else stop, f and g cannot be unified.

Note: to update σ, we must compose σ with $\{V = t\}$.
A common error is to just add $V = t$ to σ.

Computing MGU examples

1. $P(f(a), g(x))$ and $P(y, y)$: un-unifiable
2. $P(a, x, h(g(z)))$ and $P(z, h(y), h(y))$
   MGU: $\{z = a, x = h(g(a)), y = g(a)\}$
3. $P(x, x)$ and $P(y, f(y))$: un-unifiable

First-order Resolution

From the two clauses $\{\rho_1\} \cup c_1$ and $\{\neg\rho_2\} \cup c_2$, where there exists a MGU σ for ρ_1 and ρ_2, infer the clause $(c_1 \cup c_2)\sigma$

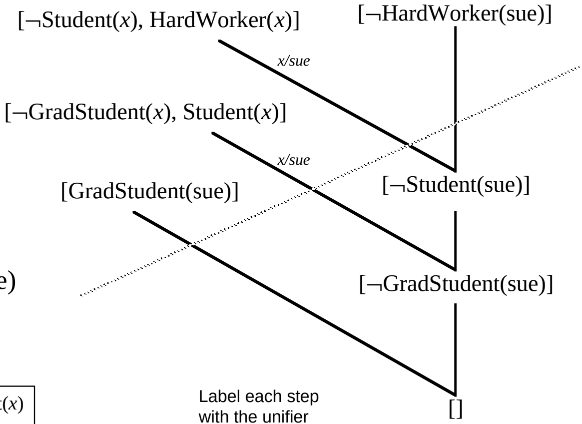
Theorem. $S \vdash ()$ iff S is unsatisfiable

A resolution example

1. $(P(x), Q(g(x)))$
2. $(R(a), Q(z), \neg P(a))$
3. R[1a,2c]{X=a} $(Q(g(a)), R(a), Q(z))$

- "R" means resolution step.
- "1a" means the 1st (a-th) literal in the first clause: $P(x)$.
- "2c" means the 3rd (c-th) literal in the second clause: $\neg P(a)$.
- 1a and 2c are the "clashing" literals (冲突文字).
- $\{X = a\}$ is the MGU applied.

Refutation example 1



?
KB |= HardWorker(sue)

KB

$\forall x$ GradStudent(x) $\supset$ Student(x)
$\forall x$ Student(x) $\supset$ HardWorker(x)
GradStudent(sue)

Label each step
with the unifier

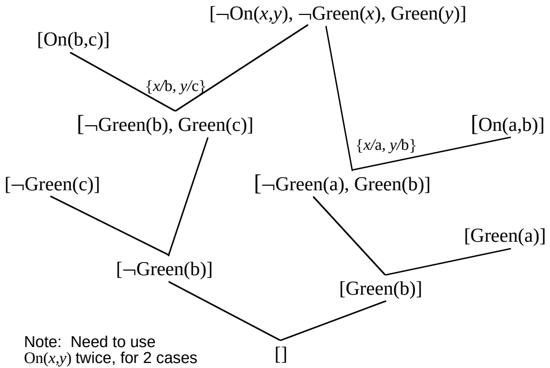
Point to relevant
literals in clauses

The 3 blocks example

KB = {On(a,b), On(b,c), Green(a), ¬Green(c)} already in CNF

Query = $\exists x \exists y[\mathrm{On}(x,y) \wedge \mathrm{Green}(x) \wedge \neg\mathrm{Green}(y)]$

Note: ¬Q has no existentials, so yields



Note: Need to use On(x,y) twice, for 2 cases

Alpine Club example

1. $A(tony)$
2. $A(mike)$
3. $A(john)$
4. $L(tony, rain)$
5. $L(tony, snow)$

$$
\begin{array}{lcl}
\forall x(A(x) \wedge \neg S(x)) \rightarrow C(x) & \Rightarrow & 6.\ (\neg A(x), S(x), C(x)) \\
\forall x(C(x) \rightarrow \neg L(x, rain)) & \Rightarrow & 7.\ (\neg C(y), \neg L(y, rain)) \\
\forall x(\neg L(x, snow) \rightarrow \neg S(x)) & \Rightarrow & 8.\ (L(z, snow), \neg S(z)) \\
\forall x(L(tony, x) \rightarrow \neg L(mike, x)) & \Rightarrow & 9.\ (\neg L(tony, u), \neg L(mike, u)) \\
\forall x(\neg L(tony, x) \rightarrow L(mike, x)) & \Rightarrow & 10.\ (L(tony, v), L(mike, v)) \\
\neg \exists x(A(x) \wedge C(x) \wedge \neg S(x)) & \Rightarrow & 11.\ (\neg A(w), \neg C(w), S(w))
\end{array}
$$

Alpine Club example refutation

12. R[5, 9a]u = snow $\neg L(mike, snow)$
13. R[8,12]z = mike $\neg S(mike)$
14. R[6b, 13]x = mike $(\neg A(mike), C(mike))$
15. R[2,14a] $C(mike)$
16. R[8a, 12]z = mike $\neg S(mike)$
17. R[2,11]w=mike $(\neg C(mike), S(mike))$
18. R[15, 17] $S(mike)$
19. R[16,18] ()

Refutation examples

Prove that $\exists y \forall x P(x, y) \models \forall x \exists y P(x, y)$

- $\exists y \forall x P(x, y) \Rightarrow 1. P(x, a)$
- $\neg \forall x \exists y P(x, y) \Leftrightarrow \exists x \forall y \neg P(x, y) \Rightarrow 2. \neg P(b, y)$
- R[1,2]$\{x = b, y = a\}()$

Exercises: Prove

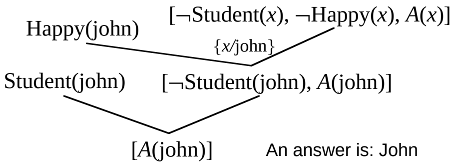
- $\forall x P(x) \vee \forall x Q(x) \models \forall x (P(x) \vee Q(x))$
- $\exists x (P(x) \wedge Q(x)) \models \exists x P(x) \wedge \exists x Q(x)$

Answer extraction (答案抽取)

- We can also answer wh- questions
- Replace query $\exists x P(x)$ by $\exists x[P(x) \wedge \neg answer(x)]$
- Negating it, we get $\forall x[\neg P(x) \vee answer(x)]$
- Instead of deriving (), derive any clause containing just the answer predicate

KB: Student(john)
Student(jane)
Happy(john)

Q: $\exists x[\text{Student}(x) \wedge \text{Happy}(x)]$



An answer is: John

Alpine Club example answer extraction

- 11. $(\neg A(w), \neg C(w), S(w), answer(w))$
- The same resolution steps as before give us $answer(mike)$

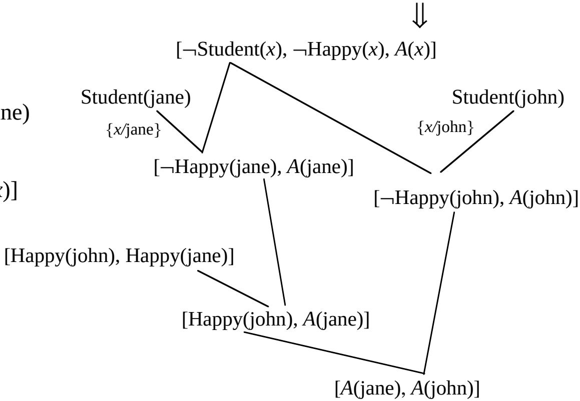
Disjunctive answers

KB:

Student(john)
Student(jane)
Happy(john) ∨ Happy(jane)

Query:

$\exists x[\text{Student}(x) \wedge \text{Happy}(x)]$



An answer is: either Jane or John

Note:

A problem

以下 4 面为扩展内容

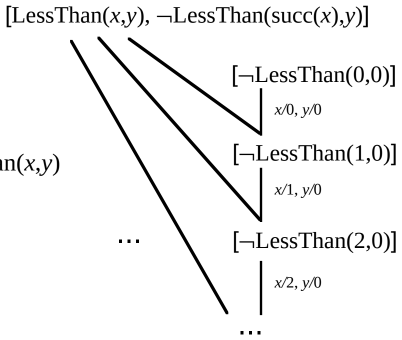
KB:

LessThan(succ(x),y) $\supset$ LessThan(x,y)

Query:

LessThan(zero,zero)

Should fail since KB $\not\models$ Q



Infinite branch of resolvents

We use 0 for zero, 1 for succ(zero), 2 for succ(succ(zero)), …

Undecidability in the first-order case

- There can be no procedure to decide if a set of clauses is satisfiable.
- **Theorem.** $S \vdash ()$ iff S is unsatisfiable
- However, there is no procedure to check if $S \vdash ()$, because
- When S is satisfiable, the search for $()$ may not terminate

Intractability in the propositional case

- Determining if a set of clauses is satisfiable was shown by Cook in 1972 to be NP-complete.
- Satisfiability is believed by most people to be unsolvable in polynomial time.
- Procedures have been proposed for determining satisfiability that appear to work much better in practice than Resolution.
- They are called SAT solvers as they are mostly used to find a satisfying interpretation for clauses that are satisfiable.

Implications for KRR

- In knowledge-based systems, actions depend on implicit beliefs, *i.e.*, logical entailments of KB
- However, as we have seen, computing entailments is unsolvable in general
- The hope is that in many practical scenarios, entailments can be efficiently computed
- In case entailments are difficult to compute, we seek for other ways out

Prolog and resolution

- Prolog is a language that is useful for doing symbolic and logic-based computation
- Resolutions forms the basis of the implementation of Prolog
- When searching for (), Prolog uses a specific depth-first left-right strategy

王浩：机器定理证明的奠基人

- 王浩（1921 - 1995），美籍华裔哲学家、数理逻辑学家。
- 1921 年出生在山东济南，1943 年西南联合大学数学系毕业，1945 年清华大学哲学系毕业，师从著名逻辑学家金岳霖。
- 1948 年哈佛大学逻辑学博士毕业，成为哈佛的助理教授。
- 1956–1961 年任牛津大学数学哲学高级讲师。
- 1961–1967 年回到哈佛任数理逻辑与应用数学教授。
- 他在 1958 年夏天写的程序在 IBM-704 上, 只用九分钟就证明了罗素《数学原理》中一阶逻辑的全部定理。
- 在 1983 年于国际人工智能联合会议荣获首届证明自动化里程碑奖（the first Milestone Prize for Automated Theorem-Proving）

Refutation exercise

- Some patients like all doctors.
- No patient likes any quack.
- Therefore no doctor is a quack.

Use predicates: $P(x), D(x), Q(x), L(x, y)$

Refutation exercise

- Whoever can read is literate.
- Dolphins are not literate.
- Flipper is an intelligent dolphin.
- Who is intelligent but cannot read.

Use predicates: $R(x), L(x), D(x), I(x)$