Knowledge Representation and Reasoning

University "Politehnica" of Bucharest
Department of Computer Science

Fall 2009

Adina Magda Florea
http://turing.cs.pub.ro/krr_09
curs.cs.pub.ro

Lecture 2

Lecture outline

- Logic based representations
- Automated reasoning
- Predicate logic
- Herbrand theorem
- Powerful inference rules

1. Logic based representations

2 possible aims

- to make the system function according to the logic
- to specify and validate the design

- Conceptualization of the world / problem
- Syntax - wffs
- Semantics - significance, model
- *Model* - the domain interpretation for which a formula is true
- Model - linear or structured
- $M \models_S \varphi$ - "φ is true or satisfied in component S of the structure M"

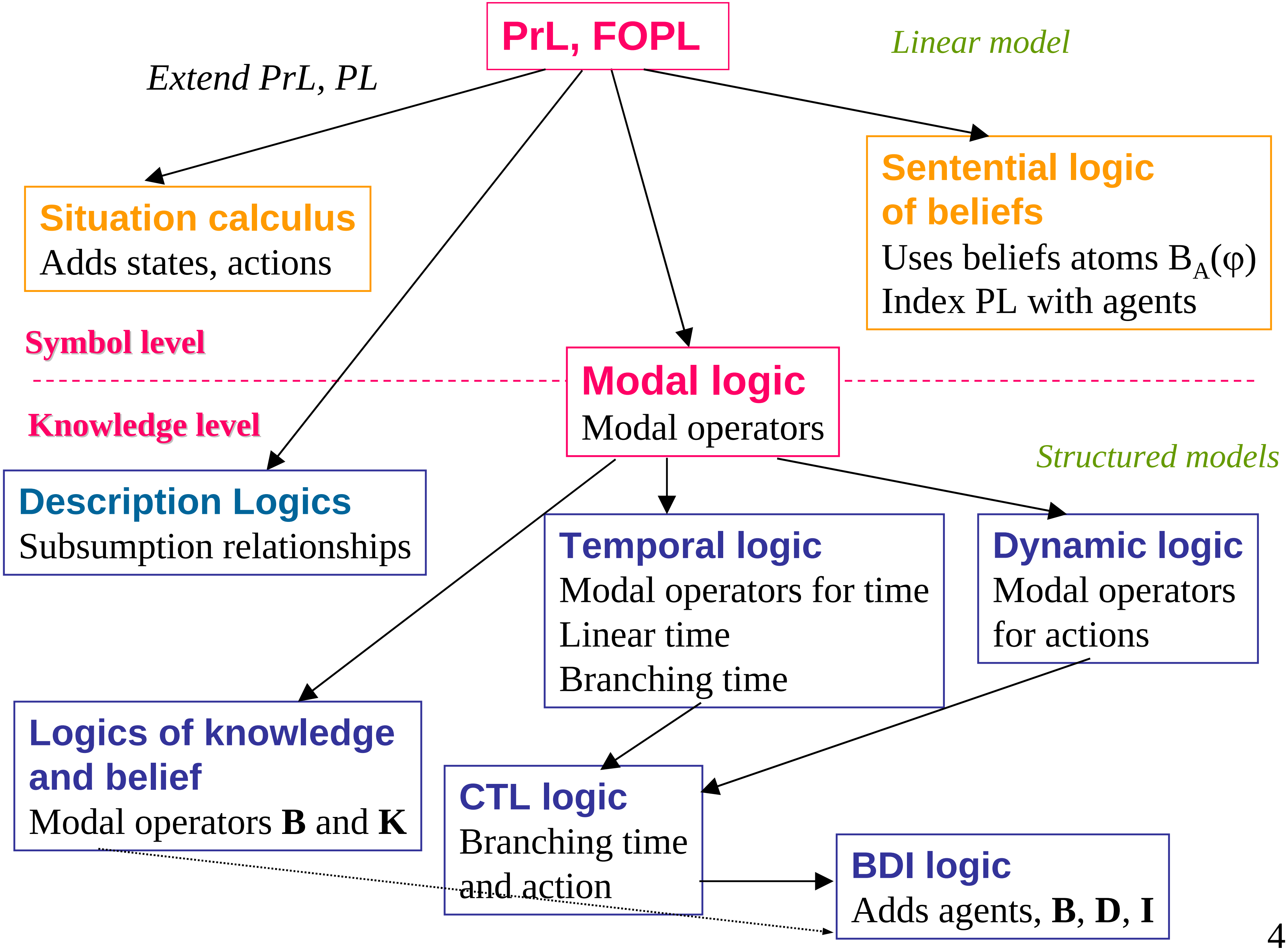
Model theory

- Generate new wffs that are necessarily true, given that the old wffs are true - *entailment* $KB \models_L \varphi$

Proof theory

- Derive new wffs based on axioms and inference rules

$KB \vdash_i \varphi$


PrL, FOPL
Extend PrL, PL
Linear model
Situation calculus
Adds states, actions
Sentential logic
of beliefs
Uses beliefs atoms $B_A(\varphi)$
Index PL with agents
Symbol level
Modal logic
Modal operators
Knowledge level
Structured models
Description Logics
Subsumption relationships
Temporal logic
Modal operators for time
Linear time
Branching time
Dynamic logic
Modal operators
for actions
Logics of knowledge
and belief
Modal operators **B** and **K**
CTL logic
Branching time
and action
BDI logic
Adds agents, **B**, **D**, **I**

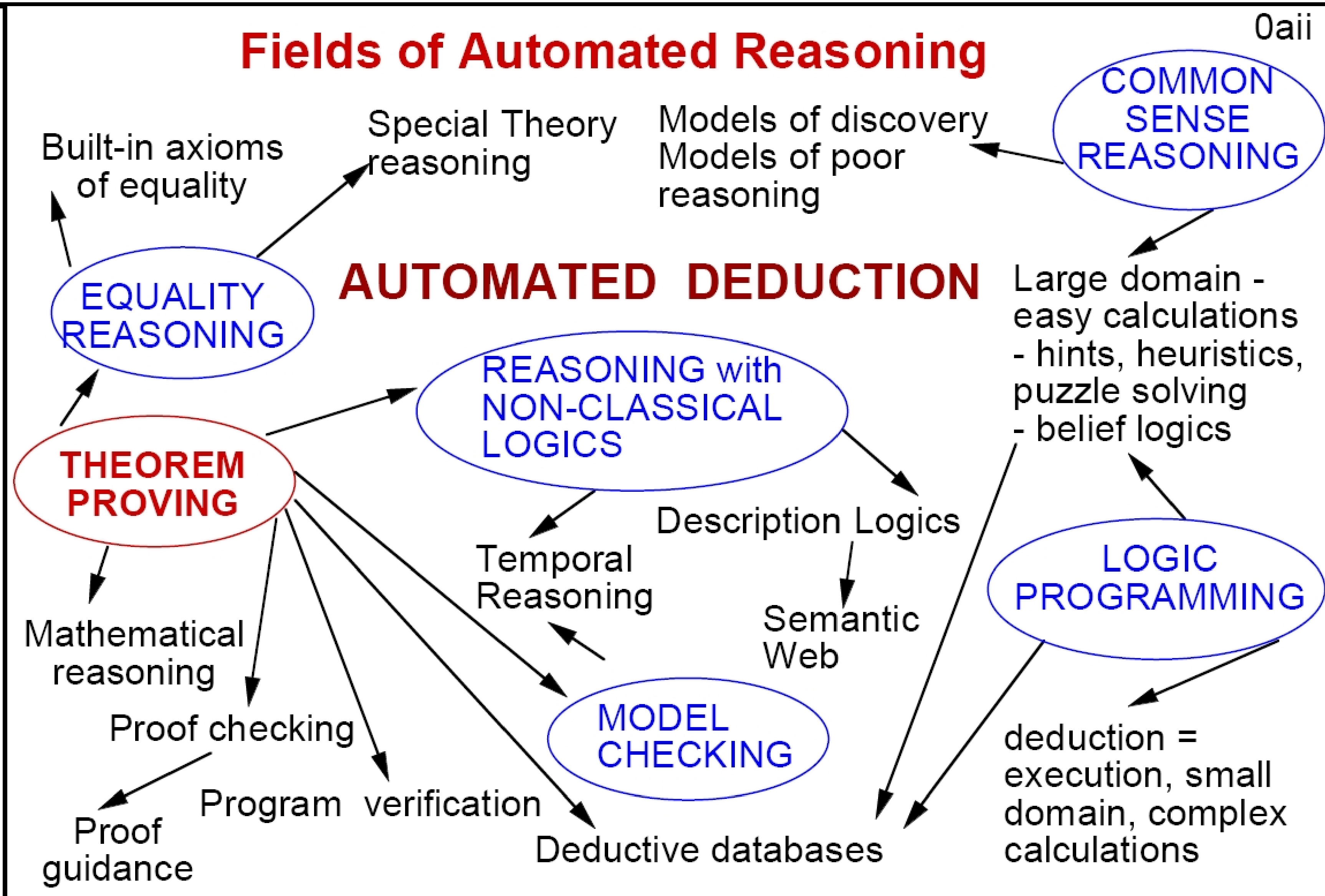
knowledge	propositional	first-order
Paul is a man	a	man(Paul)
Bill is a man	b	man(Bill)
men are mortal	c	$(\forall x)\ (man(x) \supset mortal(x))$

First order logic

knowledge	first-order	second-order
smaller is transitive	$(\forall x)\ ((\forall y)\ ((\forall z)\ ((<(x,y) \wedge <(y,z) \supset <(x,z)))))$	transitive(<)
part-of is transitive	$(\forall x)\ ((\forall y)\ ((\forall z)\ ((part\text{-}of(x,y) \wedge part\text{-}of(y,z) \supset part\text{-}of(x,z)))))$	transitive(part-of)
R is transitive iff whenever R(x,y) and R(y,z) hold, R(x,z) holds too	not expressible (see however pseudo-second order)	$(\forall R)\ ((transitive(R) \equiv (\forall x)\ ((\forall y)\ ((\forall z)\ ((R(x,y) \wedge R(y,z) \supset R(x,z)))))))$

Higher order logic

2. Automated Reasoning



Application: Compiler Validation

Problem: prove equivalence of source and target program

Example:

```
1:  y := 1
2:  if z = x*x*x
3:    then y := x*x + y
4:  endif
```

```
1: y := 1
2: R1 := x*x
3: R2 := R1*x
4: jmpNE(z,R2,6)
5: y := R1+1
```

To prove: (indexes refer to values at line numbers; index 0 = initial values)

$$y_1 \approx 1 \wedge z_0 \approx x_0 * x_0 * x_0 \wedge y_3 \approx x_0 * x_0 + y_1$$

$$y'_1 \approx 1 \wedge R1_2 \approx x'_0 * x'_0 \wedge R2_3 \approx R1_2 * x'_0 \wedge z'_0 \approx R2_3 \wedge y'_5 \approx R1_2 + 1$$

$$\wedge\, x_0 \approx x'_0 \wedge y_0 \approx y'_0 \wedge z_0 \approx z'_0 \quad \models \quad y_3 \approx y'_5$$

A logical puzzle

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.

A killer always hates his victim, and is never richer than his victim.

Charles hates no one that Aunt Agatha hates.

Agatha hates everyone except the butler.

The butler hates everyone not richer than Aunt Agatha.

The butler hates everyone Aunt Agatha hates.

No one hates everyone.

Agatha is not the butler.

Who killed Aunt Agatha?

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

- $\exists x\ (\mathrm{lives\_at\_dreadbury}(x) \wedge \mathrm{killed}(x, \mathrm{a}))$

Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.

- $\forall x\ (\mathrm{lives\_at\_dreadbury}(x) \leftrightarrow (x = \mathrm{a} \vee x = \mathrm{b} \vee x = \mathrm{c}))$

A killer always hates his victim, and is never richer than his victim.

- $\forall x, y\ (\text{killed}(x, y) \rightarrow \text{hates}(x, y))$
  $\forall x, y\ (\text{killed}(x, y) \rightarrow \neg\, \text{richer}(x, y))$

Charles hates no one that Aunt Agatha hates.

- $\forall x\ (\text{hates}(\text{c}, x) \rightarrow \neg\, \text{hates}(\text{a}, x))$

Agatha hates everyone except the butler.

- $\forall x\ (\neg\, \text{hates}(\text{a}, x) \leftrightarrow x = \text{b})$

The butler hates everyone not richer than Aunt Agatha.

- $\forall x\ (\neg\ \mathrm{richer}(x, \mathrm{a}) \rightarrow \mathrm{hates}(\mathrm{b}, x))$

The butler hates everyone Aunt Agatha hates.

- $\forall x\ (\mathrm{hates}(\mathrm{a}, x) \rightarrow \mathrm{hates}(\mathrm{b}, x))$

No one hates everyone.

- $\forall x\ \exists y\ (\neg\ \mathrm{hates}(x, y))$

Agatha is not the butler.

- $\neg\ \mathrm{a} = \mathrm{b}$

$$\frac{\mathsf{killed}(x, y) \rightarrow \mathsf{hates}(x, y) \qquad \mathsf{hates}(\mathsf{c}, y) \rightarrow \neg\, \mathsf{hates}(\mathsf{a}, y)}{\mathsf{killed}(\mathsf{c}, y) \rightarrow \neg\, \mathsf{hates}(\mathsf{a}, y)}$$

$$\frac{\mathsf{killed}(\mathsf{c}, y) \rightarrow \neg\, \mathsf{hates}(\mathsf{a}, y) \qquad \neg\, \mathsf{hates}(\mathsf{a}, y) \rightarrow y = \mathsf{b}}{\mathsf{killed}(\mathsf{c}, y) \rightarrow y = \mathsf{b}}$$

$$\frac{\mathsf{killed}(\mathsf{c}, y) \rightarrow y = \mathsf{b} \qquad \neg\, \mathsf{a} = \mathsf{b}}{\neg\, \mathsf{killed}(\mathsf{c}, \mathsf{a})}$$

3. Predicate logic - syntax

- variables
- function symbols
- terms
- predicate symbols
- atoms
- Boolean connectives
- quantifiers
- The function symbols and predicate symbols, each of given arity, comprise a signature Σ.
- A ***ground term*** is a term without any variables

Predicate logic - semantics

- **Universe** (aka Domain) : Set U
- **Variables** – values in U
- **Function symbols** – (total) functions over U
- **Predicate symbols** – relations over U
- **Terms** – values in U
- **Formulas** – Boolean truth values
- *The underlying mathematical concept is that of a Σ-algebra.*
- Interpretation of a formula

Algorithmic problems

- **Validity(F): |= F ?** (is F true in every interpretation?)
- **Satisfiability (F): F satisfiable?**
- **Entailment (F, G): F |= G?** (does F entail G?)
- **Model(A,F): A |=F ?**
- **Solve (A,F): find an assignment β such that A, β |=F**
- **Solve (F): find a substitution α such that |=Fα**
- **Abduce(F):** find G with "certain properties: such that G |= F

Refutation Theorem Proving

Suppose we want to prove $H \models G$.

- Equivalently, we can prove that
  - $F := H \rightarrow G$ is valid.

- Equivalently, we can prove that
  - $\sim F$, i.e., $H \wedge \sim G$ is unsatisfiable.

- *This principle of "refutation theorem proving" is the basis of almost all automated theorem proving methods.*

Normal forms

- Study of normal forms is motivated by:
  - reduction of logical concepts
  - efficient data structures for theorem proving.
- The main problem in first-order logic is the treatment of quantifiers. The normal form transformations are intended to eliminate many of them.

Normal forms

- Prenex normal form

prefix matrix

$$(Q_1 x_1)\ldots(Q_n x_n)M \qquad (Q_i x_i), i = 1,\ldots, n,$$

$$(\exists x_i) \quad (\forall x_i)$$

- CNF
- Eliminate existential quantifiers and conjunctions => Normal form

Herbrand universe

- **Herbrand universe** H_0 $H_0 = \{a\}$

$$H_{i+1} = H_i \cup \{f(t_1, \ldots, t_n) \mid t_j \in H_i, 1 \le j \le n\}$$

$$\mathbf{H} = \lim_{i \to \infty} H_i$$

- **Herbrand base**

$$A = \{P(t_1, \ldots, t_n) \mid t_i \in H, 1 \le i \le n, P \text{ is in } S\}$$

- Ground instances of a clause

Examples

$$S = \{P(a), \sim P(x) \vee P(f(x))\}$$

$$H_0 = \{a\}$$

$$H_1 = \{a, f(a)\}$$

$$H_2 = \{a, f(a), f(f(a))\}$$

$$S = \{P(f(x), a, g(y), b)\}$$

$$H_0 = \{a, b\}$$

$$H_1 = \{a, b, f(a), f(b), g(a), g(b)\}$$

$$H_2 = \{a, b, f(a), f(b), g(a), g(b), f(f(a)), f(f(b)), f(g(a)), f(g(b)), g(g(a)), g(g(b)), g(f(a)), g(f(b))\}$$

Herbrand interpretation

- H-interpretation

$$S = \{P(x) \vee Q(x), R(f(y))\}$$

$$H = \{a, f(a), f(f(a)), \ldots\}$$

$$A = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)), \ldots\}$$

$$I_1 = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)), \ldots\}$$

$$I_2 = \{\sim P(a), \sim Q(a), \sim R(a), \sim P(f(a)), \sim Q(f(a)), \sim R(f(a)), \ldots\}$$

Herbrand interpretation

- H-interpretation **I*** correspinding to an interpretation **I** for a set of clauses S

$$S = \{P(x), Q(y, f(y,a))\}$$

a	f(1,1)	f(1,2)	f(2,1)	f(2,2)
2	1	2	2	1

P(1)	P(2)	Q(1,1)	Q(1,2)	Q(2,1)	Q(2,2)
a	**f**	**f**	**a**	**f**	**a**

$$A = \{P(a), Q(a,a), P(f(a,a)), Q(a, f(a,a)), Q(f(a,a), a), Q(f(a,a), f(a,a))\}$$

$P(a) = P(2) = \mathbf{f}$ $\quad Q(a,a) = Q(2,2) = \mathbf{a}$ $\quad P(f(a,a)) = P(f(2,2)) = P(1) = \mathbf{a}$

$Q(a, f(a,a)) = Q(2, f(2,2)) = Q(2,1) = \mathbf{f}$ $\quad Q(f(a,a), a) = Q(f(2,2), 2) = Q(1,2) = \mathbf{a}$

$Q(f(a,a), f(a,a)) = Q(f(2,2), f(2,2)) = Q(1,1) = \mathbf{f}$

$$I^* = \{\sim P(a), Q(a,a), P(f(a,a)), \sim Q(a, f(a,a)), Q(f(a,a), a), \sim Q(f(a,a), f(a,a)), \ldots\}$$

Herbrand theorem

- **Lemma**. If an interpretation **I** over a domain D satisfies the set of clauses S (i.e., the set S is satisfiable under that interpretation), then any *H-interpretation* **I*** corresponding to **I** also satisfies S

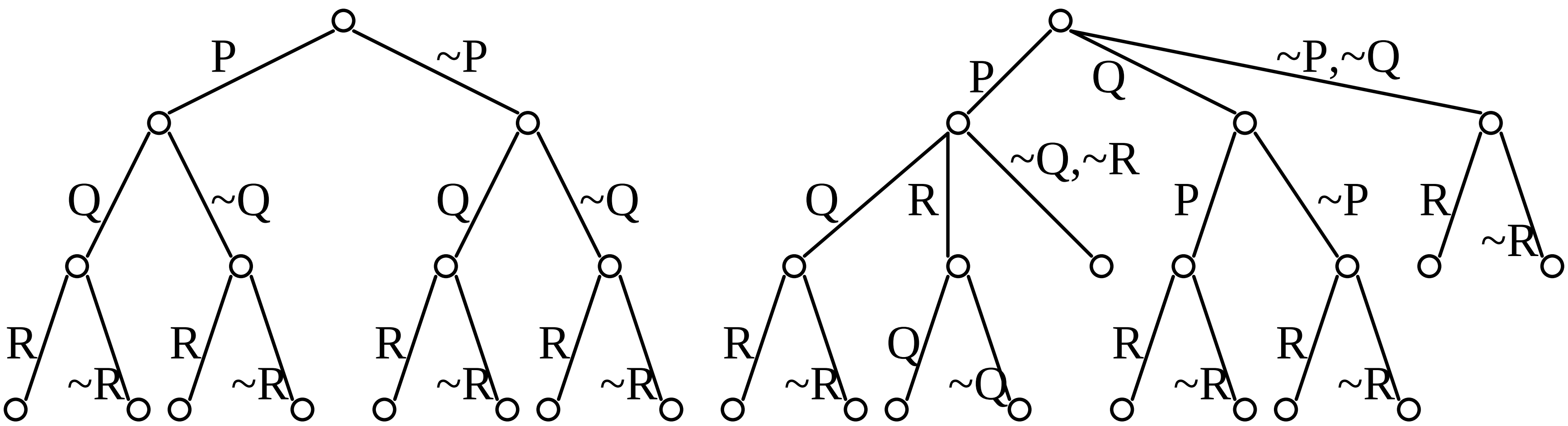
- **Theorem**. A set of clauses S is inconsistent iff *S is false for any H-interpretation of S*

Semantic Trees

- Semantic trees
- Complete semantic trees

Herbrand base of S is

$$A = \{P, Q, R\}$$

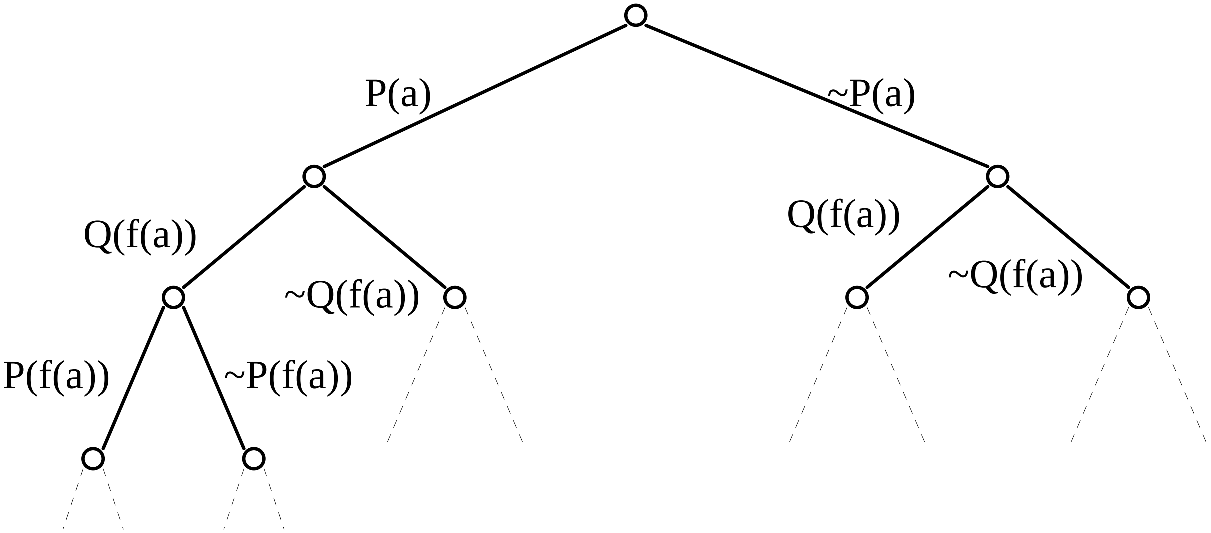


Semantic Trees

$$S = \{P(x), Q(f(x))\}$$

Herbrand base of S is

$$A = \{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), Q(f(f(a))), \ldots\}$$

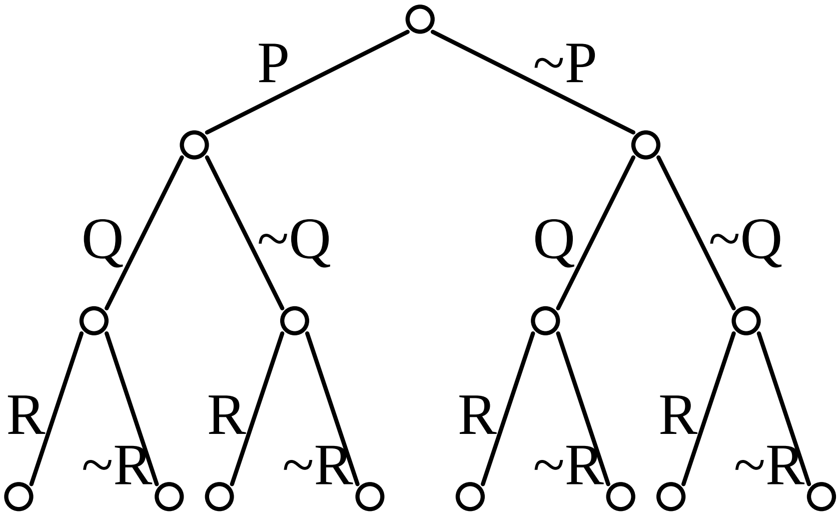


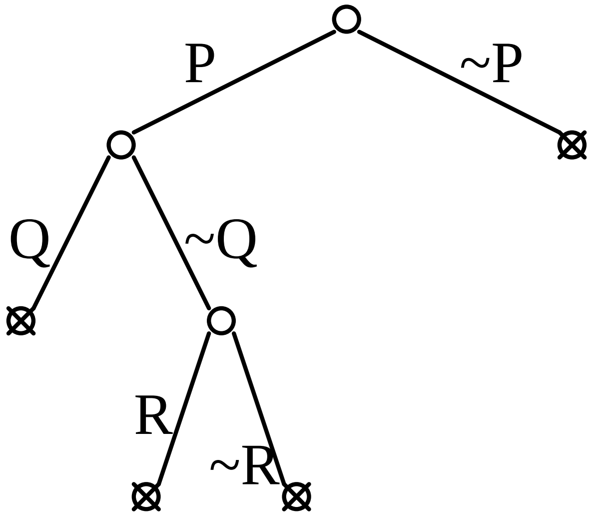
- Complete semantic tree

Closed semantic trees

$$S = \{P, Q \vee R, \sim P \vee \sim Q, \sim P \vee \sim R\}$$

$$A = \{P, Q, R\}$$

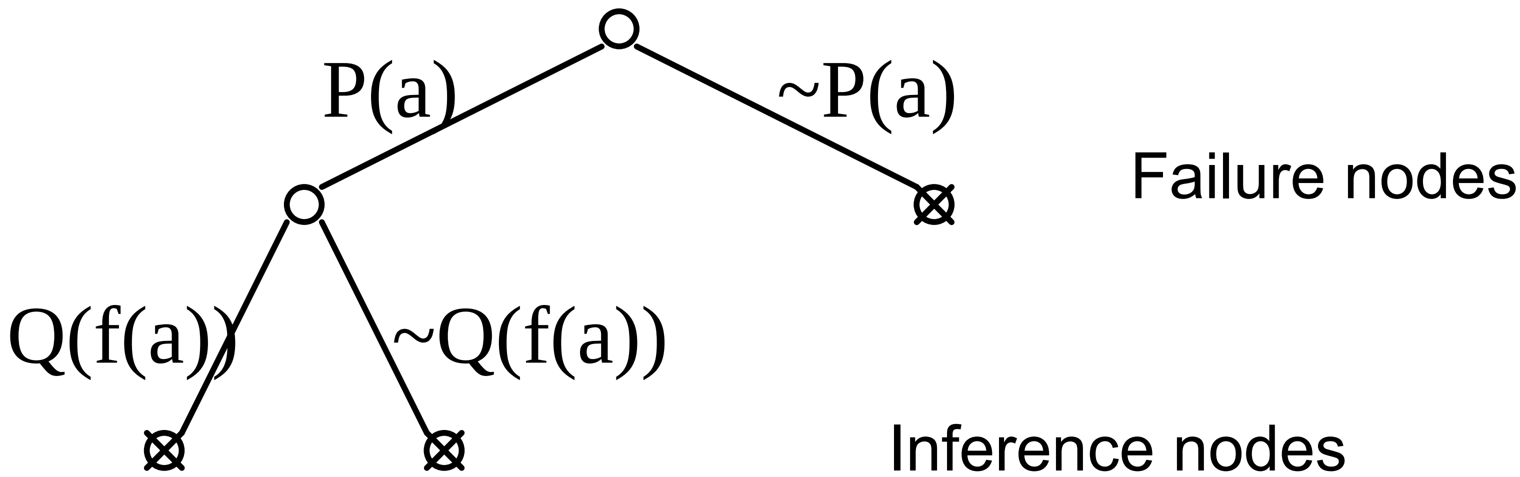




Closed semantic trees

$$S = \{P(x), \sim P(x) \vee Q(f(x)), \sim Q(f(a))\}$$

$$A = \{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), Q(f(f(a))), \ldots\}$$



Herbrand's theorem

- Idea: to test if a set S of clauses is unsatisfiable we have to test if S is unsatisfiable only for H-interpretations (interpretations over the Herbrand universe)

First version of HT

- A set of clauses S is unsatisfiable **iff** for any semantic tree of S there exists <u>a finite closed semantic tree</u>
- *(any complete semantic tree of S is a closed semantic tree)*

Herbrand's theorem

Second version of HT

- A set of clauses S is unsatisfiable **iff** there exists a finite set S' of ground instances of S which is unsatisfiable

- *(the Herbrand base of S is unsatisfiable)*

Powerful inference rules - **Resolution**

Resolution

- Binary resolution
- Factorization
- General resolution

Factorization – Russell's antinomy

A barber shaves men if and only if they do not shave themselves. Should the barber shave himself or not?

(A1) ~Shaves(x,x) ⇒ Shaves(barber,x)

(A2) Shaves(barber,y) ⇒ ~Shaves (y,y)

(C1) Shaves(x,x) ∨ Shaves (barber,x)

(C2) ~Shaves (barber,y) ∨ ~Shaves (y,y)

(Res1) ~Shaves (barber,x) ∨ Shaves (barber,x)

(Res2) Shaves(barber,barber) ∨ ~Shaves (barber,barber)

(FC1): Shaves (barber,barber)

(FC2): ~Shaves (barber,barber)

See also
http://en.wikipedia.org/wiki/Russell%27s_paradox

Factorization – Russell's antinomy

Prover 9

-Shaves(x,x) -> Shaves(barber,x).

Shaves(barber,y) -> - Shaves (y,y).

1 -Shaves(x,x) -> Shaves(barber,x) # label(non_clause). [assumption].

2 Shaves(barber,x) -> -Shaves(x,x) # label(non_clause). [assumption].

3 Shaves(x,x) | Shaves(barber,x). [clausify(1)].

4 -Shaves(barber,x) | -Shaves(x,x). [clausify(2)].

5 Shaves(barber,barber). [factor(3,a,b)].

6 $F. [factor(4,a,b),unit_del(a,5)].

Resolution

- **Theorem.** Resolution is sound. Thai is, all derived formulas are entailed by the given ones
- **Theorem:** Resolution is refutationally complete.
- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause eventually.
- If a clause set is unsatisfiable and closed under the application of resolution inference rule then it contains the empty clause.

Powerful inference rules: **Paramodulation**

- C1: P(a)
- C2: a=b
- If C1 contains a term t and there is a unity clause C2: t=s then we can infer a new clause from C1 by the substitution of a single occurrence of t in C1 with s.
- Paramodulation is a generalisation of that rule

Paramodulation

- Be C1 and C2 two clauses, which have no variables in common. If

  $C1: L[t] \vee C1'$

  $C2: r = s \vee C2'$

- where L[t] is a literal containing t , C1' and C2' are clauses, and β= mgu(t,r), then we can infer by paramodulation

  $L\beta\ [s\beta] \ \vee\ C1'\beta\ \vee\ C2'\beta$

- where $L\beta\ [s\beta]$ is obtained by replacing only one single occurrence of $t\beta$ in $L\beta$ with $s\beta$.
- *Binary paramodulant*

Paramodulation

- Paramodulation with factorization – general paramodulation
- Paramodulation with resolution is sound and refutationally complete

Example

Group axioms

% Associativity

(x * (y * z)) = ((x * y) * z).

% Identity element

((x * e) = x) & ((e * x) = x).

% Inverse element

((x * i(x)) = e) & ((i(x) * x)=e).

Prove

% R*ight regular element*

((f1 * f2) = (f0 * f2)) -> (f1 = f0).

- **1 x * e = x & e * x = x [assumption].**
- **2 x * i(x) = e & i(x) * x = e [assumption].**
- **3 f1 * f2 = f0 * f2 -> f1 = f0 [goal].**
- **4 x * (y * z) = (x * y) * z. [assumption].**
- 5 <u>(x * y)</u> * z = x * (y * z). [copy(4),flip(a)].
- 6 x * e = x. [clausify(1)].
- 7 <u>e * x</u> = x. [clausify(1)].
- 8 <u>x * i(x)</u> = e. [clausify(2)].
- 9 i(x) * x = e. [clausify(2)].
- 10 f0 * f2 = f1 * f2. [deny(3)].
- 11 f1 * f2 = f0 * f2. [copy(10),flip(a)].
- 12 f0 != f1. [deny(3)].
- 13 f1 != f0. [copy(12),flip(a)].
- **14 x * (i(x) * y) = y.** [para(8(a,1),5(a,1,1)),rewrite([7(2)]),flip(a)].
- **<u>e * z</u> = x * (i(x) * z) [8,5]**
- **z = x * (i(x) * z)   [7]   y = x * (i(x) * y)**
- **x * (i(x) * y) = y [flip]**

- 15 x * (y * i(x * y)) = e. [para(8(a,1),5(a,1)),flip(a)].
- 17 i(x) * (x * y) = y. [para(9(a,1),5(a,1,1)),rewrite([7(2)]),flip(a)].
- 22 i(f1) * (f0 * f2) = f2. [para(11(a,1),17(a,1,2))].
- 27 i(f1) * f0 = e.
  [para(22(a,1),15(a,1,2,2,1)),rewrite([5(8),8(7),6(5)])].
- 29 f1 = f0. [para(27(a,1),14(a,1,2)),rewrite([6(3)])].
- 30 $F. [resolve(29,a,13,a)].

Credit

Slides 6,7,9,10,11,12 are from the slides

First-Order Theorem Proving
Peter Baumgartner
NICTA, Logic and Computation Program, Canberra
Peter.Baumgartner@nicta.com.au