CogSysI Lecture 6: Inference in FOL

Intelligent Agents

Ute Schmid (lecture)
Emanuel Kitzelmann (practice)

Applied Computer Science, Bamberg University

last change: May 13, 2008

Remember ...

... in the last lecture we started to introduce resolution.

- Resolution calculus is a basic approach for performing logical proofs on a machine.
- Logical formula must be rewritten into clause form, using equivalence rules.
- To perform a resolution step on a pair of clauses, literals must be unified.

Clause Form

- Conjunctive Normalform (CNF): Conjunction of disjunctions of literals
  $\wedge_{i=1}^{n}(\vee_{j=1}^{m} L_{ij})$
- Clause Form: Set of disjunctions of literals (can be generated from CNF)

Rewriting of formulas to clause form:
8 steps, illustrated with example

$$
\begin{aligned}
\forall x[B(x) \rightarrow \quad & (\exists y[O(x,y) \wedge \neg P(y)] \\
\wedge \quad & \neg \exists y[O(x,y) \wedge O(y,x)] \\
\wedge \quad & \forall y[\neg B(y) \rightarrow \neg E(x,y)]\ )\ ]
\end{aligned}
$$

Clause Form cont.

(1) Remove Implications

$\forall x[\underline{\neg B(x)} \vee (\exists y[O(x,y) \wedge \neg P(y)] \wedge \neg \exists y[O(x,y) \wedge O(y,x)] \wedge$
$\forall y[\underline{\neg(\neg B(y))} \vee \neg E(x,y)]\ )\ ]$

(2) Reduce scopes of negation

$\forall x[\neg B(x) \vee (\exists y[O(x,y) \wedge \neg P(y)] \wedge \forall y[\neg O(x,y) \vee \neg O(y,x)] \wedge$
$\forall y[B(y) \vee \neg E(x,y)]\ )\ ]$

(3) Skolemization (remove existential quantifiers)

Replace existentally quantified variables by constant/function symbols.

$\exists$x p(x) becomes p(C)

("There exists a human who is a student." is satisfiable if there exists a constant in the universe $\mathcal{U}$ for which the sentence is true. "Human C is a student." is satisfiable if the constant symbol C can be interpreted such that relation p is true.)

Clause Form cont.

Skolemization cont.

If an existentially quantified variable is in the scope of a universally quantified variable, it is replaced by a function symbol dependent of this variable:

$\forall x \, \exists y \; p(x) \wedge q(x, y)$ becomes $\forall x \; p(x) \wedge q(x, f(x))$

("For all x holds, x is a positive integer and there exists a y which is greater than x." is satisfiable if for each x exists an y such that the relation "greater than" holds. E.g., $f(x)$ can be interpreted as successor-function.)

Skolemization is no equivalence transformation. A formula and its Skolemization are only equivalent with respect to satisfiability! The skolemized formula has a model iff the orginal formula has a model.

$$\forall x [\neg B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge \forall y [\neg O(x, y) \vee \neg O(y, x)] \wedge \forall y [B(y) \vee \neg E(x, y)] \;) \;]$$

Clause Form cont.

(4) Standardize variables ("bounded renaming")

A variable bound by a quantifier is a "dummy" and can be renamed. Provide that each variable of universal quantor has a different name. (Problematic case: free variables)

$$\forall x[\neg B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge \forall y[\neg O(x, y) \vee \neg O(y, x)] \wedge \forall z[B(z) \vee \neg E(x, z)]\ )\ ]$$

(5) Prenex-form

Move universal quantifiers to front of the formula.

$$\forall x \forall y \forall z[B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge (\neg O(x, y) \vee \neg O(y, x)) \wedge (B(z) \vee \neg E(x, z))\ )\ ]$$

(6) CNF

(Repeatedly apply the distributive laws)

$$\forall x \forall y \forall z[(\neg B(x) \vee O(x, f(x))) \wedge (\neg B(x) \vee \neg P(f(x))) \wedge (\neg B(x) \vee \neg O(x, y) \vee \neg O(y, x)) \wedge (\neg B(x) \vee B(z) \vee \neg E(x, z))\ ]$$

Clause Form cont.

(7) Eliminate Conjunctions

If necessary, rename variable such that each disjunction has a different set of variables.
The truth of a conjunction entails that all its parts are true.

$\forall x[\neg B(x) \vee O(x, f(x))],\ \forall w[\neg B(w) \vee \neg P(f(w))],\ \forall u\,\forall y[\neg B(u) \vee \neg O(u, y) \vee \neg O(y, u)],\ \forall v\,\forall z[\neg B(v) \vee B(z) \vee \neg E(v, z)]$

(8) Eliminate Universal Quantifiers

Clauses are implicitely univerally quantified.

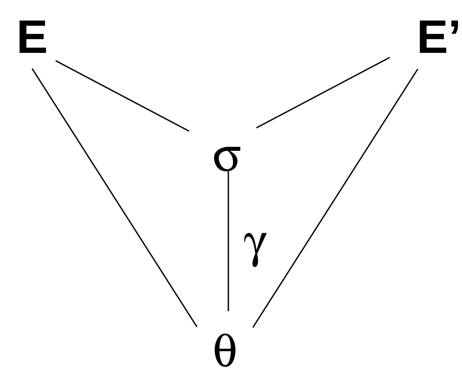
$M = \{\neg B(x) \vee O(x, f(x)), \neg B(w) \vee \neg P(f(w)), \neg B(u) \vee \neg O(u, y) \vee \neg O(y, u), \neg B(v) \vee B(z) \vee \neg E(v, z)\}$

Substitution

- A substitution is a set $\theta = \{v_1 \leftarrow t_1, \dots v_n \leftarrow t_n\}$ of replacements of variables v_i by terms t_i.
- If θ is a substitution and E an expression, $E' = E\theta$ is called instance of E. E' was derived from E by applying θ to E.
- Example: $E = p(x) \vee (\neg q(x, y) \wedge p(f(x)))$, $\theta = \{x \leftarrow C\}$, $E\theta = p(C) \vee (\neg q(C, y) \wedge p(f(C)))$
- Special case: alphabetic substitution (variable renaming).
- Composition of substitutions: Let be $\theta = \{u_1 \leftarrow t_1, \dots u_n \leftarrow t_n, v_1 \leftarrow s_1, \dots v_k \leftarrow s_k\}$ and $\sigma = \{v_1 \leftarrow r_1, \dots v_k \leftarrow r_k, w_1 \leftarrow q_1, \dots w_m \leftarrow q_m\}$. The composition is defined as $\theta\sigma =_{Def} \{u_1 \leftarrow t_1\sigma, \dots u_n \leftarrow t_n\sigma, v_1 \leftarrow s_1\sigma, \dots v_k \leftarrow s_k\sigma, w_1 \leftarrow q_1, \dots w_m \leftarrow q_m\}$
- Composition of substitutions is not commutative!

Unification

- Let be $\{E_1 \dots E_n\}$ a set of expressions. A substitution θ is a unificator of $E_1 \dots E_n$, if $E_1\theta = E_2\theta \dots = E_n\theta$.
- A unificator θ is called most general unifier (mgu), if for each other unificator σ for $E_1 \dots E_n$ there exists a substitution γ with $\sigma = \theta\gamma$.
- Theorem: If exists a unificator, then exists an mgu.



There are lots of unification algorithms, e.g. one proposed by Robinson.

Examples

(1)	$\{P(x), P(A)\}$	$\theta = \{x \leftarrow A\}$
(2)	$\{P(f(x), y, g(y)), P(f(x), z, g(x))\}$	$\theta = \{y \leftarrow x, z \leftarrow x\}$
(3)	$\{P(f(x, g(A, y)), g(A, y)), P(f(x, z), z)\}$	$\theta = \{z \leftarrow g(A, y)\}$
(4)	$\{P(x, f(y), B), P(x, f(B), B)\}$	$\theta = \{x \leftarrow A, y \leftarrow B\}$
		$\sigma = \{y \leftarrow B\}$

In (4) holds:
σ is more general than θ: $\theta = \sigma\gamma$, with $\gamma = \{x \leftarrow A\}$
σ is mgu for $\{P(x, f(y), B), P(x, f(B), B)\}$

Unification Algorithm

For a given set of formula S:

1. Let be $\theta = \{\}$
2. While $|S| > 1$ DO
   (a) Calculate the disagreement set D of S
   (b) If D contains a variable x and a term t in which x does not occur
   Then $\theta = \theta\{x \leftarrow t\}$ and $S = S\theta$
   Else stop (S not unifiable)
3. Return θ as mgu of S

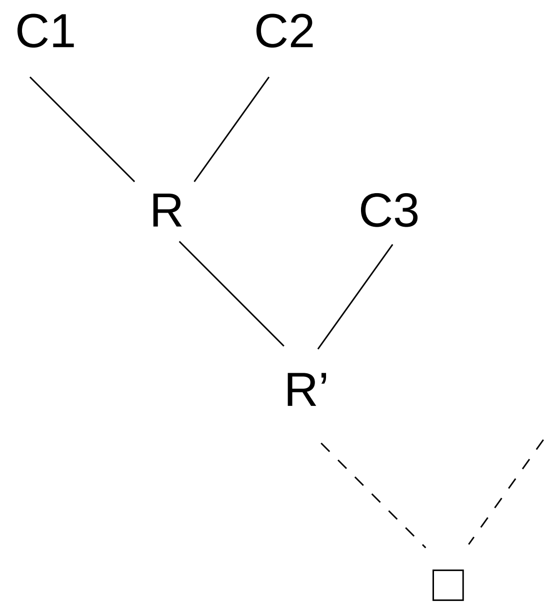
Resolution

A clause $C = \vee_{i=1}^{n} L_i$ can be written as set $C = \{L_1, \ldots L_n\}$. Let be C_1, C_2 and R clauses. R is called resolvent of C_1 and C_2 if:

- There are alphabetical substitutions σ_1 und σ_2 such that $C_1\sigma_1$ and $C_2\sigma_2$ have no common variables.
- There exists a set of literals $L_1, \ldots L_m \in C_1\sigma_1 (m \geq 1)$ and $L'_1, \ldots L'_n \in C_2\sigma_2 (n \geq 1)$ such that $L = \{\neg L_1, \neg L_2, \ldots \neg L_m, L'_1, L'_2, \ldots L'_n\}$ are unifiable with θ as mgu of L.
- R has the form:
  $R = ((C_1\sigma_1 \setminus \{L_1, \ldots L_m\}) \cup (C_2\sigma_2 \setminus \{L'_1, \ldots L'_n\}))\theta$.

Resolution cont.

Derivation of a clause by application of the resolution rule can be described by a refutation tree:



Illustration

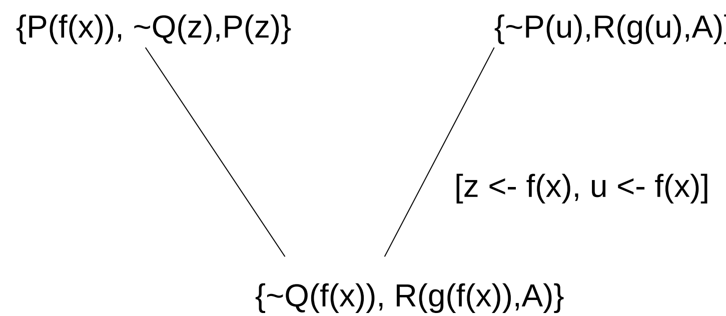
$C_1 = \{P(f(x)), \neg Q(z), P(z)\}$
$C_2 = \{\neg P(x), R(g(x), A)\}$

$\sigma_1 = \{\}$, $\sigma_2 = \{x \leftarrow u\}$

$L = \{P(f(x)), P(z), \neg\neg P(x)\} = \{P(f(x)), P(z), P(u)\}$

$\theta = \{z \leftarrow f(x), u \leftarrow f(x)\}$

$R = [(\{P(f(x)), \neg Q(z), P(z)\} \setminus \{P(f(x)), P(z)\}) \cup (\{\neg P(u), R(g(u), A)\} \setminus \{P(u)\})]\theta = \{\neg Q(f(x)), R(g(f(x)), A)\}$



Resolution Proofs

- To prove that formula G (assertion) logically follows from a set of formula (axioms) $F_1 \ldots F_n$: Include the negated assumption in the set of axioms and try to derive a contradiction (empty clause).
- Theorem: A set of clauses is not satisfiable, if the empty clause ($\Box$) can be derived with a resolution proof.
- (Contradiction: $C_1 = A, C_2 = \neg A$, stands for $(A \wedge \neg A)$ and $(A \wedge \neg A) \vdash \Box$)

Example

- Axiom "All humans are mortal" and fact "Socrates is human" (both are non-logical: their truth is presupposed)
- Assertion "Sokrates is mortal."
- Formalization:

  F_1 : ∀x Human(x) → Mortal(x)

  F_2 : Human(S)

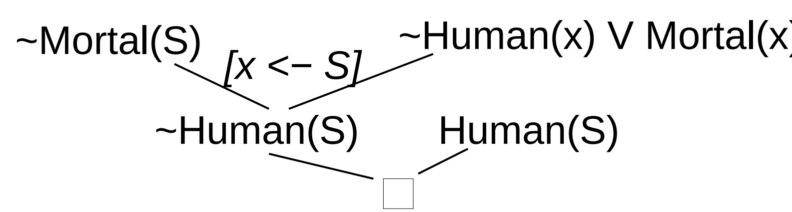
  F_3 : ¬Mortal(S) (negation of assertion)

  Clause form:

- F'_1 : ¬Human(x) ∨ Mortal(x)

  F'_2 : Human(S)

  F'_3 : ¬Mortal(S)



Soundness and Completeness of Res.

- A calculus is sound, if only such conclusions can be derived which also hold in the model.
- A calculus is complete, if all conclusions can be derived which hold in the model.
- The resolution calculus is sound and refutation complete.

  Refutation completeness means, that if a set of formula (clauses) is unsatisfiable, then resolution will find a contradiction. Resolution cannot be used to generate all logical consequences of a set of formula, but it can establish that a given formula is entailed by the set. Hence, it can be used to find all answers to a given question, using the “negated assumption” method.

Remarks

- The proof ideas will given for resolution for propositional logic (or ground clauses) only.
- For FOL, additionally, a lifting lemma is necessary and the proofs rely on Herbrand structures.
- We cover elementary concepts of logic only.
- For more details, see

  Uwe Schöning, Logik für Informatiker, 5. Auflage, Spektrum, 2000.

  Volker Sperschneider & Grigorios Antoniou, Logic – A foundation for computer science, Addison-Wesley, 1991.

Resolution Theorem

Theorem: A set of clauses F is not satisfiable iff the empty clause $\Box$ can be derived from F by resolution.

- Soundness: (Proof by contradiction)
  Assume that $\Box$ can be derived from F. If that is the case, two clauses $C_1 = \{L\}$ and $C_2 = \{\neg L\}$ must be contained in F. Because there exists no model for $L \wedge \neg L$, F is not satisfiable.
- Refutation completeness: (Proof by induction over the number n of atomar formulas in F)
  Assume that F is a set of formula which is not satisfiable.
  Because of the compactness theorem, it is enough to consider the case that a finite non-satisfiable subset of formula exists in F.
  To show: $\Box$ is derived from F. (see e.g., Schöning)

Resolution Strategies

- In general, there are many possibilities, to find two clauses, which are resolvable. Of the many alternatives, there are possibly only a few which help to derive the empty clause $\hookrightarrow$ combinatorial explosion!
- For feasible algorithms: use a resolution strategy
- E.g., exploit subsumption to keep the knowledge space, and therefore the search space, small. Remove all sentences which are subsumed (more special than) an existing sentence.
  If $P(x)$ is in the knowledge base, sentences as $P(A)$ or $P(A) \vee Q(B)$ can be removed.
- Well known efficient strategy: SLD-Resolution (*linear resolution with selection function for definite clauses*) (e.g. used in Prolog)

SLD-Resolution

- linear: Use a sequence of clauses $(C_0 \ldots C_n)$ starting with the negated assertion C_0 and ending with the empty clause C_n. Each C_i is generated as resolvent from C_{i-1} and a clausel from the original set of axioms.
- Selection function (for the next literal which will be resolved) e.g. top-down-left-to-right in PROLOG; makes the strategy incomplete! (“user” must order clauses in a suitable way)
- definite Horn clauses: A Horn clause contains maximally one posititive literal; a definite Horn clause contains exactly one positive literal (Prolog rule)

Prolog

	PROLOG	Logic	
Fact	isa(fish,animal).	isa(Fish,Animal)	positive literal
	isa(trout,fish).	isa(Trout,Fish)	
Rule	is(X,Y) :- isa(X,Y).	is(x,y) $\vee$ $\neg$isa(x,y)	definite Clause
	is(X,Z) :- isa(X,Y), is(Y,Z).	is(x,z) $\vee$ $\neg$isa(x,y) $\vee$ $\neg$is(y,z)	
Query	is(trout,animal).	$\neg$is(Trout,Animal)	Assertion
	is(Fish,X)	$\neg$is(Fish,x)	

- $:-$ denotes the "reversed" implication arrow.

  is(X,Z) :- isa(X,Y), is(Y,Z).

  $isa(x, y) \wedge is(y, z) \rightarrow is(x, z) \equiv$

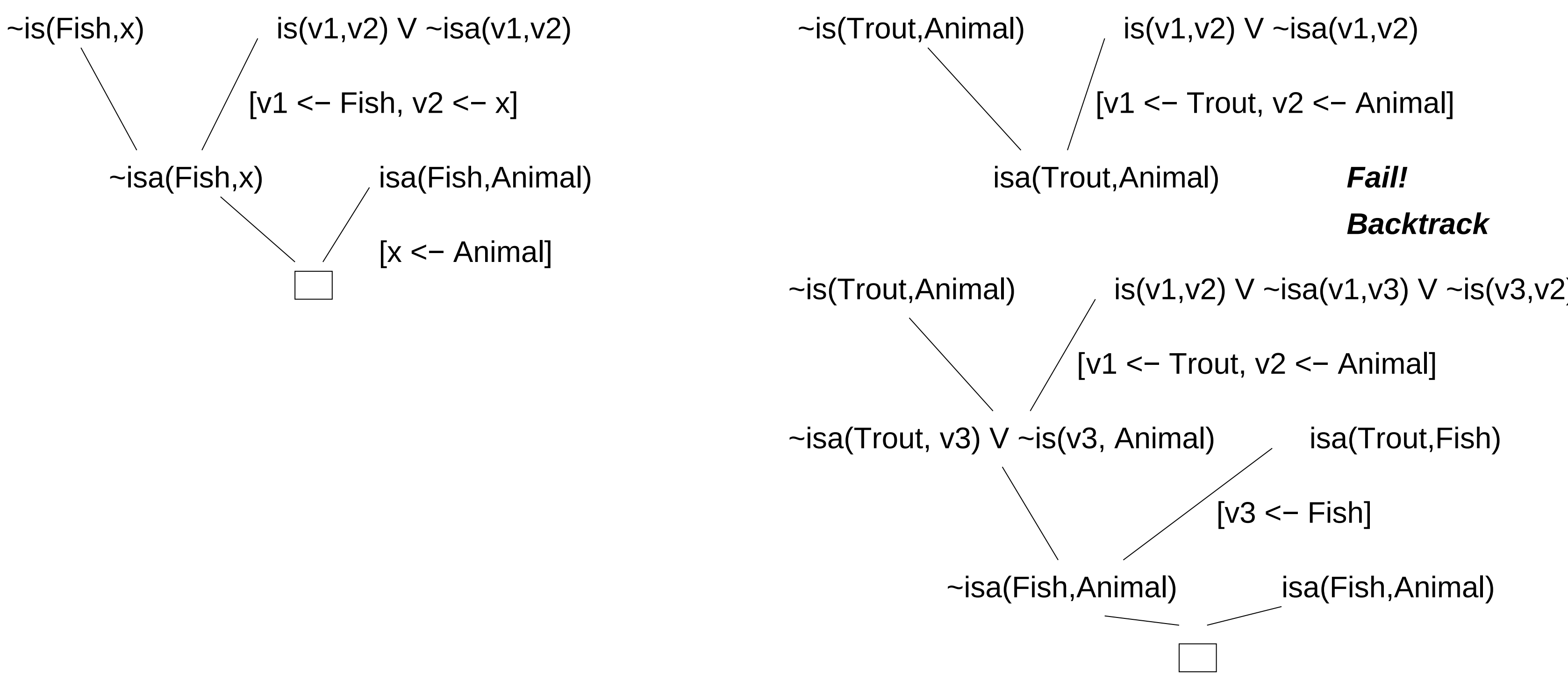
  $\neg(isa(x, y) \wedge is(y, z)) \vee is(x, z) \equiv \neg isa(x, y) \vee \neg is(y, z) \vee is(x, z)$

- Variables which occur in the head of a clause are implicitely universally quantified. Variables which occur only in the body are existentially quantified.

  $\forall x \forall z \exists y\ \neg isa(x, y) \vee \neg is(y, z) \vee is(x, z)$

Prolog Example

- Query: is(fish,X)
  (stands for $\exists x \; is(Fish, x)$)
- Negation of query: $\neg \exists x \; is(Fish, x) \equiv \forall x \; \neg is(Fish, x)$
- SLD-Resolution:*(extract)*



Remarks on Prolog

- When writing Prolog programs, one should be know how the interpreter is working (i.e., understand SLD-resolution)
- Sequence of clauses has influence whether an assertion which follows logically from a set of clauses can be derived!
- Efficiency: Facts before rules
- Termination: non-recursive rule before recursive.

```
% Program                          % Query
isa(trout,fish).                   ? is(trout,animal).
isa(fish,animal).

is(X,Z) :- is(X,Y), isa(Y,Z).     is(trout,Y),isa(Y,animal)
is(X,Y) :- isa(X,Y).               is(trout,Y'),isa(Y',animal),isa(Y,animal)
                                   ...
```

Applications of Resolution Calculus

- PROLOG
- as *a basic* method for theorem proving (others: e.g. tableaux)
- Question Answering Systems

- Yes/No-Questions: Assertion/Query $mortal(s)$
- Query $is(trout, X)$ corresponds to “What is a trout?” The variable X is instantiated during resolution and the answer is “a fish”.
- $buys(peter, john, X)$: “What does John buy from Peter?”
- $buys(peter, X, car)$: “Who buys a car from Peter?”

Theorem Provers

- Theorem provers typically are more general than Prolog:
  not only Horn clauses but full FOL; no interleaving of logic and control (i.e. ordering of formulas has no effect on result)
- Examples: Boyer-Moore (1979) theorem prover; OTTER, Isabelle
- Theorem provers for mathematics, for verification of hardware and software, for deductive program synthesis.

Forward- and Backward Chaining

- Rules (e.g. in Prolog) have the form:
  Premises $\rightarrow$ *Conclusion*
- All rule-based systems (production systems, planners, inference systems) can be realized using either forward-chaining or backward-chaining algorithms.
- Forward chaining: Add a new fact to the knowledge base and derive all consequences (data-driven)
- Backward chaining: Start with a goal to be proved, find implication sentences that would allow to conclude the goal, attempt to prove the premises, etc.
- Well known example for a backward reasoning expert system: MYCIN (diagnosis of bacterial infections)

Logic Calculi in AI

- Variants of logic calculi are part of many AI systems
- Logic and logical inference is the base of most types of knowledge representation formalisms (e.g. description logics)
- Most knowledge-based systems (e.g. expert systems) are relying on some type of deductive inference mechanism
- Often, classical logic is not adequate: non-monotonic, probabilistic or fuzzy approaches (see “Semantische Informationsverarbeitung”)
- Extensions of classical logic for dealing with time or believe: Modal Logic (e.g., BDI-Logic for Multiagent Systems)

Deductive Planning

- Deductive inference can be used to solve planning problems.
- Introduce a situation variable to store the partial plans:
  $s_{i+1} = \text{put}(A, B, s_i)$, ... $s_2 = puttable(A, s_1)$
  $s = \text{put}(A, B, \text{puttable}(A, [\text{on}(A, C), \text{clear}(A) \ldots]))$
- Situation calculus: Introduced by McCarthy (1963) and used for plan construction by resolution by Green (1969)
- In general: extensions of FOL (action languages)
- Proof logically, that a set of goals follows from an initial state given operator definitions (axioms)
- Perform the proof in a constructive way (plan is constructed as a byproduct of the proof)

Situation Calculus

A1 *on(a, table, s_1)* (literal of the initial state)

A2 $\forall\, S[$on(a, table, S) $\rightarrow$ on(a, b, put(a, b, S))$] \equiv$ (axiom for put-operator)
$\neg$*on(a, table, S)* $\vee$ *on(a, b, put(a, b, S))* *(clausal form)*

Proof the goal predicate *on(a, b, S_F)*

1. $\neg$*on(a, b, S_F)* (Negation of the theorem)
2. $\neg$ *on(a, table, S)* $\vee$ *on(a, b, put(a, b, S))* (A2)
3. $\neg$ *on(a, table, S)* (Resolve 1, 2)
   answer(put(a, b, S))
4. *on(a, table, s_1)* (A1)
5. contradiction (Resolve 3, 4) $\hookrightarrow$ answer(put(a, b, s_1))

s_2 = *on(a, table, s_1)* with *on(a, b, s_2)* exists and s_2 can be reached by putting a on b in situation s_1.

Frame Problem

- No closed world assumption $\hookrightarrow$ full expressive power of FOL
- Problem: additionally to axioms describing the effects of actions, frame axioms become necessary
- Frame axioms are necessary to allow proofing conjunctions of goal literals.
- Example for a frame axiom:
  $\forall\ S[\text{on}(Y, Z, S) \rightarrow \text{on}(Y, Z, \text{put}(X, Y, S))]$ `on(Y, Z, put(X, Y, S))` $\leftarrow$ `on(Y, Z, S)`
  After a block X was put on a block Y, it still holds that Y is lying on a block Z, if this did hold before the action was performed.

Blocksworld in Prolog

Effect Axioms:

on(X, Y, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S)
clear(Z, put(X, Y, S)) $\leftarrow$	on(X, Z, S) $\wedge$ clear(X, S) $\wedge$ clear(Y, S)
clear(Y, puttable(X, S)) $\leftarrow$	on(X, Y, S) $\wedge$ clear(X, S)
ontable(X, puttable(X, S)) $\leftarrow$	clear(X, S)

Frame Axioms:

clear(X, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S)
clear(Z, put(X, Y, S)) $\leftarrow$	clear(X,S) $\wedge$ clear(Y, S) $\wedge$ clear(Z, S)
ontable(Y, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S) $\wedge$ ontable(Y, S)
ontable(Z, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S) $\wedge$ ontable(Z, S)
on(Y, Z, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S) $\wedge$ on(Y, Z, S)
on(W, Z, put(X, Y, S)) $\leftarrow$	clear(X, S) $\wedge$ clear(Y, S) $\wedge$ on(W, Z, S)

Blocksworld in Prolog cont.

Frame Axioms cont.:

clear(Z, puttable(X, S)) $\leftarrow$ clear(X, S) $\wedge$ clear(Z, S)

ontable(Z, puttable(X, S)) $\leftarrow$ clear(X, S) $\wedge$ ontable(Z, S)

on(Y, Z, puttable(X, S)) $\leftarrow$ clear(X, S) $\wedge$ on(Y, Z, S)

clear(Z, puttable(X, S)) $\leftarrow$ on(Y, X, S) $\wedge$ clear(Y, S) $\wedge$ clear(Z, S)

ontable(Z, puttable(X, S)) $\leftarrow$ on(Y, X, S) $\wedge$ clear(Y, S) $\wedge$ ontable(Z, S)

on(W, Z, puttable(X, S)) $\leftarrow$ on(Y, X, S) $\wedge$ clear(Y, S) $\wedge$ on(W, Z, S)

Facts (Initial State):

on(d, c, s_1) on(c, a, s_1)

clear(d, s_1) clear(b, s_1)

ontable(a, s_1) ontable(b, s_1)

Theorem (Goal):

on(a, b, S) $\wedge$ on(b, c, S)

Running Gag

Question: How many AI people does it take to change a lightbulb?
Answer: At least 81. **The Logical Formalism Group (16)**

- One to figure out how to describe lightbulb changing in first order logic.
- One to figure out how to describe lightbulb changing in second order logic.
- One to show the adequacy of FOL.
- One to show the inadequacy of FOL.
- One to show that lightbulb logic is non-monotonic.
- One to show that it isn't non-monotonic.
- One to show how non-monotonic logic is incorporated in FOL.
- One to determine the bindings for the variables.
- One to show the completeness of the solution.
- One to show the consistency of the solution.
- One to show that the two just above are incoherent.
- One to hack a theorem prover for lightbulb resolution.
- One to suggest a parallel theory of lightbulb logic theorem proving.
- One to show that the parallel theory isn't complete.
- One to indicate how it is a description of human lightbulb changing behaviour.
- One to call the electrician.