Introduction to proof theory

Proof theory considers the mechanics of generating a set of sentences from others

Basics of proof theory

1. Elements of proof theory
2. Theorem proving and proof procedures
3. Resolution for propositional logic
4. Substitutions, and resolution for 1^{st} order logic
5. SLD resolution

Elements of proof theory

Proof theory considers the "derivability" of a sentence given a set of inference rules $\mathcal{R}$

– The sentences given initially are called the $axioms$, and those derived are $theorems$ (syntactic consequences)

A sentence is derivable from a set of axioms S using $\mathcal{R}$: $S \vdash_{\mathcal{R}} s$

– Axioms can be *logical* (valid sentences of logic) or *non-logical* (problem specific sentences in logic)

Axioms + $\mathcal{R}$ = Inference system

Axioms + all theorems = Theory

– A theory is consistent iff there is no sentence s s.t. the theory contains both s and $\sim s$

Soundness and completeness

We would like theorems derived to be logical consequences of the axioms provided

- We can then be sure of the correctness of the theorem in the intended model for the axioms

- Remember, logical consequences of the axioms are true in all models for the axioms

This property depends entirely on the inference rules chosen, and those that have this property are called *sound*

- if $S \vdash_{\mathcal{R}} s$ then $S \models s$

- Examples of sound inference rules:

modus ponens $\{q, p \leftarrow q\} \vdash p$

modus tollens $\{\sim p, p \leftarrow q\} \vdash \sim q$

We would also like to derive *all* logical consequences, and rules with this property are said to be *complete*

– if $S \models s$ then $S \vdash_{\mathcal{R}} s$

That is $S \models s \equiv S \vdash_{\mathcal{R}} s$

Proof procedures

Axioms and inference rules are not enough. We need a strategy to apply the rules.

– Inference system + strategy = Proof procedure

For logic programs:

– 1 inference rule: *resolution*

– Strategy: **S**elected **L**inear **D**efinite (SLD)

– Proof procedure: SLD-resolution

Resolution for propositional logic

Consider the clauses:

C_1: $is\_dangerous \leftarrow is\_cheetah$

C_2:
$is\_cheetah \leftarrow is\_carnivore, has\_tawny\_colour, has\_dark\_spots$

- The *resolvent* of C_1, C_2 is the clause:

C:

$is\_dangerous \leftarrow is\_carnivore, has\_tawny\_colour, has\_dark\_spots$

- Remember

C_1: $is\_dangerous \vee \sim is\_cheetah$

C_2: $is\_cheetah \vee \sim is\_carnivore \vee \sim has\_tawny\_colour \vee \sim has\_dark\_spots$

C: $is\_dangerous \vee \sim is\_carnivore \vee \sim has\_tawny\_colour \vee \sim has\_dark\_spots$

- C_1, C_2 are called the *parent* clauses, and $is\_cheetah$ is the the literal that is resolved upon

Soundness of resolution

A single resolution step does the following:

– From $p \leftarrow q$ and $q \leftarrow r$

– Infer $p \leftarrow r$

Since resolution is sound, we can always add the clauses inferred to the original program

Completeness of resolution

Resolution has these properties

- Consider a set of clauses s.t. each clause has *at most* 1 positive literal. Such clauses are called *Horn* clauses

- If a set of Horn clauses is unsatisfiable then resolution will derive the empty clause. Resolution is thus "refutation complete"

- However, it is not "affirmation complete". That is, if $P \models s$, then it need not follow that $P \vdash s$ using resolution

  $$\{p \leftarrow, q \leftarrow\} \models p \leftarrow q$$

- But, if $P \cup \{\sim s\} \vdash \Box$ using resolution then $P \cup \{\sim s\} \models \Box$ or $P \models s$

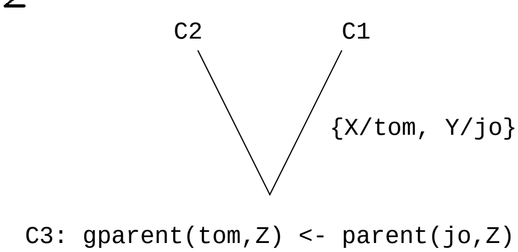
Resolution in 1^{st} order logic: substitutions

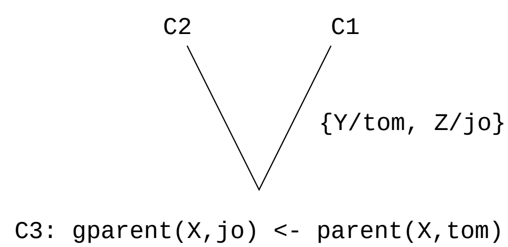
Recall the clauses:

$C_1 : \ gparent(X, Z) \ \leftarrow \ parent(X, Y), parent(Y, Z)$

$C_2 : \ parent(tom, jo) \ \leftarrow$

From earlier lectures recall that values were assigned to variables as computation proceeded. C_1 and C_2 can can be resolved in one of two ways:





– Constructing resolvents requires "substituting" some terms for variables (exactly which depends on literals being resolved)

– A mapping from variables to terms is called a *substitution*

Applying a substitution to a sentence gives a "substitution instance of" that sentence

$s = p(X, Y, f(Z))$ and $\theta = \{X/a, Y/b, Z/f(d)\}$

$s\theta = p(a, b, f(f(d)))$

We usually require the following properties of substitutions

1. They should be functional, i.e. each variable to the left of the / should be distinct

2. Idempotence, that is $(s\theta)\theta = \theta$. Each term to the right of / should not contain any variable that occurs to the left of /

Renaming	Substitution?
$\{X/Y, Y/tom\}$	
$\{X/tom, X/jo, Y/peter\}$	
$\{X/tom, Y/tom\}$	
$\{X/f(X), Y/a\}$	

Resolution in 1^{st} order logic: unification

For a single resolution step, we must somehow "match" the negative literal of one clause with the positive literal of another

$C_1: \; gparent(X,Z) \lor \sim \underline{parent(X,Y)} \lor \sim parent(Y,Z)$

$C_2: \; \underline{parent(tom,jo)}$

- What substitution θ would make the $\underline{literals}$ complementary? That is $parent(tom,jo)\theta = parent(X,Y)\theta$

  $\theta = \{X/tom, Y/jo\}$ is said to be a *unifier* for the literals

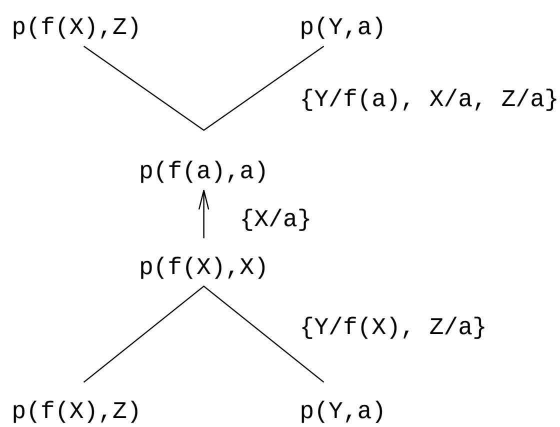
- Is $\theta = \{Y/f(a), X/a, Z/a\}$ a unifier for $p(f(X),Z)$ and $p(Y,a)$?

- What about $\theta = \{Y/f(X), Z/a\}$?

Some substituitions are "more general" than others in that they impose less severe constraints on the variables

Most general unifier θ:

– Let s_1 s_2 be two atoms (or terms) and θ a unifier. Let σ be some other unifier for $s_{1,2}$. For every $\sigma \neq \theta$ that unifies $s_{1,2}$, there is a substituition μ s.t. $\sigma = \theta \cdot \mu$



Resolution with 1^{st}-order clauses

Step 0. Given a pair of clauses:

$$C_1 : likes(steve, X) \leftarrow buys(X, ilp\_book)$$

$$C_2 : buys(X, ilp\_book) \leftarrow sensible(X), rich(X)$$

Step 1. Rename all variables apart.

$$C_1 : likes(steve, A) \leftarrow buys(A, ilp\_book)$$

$$C_2 : buys(B, ilp\_book) \leftarrow sensible(B), rich(B)$$

Step 2. Identify complementary literals and see if mgu exists.

$$buys(B, ilp\_book)\theta = buys(A, ilp\_book)\theta$$

$$\theta = \{A/B\}$$

Step 3. Apply θ and form resolvent C.

1. Let $C_1\theta = h_1 \vee \sim l_1 \vee \sim l_2 \ldots \vee \sim l_j$
2. Let $C_2\theta = l_1 \vee \sim m_1 \vee \sim m_2 \ldots \vee \sim m_k$
3. Then $C = h_1 \vee \sim m_1 \vee \ldots \vee \sim m_k \vee \sim l_2 \ldots \vee \sim l_j$

Earlier example:

C: $likes(steve, B) \leftarrow sensible(B), rich(B)$

Resolution remains sound and refutation-complete with clausal logic (proof not required here)

Clauses as sets and resolution

Clauses are often represented as sets of literals

The clause
$likes(X, Y) \leftarrow vulcan(X), logical(Y)$
can be represented as the set
$\{likes(X, Y), \neg vulcan(X), \neg logical(Y)\}$

Applying a substitution to a clause yields an instance of the clause

Let $C =$
$\{likes(X, Y), \neg vulcan(X), \neg logical(Y)\}$
and $\theta = \{X/spock, Y/data\}$.

$C\theta =$

Resolving a pair of clauses requires a substitution that unifies a pair of complementary literals

Let $D = \{logical(A), \neg android(A)\}$ and $\theta = \{A/Y\}$

$D\theta =$

The resolvent of C, D is $E = \{likes(X, Y), \neg vulcan(X), \neg android(Y)\}$

E $= (C - \{l\})\theta \cup (D - \{m\})\theta = (C\theta - \{l\}\theta) \cup (D\theta - \{m\}\theta)$

where $l\theta = \neg m\theta$

Resolution and queries

Gven a program P, a query $q(X_1, X_2, \ldots, X_n)$? actually asks

- Are there any $X_1 X_2 \ldots X_n$ s.t. $q(X_1, X_2, \ldots, X_n)$ is true
- That is, are there any $X_1 X_2 \ldots X_n$ s.t. $\exists X_1 X_2 \ldots X_n q(X_1, X_2, \ldots, X_n)$ is a logical consequence of P
- That is, (using the deduction theorem) $P \cup \{\sim \exists X_1 X_2 \ldots X_n q(X_1, X_2, \ldots, X_n)\} \models \Box$
- Or $P \cup \{\leftarrow q(X_1, X_2, \ldots, X_n)\} \models \Box$
- Or, since resolution is sound and refutation complete $P \cup \{\leftarrow q(X_1, X_2, \ldots, X_n)\} \vdash \Box$

To see if there are variables $X_1 \ldots X_n$ for which the answer to $q(X_1, X_2, \ldots, X_n)$ is "yes":

1. Add query as a headless clause to P
2. See if $\Box$ can be derived using resolution

3. If $\Box$ can be derived, collect all substitutions for $X_1 \ldots X_n$ in the derivation of $\Box$

This still leaves open the proof strategy to be used to derive $\Box$. Most logic programming systems use a strategy called **SLD**

Selected Linear resolution for Definite clauses

Gven a program P, a query Q
$q(\ldots), r(\ldots), \ldots ?$

1. Select a literal l_i in Q using some *computation rule*.

2. Select a clause C_i from P that can resolve with the selected literal. If no C_i is possible $FAIL$

3. Construct resolvent C using C_i and $\leftarrow l_i$ as parents

4. If $C = \square$ $STOP$ otherwise $Q = C$, Goto Step 1

Resolution remains sound and refutation complete with this strategy (proof not required here)

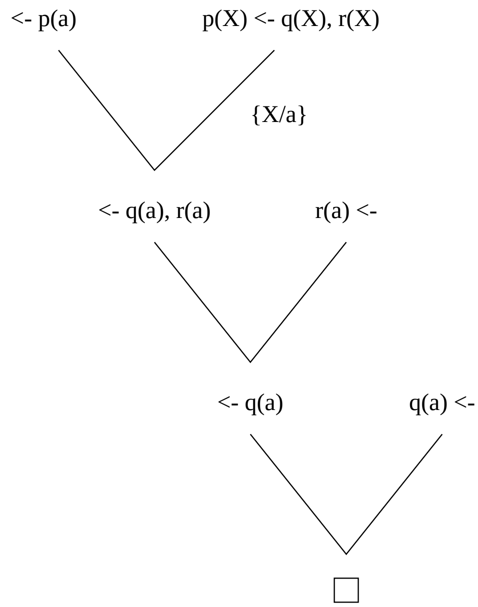
Here is an example of SLD resolution

P:

```
p(X) <- q(X), r(X)
q(a) <-
r(a) <-
```

Q:

p(a)?



Usually, a SLD-resolution proof is shown as a search tree where each node in the tree is a resolvent. The root node is the query $\leftarrow q(\ldots)\ldots$? Such trees are called SLD-trees

– Search trees that we considered under

"computations and answers" were SLD-trees. Answer-substitutions for variables in the query are obtained by collecting up substitutions from root to $\Box$ in a SLD-tree

– Draw the SLD-tree for the previous program for the query $p(X)$?

Recall that besides the proof-strategy, a practical implementation also requires a method to search the SLD tree

– This could cause problems in finding a path from root to $\Box$ even if one existed in the tree

A drawback: evaluating term equality

The resolution procedure as described here has a limitation concerned with term evaluation

- Consider a function $sqr/1$ that accepts a natural number and returns its square

- The mgu algorithm cannot unify $p(sqr(2))$ and $p(2)$

Extensions are possible to overcome this

- Resolution with "paramodulation" performs term rewrites to achieve this

- But, logic programming systems use a special predicate that forces term evaluation

- Thus, $p(sqr(2))$ is usually written as $X\ is\ 2*2, p(X)$. $p(X)$ unifies with $p(4)$ after forced evaluation the value of X by the $is/2$ predicate