The syntax of first-order predicate calculus: terms

1. Every constant symbol is a term, e.g. a, b, c, *clyde*

2. Every variable is a term, e.g. X, Y, Z, *Animal*

3. For every function symbol F of arity n and any n terms $T_1, T_2, \ldots, T_n$ then

   $$F(T_1, T_2, \ldots, T_n)$$

   is a term, e.g. $f(a), g(b, c), g(f(a), c), g(f(f(a)), b)$

4. Nothing else is a term.

The syntax of first-order predicate calculus: atoms

- For any predicate symbol P of arity n and any n terms $T_1, T_2, \ldots T_n$ then

$$P(T_1, T_2, \ldots T_n)$$

is an **atom**, e.g.
$p(a)$, *elephant(clyde)*,
likes(clyde, gertie), $q(a, f(a))$

The syntax of first-order predicate calculus: well-formed formulae

1. Every atom is a wff.

2. For any variable X and any wff W, $(\forall X W)$ and $(\exists X W)$ are wffs (**quantified wffs**).

3. For any wffs W, W_1 and W_2, the following are also wffs: $(\neg W)$, $(W_1 \wedge W_2)$, $(W_1 \vee W_2)$, $(W_1 \rightarrow W_2)$ and $(W_1 \leftrightarrow W_2)$

4. No other string of symbols is a wff.

E.g.
elephant(clyde), $\neg$*likes(clyde, gertie)*,
elephant(clyde) $\vee\neg$*likes(clyde, gertie)*,
$\forall X(p(X) \rightarrow (q(b,c) \wedge q(f(a),X)))$

Unnecessary parentheses will be omitted.

The semantics of first-order predicate calculus

In predicate logic, the meaning of the lowest level expressions (constant symbols, function symbols and predicate symbols) will be *stipulated* by an *interpretation*. Then, the meaning of a wff will be determined by the meanings of its subexpressions (in this case, its terms and subwffs).

We assume that there is some set of entities (possibly an infinite set) which we refer to as the **universe of discourse**, and which we denote as $\mathcal{U}$.

It is obviously possible to define a large number of relations *on* $\mathcal{U}$.

Example: The Books World

- Entities: Here we have five entities (excluding the table): Down&Out, Zen, Cuckoo, Jude and Spire.

- Functions: Here we have such functions as the unary function that maps from a book to the book directly on top of it.

- Relations: Here we have such relations as the binary relation that tells us whether a pair of books is such that the second is *somewhere* above the first, and the unary relation that tells us which books have a clear top.

Interpretations

In predicate logic, an interpretation $\mathcal{I}$ makes the following mappings:

1. For every constant symbol C, $\mathcal{I}(C)$ is an entity from $\mathcal{U}$. (Different constant symbols need not denote different entities, and there may be unnamed entities.)

2. For every n-ary function symbol F, $\mathcal{I}(F)$ is an n-ary function on $\mathcal{U}$, i.e. one that maps n-tuples to an entity. (Different function symbols need not denote different functions, and there may be unnamed functions.)

3. For every n-ary predicate symbol P, $\mathcal{I}(P)$ is an n-ary relation on $\mathcal{U}$, i.e. one that holds of n-tuples. (Different predicate symbols need not denote different relations, and there may be unnamed relations.)

Interpretation: example

For constant symbols a, b, c, d, e, a unary function symbol f and the binary predicate symbols p and q, we give an interpretation $\mathcal{I}_1$:

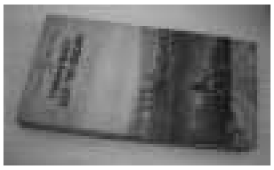
$\mathcal{I}_1(a)$ is 

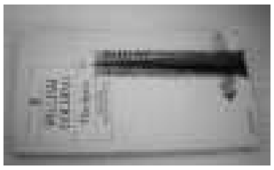
(Down&Out)

$\mathcal{I}_1(b)$ is 

(Zen)

$\mathcal{I}_1(c)$ is  (Cuckoo)

$\mathcal{I}_1(d)$ is  (Jude)

$\mathcal{I}_1(e)$ is  (Spire)

Interpretation: example, cont'd.

$\mathcal{I}_1(f)$ is	the unary function that maps a book to the book *directly on top* of it
$\mathcal{I}_1(p)$ is	the binary relation whose graph/extension contains pairs of books where the second book of the pair is *directly on top* of the first book of the pair
$\mathcal{I}_1(q)$ is	the binary relation whose graph/extension contains pairs of books where the second book of the pair is *somewhere above* the first book of the pair

Interpretation: example

Here is another example interpretation, $\mathcal{I}_2$, for the same universe of discourse and the same symbols. $\mathcal{I}_2$ is identical to $\mathcal{I}_1$ except:

$\mathcal{I}_2(a)$ is  (Zen)

$\mathcal{I}_2(b)$ is 

(Down&Out)

$\mathcal{I}_2(p)$ is the binary relation whose graph/extension contains pairs of books where the second book of the pair is *directly below* the first book of the pair

$\mathcal{I}_2(q)$ is the binary relation whose graph/extension contains pairs of books where the second book of the pair is *directly on top* of the first book of the pair

The semantics of first-order predicate calculus, cont'd.

$\mathcal{U}$ and $\mathcal{I}$ define a model M:

$$M = \langle \mathcal{U}, \mathcal{I} \rangle$$

The **semantic value** or **denotation** of some *term* or *wff* α with respect to model M is written

$$[\![\alpha]\!]^M$$

- If α is a constant, function symbol or predicate symbol, then

  $$[\![\alpha]\!]^M = \mathcal{I}(\alpha)$$

  (i.e. the meaning is stipulated by $\mathcal{I}$).

The semantics of first-order predicate calculus, cont'd.

- For any term comprising an n-ary function symbol F applied to terms $T_1, T_2, \ldots, T_n$, then

  $$\llbracket F(T_1, T_2, \ldots, T_n) \rrbracket^M = \\ \llbracket F \rrbracket^M(\llbracket T_1 \rrbracket^M, \llbracket T_2 \rrbracket^M, \ldots, \llbracket T_n \rrbracket^M)$$

  i.e. you get the function that corresponds to F (as stipulated by $\mathcal{I}$), you work out the objects that correspond to $T_1, T_2, \ldots, T_n$, and then you apply the function to these objects. This gets you an object, and this object is the meaning of the term $F(T_1, T_2, \ldots, T_n)$.

The semantics of first-order predicate calculus, cont'd.

- For any atom comprising an n-ary predicate symbol P applied to terms $T_1, T_2, \ldots, T_n$ then

  $$\begin{aligned}&[\![P(T_1, T_2, \ldots, T_n)]\!]^M = \text{ true iff}\\ &\qquad \langle [\![T_1]\!]^M, [\![T_2]\!]^M, \ldots, [\![T_n]\!]^M \rangle \in [\![P]\!]^M\end{aligned}$$

  I.e. you get the relation that corresponds to P (as stipulated by $\mathcal{I}$), you work out the objects that correspond to $T_1, T_2, \ldots, T_n$, and you see whether this n-tuple of objects is an element of the relation.

The semantics of first-order predicate calculus, cont'd

- For any quantified wff $\forall X W$, $[\![\forall X W]\!]^M$ is true iff $[\![W]\!]^M$ is true for all assignments of entities in $\mathcal{U}$ to X. Otherwise, it is false.

- For any quantified wff $\exists X W$, $[\![\exists X W]\!]^M$ is true iff $[\![W]\!]^M$ is true for at least one assignment of an entity in $\mathcal{U}$ to X. Otherwise, it is false.

- For any compound wff $\neg W, W_1 \wedge W_2, W_1 \vee W_2$, $W_1 \rightarrow W_2$ and $W_1 \leftrightarrow W_2$, determine $[\![W]\!]^M$, $[\![W_1]\!]^M$ and $[\![W_2]\!]^M$ and use these truth values as in propositional logic to determine the truth value of the compound wff.

The semantics of first-order calculus: making it more formal

- We introduce a **value assignment function**, g. Then compute the semantic value or denotation of α w.r.t. both M and g:

$$[\![\alpha]\!]^{M,g}$$

- Gives a more formal assignment of entities to variables:
  - allows free variables
  - makes definitions more properly recursive.

Metatheoretic definitions

- A wff α is **satisfiable** iff
  there exists some model M such that $[\![\alpha]\!]^M$ is true.

- E.g.

$$p(a, b)$$

$$p(b, a)$$

$$\forall X(q(X) \rightarrow r(X))$$

Metatheoretic definitions continued

- A wff α is **unsatisfiable** iff
  there exists no model M such that $[\![\alpha]\!]^M$ is true.

- E.g.

$$p(a, b) \wedge \neg p(a, b)$$

$$\neg(p(b, a) \rightarrow p(b, a))$$

Metatheoretic definitions continued

- A wff α is **valid** iff
  for every model M, $[\![\alpha]\!]^M$ is true.

- E.g.

$$p(a,b) \vee \neg p(a,b)$$

$$p(b,a) \rightarrow p(b,a)$$

Metatheoretic definitions continued

- A set of wffs Φ has wff α as a **logical consequence** or **entailment** iff
  every model that makes Φ true also makes α true.

- We write $\Phi \models \alpha$

- E.g. $\{\forall X(p(X) \rightarrow q(X)), p(a)\} \models q(a)$

- N.B. A semantic notion. Requires checking *all* models.

Proof theories for first-order predicate calculus

- Enables us to *derive/ deduce/ prove* that a conclusion α follows from a set of premisses Φ by *syntactic manipulation alone*.

- We write $\Phi \vdash \alpha$

- A proof theory comprises
  - a set of *inference rules*
  - a set of *logical axiom schemata*

- You supply Φ, the *premisses* (*initial axioms/ proper axioms*).

Formal proof

- A **proof** that $\Phi \vdash \alpha$ is a sequence of wffs such that each wff is either
  - a member of Φ
  - a logical axiom
    (produced by substitution into a logical axiom schemata)
  - the result of applying an inference rule

  and the final wff is α.

A (Hilbert-style) proof theory for propositional logic

- Logical axiom schemata:

  Schemata 1: $(\neg\alpha \to \alpha) \to \alpha$
  Schemata 2: $\alpha \to (\neg\alpha \to \beta)$
  Schemata 3: $(\alpha \to \beta) \to ((\beta \to \gamma) \to (\alpha \to \gamma))$

- One inference rule: $\frac{\vdash\alpha \quad \vdash\alpha\to\beta}{\vdash\beta}$ (Modus ponens)

- Proper axioms, e.g.

  1. *rain* $\to$ *wet*

  2. *wet* $\to$ *cold*

  Can prove *rain* $\to$ *cold*

- Finding a proof is a search process. The search space, particularly in this kind of proof theory, can be huge.

Soundness and completeness

- Does the syntactic manipulation give results that correspond to the logical consequences?

- **Soundness**
  A proof theory is sound if

$$\Phi \vdash \ \text{implies}\ \Phi \models \alpha$$

- **Completeness**
  A proof theory is complete if

$$\Phi \models \alpha\ \text{implies}\ \Phi \vdash \alpha$$

- The given (Hilbert-style) proof theory is sound and complete for proper axioms that use only $\rightarrow$ and $\neg$.

Clausal form logic

- A canonical/ normal/ standard form for wffs.
- As expressive as first-order predicate calculus.
- But FOPC and corresponding CFL are not logically equivalent
  - preserves (un)satisfiability
  - our proof theory is based on showing inconsistency (unsatisfiability)
- For CFL, there is a sound and complete proof theory
  - only one inference rule (resolution)
  - no logical axiom schemata

Conversion from FOPC to CFL

1. Relabel clashing variables.

2. Eliminate $\rightarrow$ and $\leftrightarrow$.

3. Move $\neg$ inwards.

4. Skolemise existentially quantified variables.

5. Drop universal quantifiers.

6. Distribute disjunction over conjunction.

7. Rewrite as clauses.

8. Standardise variables apart.

— *in that order*.

1. Relabel clashing variables

$$\forall X \forall Y \neg (p(X,Y) \rightarrow \forall Y q(Y,Y))$$

becomes

$$\forall X \forall Y \neg (p(X,Y) \rightarrow \forall Z q(Z,Z))$$

(Better still, avoid clashing variables in the first place.)

2. Eliminate $\rightarrow$ and $\leftrightarrow$

$$\begin{array}{rcl}(\alpha \rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) \\ (\alpha \leftrightarrow \beta) & \equiv & (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)\end{array}$$

$$\forall X \forall Y \neg (p(X,Y) \rightarrow \forall Z q(Z,Z))$$

becomes

$$\forall X \forall Y \neg (\neg p(X,Y) \vee \forall Z q(Z,Z))$$

3. Move $\neg$ 'inwards' until the only expressions being negated are atoms.

$$\begin{array}{rcl} \neg\forall X\alpha & \equiv & \exists X\neg\alpha \\ \neg\exists X\alpha & \equiv & \forall X\neg\alpha \\ \neg(\alpha\vee\beta) & \equiv & (\neg\alpha\wedge\neg\beta) \\ \neg(\alpha\wedge\beta) & \equiv & (\neg\alpha\vee\neg\beta) \\ \neg\neg\alpha & \equiv & \alpha \end{array}$$

$$\forall X\forall Y\neg(\neg p(X,Y)\vee\forall Z q(Z,Z))$$

becomes

$$\forall X\forall Y(\neg\neg p(X,Y)\wedge\neg\forall Z q(Z,Z))$$

which then becomes

$$\forall X\forall Y(p(X,Y)\wedge\exists Z\neg q(Z,Z))$$

4. Skolemise existentially quantified variables

Schematically,

a. $\exists Y \forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, Y)$
becomes
$\forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, c)$

b. $\forall X_1, \ldots, \forall X_n \exists Y \, p(X_1, \ldots, X_n, Y)$
becomes
$\forall X_1, \ldots, \forall X_n p(X_1, \ldots, X_n, f(X_1, \ldots, X_n))$

where c and f are new constants/function symbols

$$\forall X \forall Y (p(X, Y) \land \exists Z \neg q(Z, Z))$$

becomes

$$\forall X \forall Y (p(X, Y) \land \neg q(f(X, Y), f(X, Y)))$$

Another example

$$\exists W (p(W) \land \forall X \exists Y \exists Z q(W, X, Y, Z))$$

becomes

$$p(c) \land \forall X q(c, X, g(X), h(X))$$

5. Drop universal quantifiers

$$\forall X \forall Y (p(X, Y) \wedge \neg q(f(X, Y), f(X, Y)))$$

becomes

$$p(X, Y) \wedge \neg q(f(X, Y), f(X, Y))$$

6. Distribute disjunction over conjunction

—will end up in *conjunctive normal form*: a conjunction of disjunctions, i.e. something of the form

$$(P_1 \vee P_2 \vee \ldots \vee P_l) \wedge (Q_1 \vee Q_2 \vee \ldots \vee Q_m) \wedge \ldots$$
$$\ldots \wedge (R_1 \vee R_2 \vee \ldots \vee R_n)$$

where each P_i, Q_i and R_i is an atom or a negated atom.

$$p(X,Y) \wedge \neg q(f(X,Y), f(X,Y))$$

—no change

CNF continued

$$\begin{aligned}(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \\ ((\beta \wedge \gamma) \vee \alpha) &\equiv ((\beta \vee \alpha) \wedge (\gamma \vee \alpha))\end{aligned}$$

Also, associativity can be useful:

$$\begin{aligned}(\alpha \vee (\beta \vee \gamma)) &\equiv ((\alpha \vee \beta) \vee \gamma) \equiv \alpha \vee \beta \vee \gamma \\ (\alpha \wedge (\beta \wedge \gamma)) &\equiv ((\alpha \wedge \beta) \wedge \gamma) \equiv \alpha \wedge \beta \wedge \gamma\end{aligned}$$

$$p \vee (q \wedge r \wedge s \wedge (t \vee u))$$

becomes

$$(p \vee q) \wedge (p \vee r) \wedge (p \vee s) \wedge (p \vee t \vee u)$$

$$p \wedge (q \vee r \vee s \vee (t \wedge u))$$

becomes

$$p \wedge (q \vee r \vee s \vee t) \wedge (q \vee r \vee s \vee u)$$

7. Rewrite as clauses

Each conjunct is a clause.

$(P_1 \vee \ldots P_l) \wedge (Q_1 \vee \ldots \vee Q_m) \wedge (R_1 \vee \ldots \vee R_n)$

becomes

$$P_1 \vee \ldots \vee P_l$$

$$Q_1 \vee \ldots \vee Q_m$$

$$R_1 \vee \ldots \vee R_n$$

$$p(X, Y) \wedge \neg q(f(X, Y), f(X, Y))$$

becomes

$$p(X, Y)$$

$$\neg q(f(X, Y), f(X, Y))$$

8. Standardise *variables* in different clauses apart.

$$p(X, Y)$$

$$\neg q(f(X, Y), f(X, Y))$$

become

$$p(X_1, Y_1)$$

$$\neg q(f(X_2, Y_2), f(X_2, X_2))$$

English	Every student uses some computer, but at least one (specific) computer is used by every student.
Key	$s(X)$ — X is a student $c(X)$ — X is a computer $u(X,Y)$ — X uses Y
FOPC	

1. **Relabel clashing variables**

2. **Eliminate $\rightarrow$ and $\leftrightarrow$**

3. **Move $\neg$ inwards**

4. **Skolemise existentially quantified variables**

5. Drop universal quantifiers

6. Distribute disjunction over conjunction (to give a conjunction of disjunctions)

7. Rewrite as clauses

8. Standardise variables apart

CFL

- A **literal** is an atom or the negation of an atom.
- A **clause** is therefore a disjunction of literals.
- Clauses can be thought of as *sets* of literals
  - so order of literals is irrelevant
  - and can remove duplicates
- The **empty clause** is written $\Box$.
- A **ground clause/ literal/ atom/ term/ wff** is one without variables.

Resolution

- Our only inference rule.

- Example

  1. $p_1 \vee p_2$

  2. $\neg p_1 \vee p_3$

  3. $p_2 \vee p_3$ (from resolution of 1 and 2)

  - (1) and (2) are the **parents**

  - p_1 and $\neg p_1$ are the **selected literals**

  - p_1 and $\neg p_1$ are a **complementary pair**

  - (3) is the **resolvent**

Resolution continued

- Depict in tree form:

- Soundness

  i.e. $\{p_1 \vee p_2, \neg p_1 \vee p_3\} \models p_2 \vee p_3$

p_1	p_2	p_3	$p_1 \vee p_2$	$\neg p_1 \vee p_3$	$p_2 \vee p_3$	
T	T	T	T	T	T	*
T	T	F	T	F	T	
T	F	T	T	T	T	*
T	F	F	T	F	F	
F	T	T	T	T	T	*
F	T	F	T	T	T	*
F	F	T	F	T	T	
F	F	F	F	T	F	

Some examples

- Clauses are sets, so duplicates can be removed from resolvents:

1.	$p \vee q$
2.	$\neg p \vee q$
3.	

- Modus ponens is a special case of resolution:

1.	p	1.
2.	$p \to q$	2.
3.	q	3.

More examples

- Modus tollens is a special case of resolution:

1.	$\neg q$	1.	
2.	$p \to q$	2.	
3.	$\neg p$	3.	

- Resolving two unit clauses gives the empty clause:

1. p
2. $\neg p$
3.

Examples requiring care

- Example 1
  1. $p \vee q$
  2. $\neg p \vee \neg q$
  3. (from resolution of 1 and 2)
  4. (from resolution of 1 and 2)

- Example 2
  1. $p \vee \neg q$
  2. $\neg p \vee q$
  3. (from resolution of 1 and 2)
  4. (from resolution of 1 and 2)

The empty clause is *not* a resolvent in either example. In each example, there are *two* resolvents, both simplifying to true.

Exercises

- Exercise 1
  1. $p \vee \neg q \vee r$
  2. $p \vee \neg r \vee q \vee s$
  3.
  4.

- Exercise 2
  1. $p \vee p$
  2. $\neg p \vee \neg p$
  3.

Bindings and substitutions

- A **binding**

$$V \mapsto e$$

  - variable V is bound to expression e
  - e may be a constant, another variable, or a function symbol with its arguments
    (but e cannot contain V, e.g. $X \mapsto f(X)$ is illegal)

- A **substitution**
  - a set of bindings
  - e.g. $\{X \mapsto a, Y \mapsto Z, Z \mapsto f(a, b)\}$

Applying a substitution

Replace variables in the expression by what they are bound to.

E.g. let e be $p(X, f(Y), b)$

θ	$e\theta$
$\{X \mapsto Z, Y \mapsto W\}$	
$\{Y \mapsto c\}$	
$\{X \mapsto g(Z), Y \mapsto d\}$	
$\{X \mapsto c, Y \mapsto d\}$	

Composing two substitutions

Apply the 2nd set to each expression in the 1st set, then add bindings from the 2nd set whose variables do not appear among the variables of the 1st set.

$\theta \quad \{Z \mapsto f(X, Y)\}$

$\tau \quad \{X \mapsto b, Y \mapsto c, W \mapsto d, Z \mapsto e\}$

$\theta\tau$

$\theta \quad \{X_1 \mapsto f(Y_1), X_2 \mapsto Y_2, X_3 \mapsto g(Y_1, Y_2)\}$

$\tau \quad \{Y_1 \mapsto a, Y_2 \mapsto Y_3\}$

$\theta\tau$

Unification

- Two expressions, e_1 and e_2, **unify** if there exists a substitution θ such that

$$e_1\theta = e_2\theta$$

then we say that θ is a unifier of e_1 and e_2.

- E.g.

$p(a, X, f(Y), b)$

$p(Z, Z, f(c), b)$

Most general unifier (mgu)

- One unifier

$$p(X, f(Y), b)\,\{X \mapsto c, Y \mapsto b\} \quad =$$

$$p(X, f(b), b)\,\{X \mapsto c, Y \mapsto b\} \quad =$$

- Another unifier

$$p(X, f(Y), b)\,\{Y \mapsto b\} \quad =$$

$$p(X, f(b), b)\,\{Y \mapsto b\} \quad =$$

- $\{Y \mapsto b\}$ is more general than $\{X \mapsto c, Y \mapsto b\}$

Resolution with variables

- Clauses can be resolved if there is some pair in the parents that is complementary when some mgu is applied to them.

- The mgu is also applied to the resolvent.

- E.g.
  1. $p(X) \vee q(X, Y)$
  2. $\neg p(a) \vee r(b, Z)$
  3.

Exercises

- Exercise 1
  1. $p(X) \vee q(X, Y)$
  2. $\neg p(f(Z)) \vee \neg r(f(a), b) \vee s(Z)$
  3.

- Exercise 2
  1. $p(X, a) \vee q(X, Y)$
  2. $\neg p(f(Z), Z) \vee r(Z)$
  3.

Resolution

- In general, let α, β and γ be clauses.
  Then, γ is a resolvent of α and β iff
  - there is a non-empty set of *positive* literals

    $$A \subseteq \alpha$$

    and a non-empty set of *negative* literals

    $$B \subseteq \beta$$

    such that A and the complement of the literals in B unify with some mgu θ;
  - $\gamma = ((\alpha - A) \cup (\beta - B))\theta$

- E.g.
  1. $p(U) \vee p(V)$
  2. $\neg p(X) \vee \neg p(Y)$
  3.

Resolution deduction

(1) English: All elephants are grey

FOPC:

(2) English: Clyde is an elephant

FOPC:

To show

(3) English: Clyde is grey

FOPC:

(1)

(2)

(3)

Resolution deduction is incomplete

- A case where $\Phi \models \alpha$ but $\Phi \not\vdash \alpha$

- $\{$*elephant(clyde)*, $\neg$*elephant(clyde)*$\} \models$ *mouse(martie)*

- Resolution deduction does not derive *mouse(martie)*

  1.

  2.

  3.

- For completeness, we would have to add some *logical axiom schemata* to our proof theory.

- Fortunately, there is an alternative.

Resolution refutation

- Instead of using resolution to show

  $$\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \vdash \beta$$

  we use resolution to show that

  $$\alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n \wedge \neg\beta$$

  is **inconsistent**.

  – proof by contradiction/refutation proof

  – i.e. try to derive $\Box$ from

    $$\{\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\beta\}$$

    If you succeed then

    $$\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \models \beta$$

    Otherwise

    $$\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \not\models \beta$$

- Resolution refutation on clausal form is complete.

Resolution refutation

1. Convert all the *premisses* to clausal form.

2. Negate the goal *and then* convert to clausal form.

3. Repeat until either a contradiction is found, no progress can be made or a predetermined amount of effort has been expended

    - Select two clauses (the parents)

    - Resolve them together

    - If one of the resolvents is the empty clause, then a contradiction has been found
      If not, standardise the variables apart in these new clauses and then add them to the set of clauses available to the procedure.

(Show the proof in the form of a **refutation tree**.)

1. Every elephant is grey.
2. Clyde is an elephant.
3. Is Clyde grey?

1. If one is in Paris, then one is not in Moscow.
2. Flopsy is in Paris.
3. Is Flopsy in Moscow?

Exercise

1. Everyone who saves money earns interest.
2. Show if there is no interest earned, then nobody saves money.

Use $s(X)$ for X saves money and $e(X)$ for X earns interest.

1. Flopsy is Clyde’s friend.
2. Who is Clyde’s friend?

1. For all X, Y and Z, if X is the father of Y and Z is the father of X, then Z is the grandfather of Y.
2. Everyone has a father.
3. For some X, who is the grandfather of X?

Search

- Which clauses should be the parents?

- E.g.

$$\neg s \vee \neg r \qquad q \vee \neg p \qquad p \qquad r \vee \neg q \qquad s$$

- For systematicity, an automated theorem-prover will have a **search rule** that specifies which parents to try next.

Search

- Within parents, which literals should be selected as the complementary pair?

- E.g.

$p(a) \vee p(b)$ $\qquad\qquad$ $q(X) \vee \neg p(X)$

- For systematicity, an automated theorem-prover will have a **selection rule** that specifies which literals to try next.

Search

- The **search space** — *all* possible resolvents.

- The **derivation graph** —the resolvents actually computed.

- The **(resolution) refutation tree** —the resolvents that derive □

—we want a control strategy which explores as little of the search space as possible (for efficiency) without losing completeness.

Search rules

- **Breadth-first resolution**
  (*level-saturation resolution*)
  - compute all resolvents, level by level
  - complete but grossly inefficient
- **Linear resolution**
  (*ancestry-filtered resolution*)
  - at least one parent is a member of the initial axioms or an ancestor of the other parent
  - complete and sometimes better than breadth-first

Search rules

- **Set of support resolution**
  - at least one parent is from the negated goal or its descendants
  - complete (if original clauses are consistent) and more efficient
- **Unit resolution**
  - at least one parent is a unit clause
  - not complete in general, can be very efficient
- **Input resolution**
  - at least one parent is a member of the initial clauses (inc. the goal)
  - not complete in general, can be very efficient

Horn clauses

- Many strategies that are not complete in general are complete if the initial clauses and goal satisfy certain properties — if they are **Horn clauses.**
- **Horn clauses** — clauses with *at most* one positive literal
- **Positive Horn clauses** — clauses with *exactly* one positive literal
- **Negative Horn clauses** — clauses with *no* positive literals
- Input resolution and unit resolution are complete for problems comprising only Horn clauses.

SLD-resolution

- an even more restrictive strategy

- with a very efficient implementation

- complete for problems in which
  - there is only one negative Horn clause (the goal)
  - all the others are positive Horn clauses (with the positive literal written *leftmost*)

- this is no loss to us:

  —any Turing-machine-computable function can be expressed in this form

SLD-resolution

- *Search rule*
  - *Linear/set-of-support*: one parent is the goal or one of its descendants (hence a, -ve Horn clause)
  - *Input*: the other parent is a member of the initial clauses (hence, a +ve Horn clause)

- *Selection rule*
  - the parents are resolved on their *leftmost* literals only
  - the literals in the resolvent preserve the order from the parent clauses with those from the +ve Horn clause preceding those from the -ve Horn clause.

Prolog clauses are Horn clauses

- Prolog rules and facts are positive Horn clauses.

$$\texttt{p :- q}_1\texttt{,...,q}_n\texttt{.} \quad \equiv \quad p \leftarrow (q_1 \wedge \ldots \wedge q_n)$$
$$\equiv \quad p \vee \neg q_1 \vee \ldots \vee \neg q_n$$

- Prolog queries are negative Horn clauses.

$$\texttt{?- p}_1\texttt{, ..., p}_n \quad \equiv \quad \neg(p_1 \wedge \ldots \wedge p_n)$$
$$\equiv \quad \neg p_1 \vee \ldots \vee \neg p_n$$

Pure Prolog

- Prolog — an SLD-resolution refutation theorem-prover on one -ve Horn clause (the goal) and a number of +ve Horn clauses (the program)

- But the search is **depth-first** —path-by-path (implemented by a **backtracking algorithm**)

  – one parent will be the most recently produced unexplored descendant of the goal

  – other parent is from initial axioms tried from top of Prolog database to bottom.

- Advantages: backtracking is space-efficient, with possible rapid homing-in on solutions

- Disadvantages: more susceptible to problems of infinite search spaces (hence incompleteness), so programmers have to think about ordering.

Prolog

- While resolution is sound, and resolution refutation is complete, and SLD-resolution is a complete control strategy for positive Horn clauses,
  - Prolog's implementation of resolution is not sound, due to its implementation of unification, which lacks the 'occurs check'
  - Prolog's control strategy is not complete due to it being depth-first
- In both cases, a sacrifice has been made in order to gain efficiency.

Program

```
loves(X, person_loved_by(X)).
```

Goal

```
?- loves(Y, Y).
```

Procedural and declarative semantics

- *Declarative semantics*
  - based on the model theory of the logic
  - e.g.

    ```
    p :- q, r.
    ```

    means the same as

    ```
    p :- r, q.
    ```

  - e.g.

    ```
    q :- a, b.
    q :- c, d.
    ```

    means the same as

    ```
    q :- c, d.
    q :- a, b.
    ```

  - useful for easy understanding and for proving properties of Prolog programs

Procedural and declarative semantics

- *Procedural semantics*
  - how the Prolog interpreter executes the program
  - e.g.
    `p :- q, r.`

    ‘to prove `p`, Prolog *first* proves `q`, *then* proves `r`
  - unfortunately, we must sometimes take this into account
    * to avoid infinite search
    * for efficiency
    * to hack certain tricks
    * to use some impurities

Ordering

The following have the same declarative semantics

```
anc1(A, D) :- par(A, D).
anc1(A, D) :- par(A, I), anc1(I, D).

anc2(A, D) :- par(A, I), anc2(I, D).
anc2(A, D) :- par(A, D).

anc3(A, D) :- par(A, D).
anc3(A, D) :- anc3(I, D), par(A, I).

anc4(A, D) :- anc4(I, D), par(A, I).
anc4(A, D) :- par(A, D).
```

but very different procedural semantics. Consider for fact `par(a, b)` the goal

?- $\texttt{anc}_n\texttt{(X, Y).}$

Ordering: summary

When writing a logic program, first think declaratively (any version of $\mathtt{anc}_n$ will do), and then think procedurally (re-order your program to avoid infinite computation and to achieve efficiency and/or polymodality).

Order of clauses

- Prolog tries them from top-to-bottom.
- Swapping their order swaps the order in which paths are explored.
  - This can affect the order in which solutions are found (which is rarely significant to us);
  - if some branch is infinite, you don't want this to be explored prior to branches that contain solutions (which is why we generally put base cases first).

Ordering: summary cont'd.

Order of subgoals in a rule

- Prolog treies them from left-to-right.

- Swapping their order changes the 'contents' of the derivation graph, i.e. different resolvents are computed (because the selected literals in SLD-resolution are the ones that come leftmost).

  – Where a rule is left-recursive, we get an infinite branch;

  – this can also affect efficiency: other things being equal, you want the most restrictive subgoal first (though which is most restrictive can be query-dependent).