Search

- Which clauses should be the parents?

- E.g.

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

- 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 $\Box$

—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

Restricting expressiveness

- CFL is as expressive as FOL and proof is more manageable, but there may still be a lot of search
  - the more efficient search strategies lose completeness
  - alternative is to restrict expressiveness
    - allow only a subset of CFL (some statements can no longer be represented)
    - i.e. initial clauses (premisses / knowledge base) and goal must satisfy some property
    - then may get completeness (*w.r.t. the less expressive logic*) even for the more efficient strategies.

Horn clauses

- Clauses with *at most* one positive literal.
- Loss of expressiveness
  - e.g. $\neg p \vee q \vee r$ disallowed
  - a severe loss for a general intelligent agent? some forms of uncertainty cannot be represented, e.g.
    colour(clyde, grey) $\vee$ *colour(clyde, brown)*
  - not a big loss in many applications? (see later)
- Resolution deduction remains incomplete (irrespective of search strategy)
- Resolution refutation is complete for all the search strategies we saw, i.e. even for input resolution and unit resolution

Restricting expressiveness more

- Positive Horn clauses
  — clauses with *exactly* one positive literal
- Negative Horn clauses
  — clauses with *no* positive literals
- Some even more efficient strategies are complete for problems in which
  - there is only one negative Horn clause (the goal)
  - the initial clauses (premisses / knowledge base) are all positive Horn clauses

Restricting expressiveness even more, cont'd

- Again, a severe loss for general intelligence, but may be OK in many applications (see later) — it allows us to encode *facts* and *rules*.

As disjunctions	As facts & rules
p_1	p_1
$\neg p_1 \vee p_2$	$p_1 \rightarrow p_2$
$\neg p_1 \vee \neg p_2 \vee p_3$	$(p_1 \wedge p_2) \rightarrow p_3$
$\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee p_4$	$(p_1 \wedge p_2 \wedge p_3) \rightarrow p_4$

As disjunctions	As a query/goal (implicitly negated)
$\neg p_1$	p_1
$\neg p_1 \vee \neg p_2$	$p_1 \wedge p_2$
$\neg p_1 \vee \neg p_2 \vee \neg p_3$	$p_1 \wedge p_2 \wedge p_3$

Restricting expressiveness even more, cont'd

- Much human knowledge can still be encoded
- *Taxonomic knowledge* (classes, subclasses, instances)

$$\textit{elephant(clyde)}$$

$$\forall x(\textit{elephant}(x) \rightarrow \textit{mammal}(x))$$

- *Causal knowledge*

$$\forall x(\textit{hasMeasles}(x) \rightarrow \textit{hasSpots}(x))$$

$$\forall x(\textit{standsInRain}(x) \rightarrow \textit{getsWet}(x))$$

- *Action knowledge*

$$\forall x \forall y \forall z \forall s \; ((\textit{on}(x,y,s) \wedge \textit{clear}(x,s) \wedge \textit{clear}(z,s)) \rightarrow \textit{on}(x,z,\textit{do}(\textit{move}(x,y,z),S)))$$

Restricting expressiveness even more, cont'd

- This level of expressiveness (and this fact and rule notation) is used
  - in Prolog and other logic programming languages (any Turing-machine-computable function can be expressed in this form)
  - in deductive databases
  - in so-called rule-based systems (e.g. `IF...THEN..` rules encoding causal knowledge)
- Two complete search strategies can be used
  - *backward resolution*: much used in all three of the above
  - *forward resolution*: as good as never used in ther first 2 above; but often used in the 3rd

Backw'd res, described for disjunction notation

- In +ve Horn clauses, the +ve literal must be written *leftmost*, e.g. $p_2 \vee \neg p_1$ OK; $\neg p_1 \vee p_2$ not OK
- 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

Backward resolution, described for rule notation

- The same:
  - one parent is goal or one of its descendants
  - other is an initial clause (a fact or a rule)
  - these two are resolved so that leftmost of goal resolves with rule consequent or fact
- You can see this as *backwards chaining* from consequents to antecedents
- Also called *goal-driven* or *hypothesis-driven* reasoning.

Backward resolution

- Also called SLD-resolution
- Still have a choice about search strategy
  — breadth-first, depth-first or other?
- Prolog goes depth-first
  — space-efficient, may home-in on solutions, but incomplete again
- In rule-based systems, we might use further knowledge (e.g. heuristics, certainty factors, probabilities, etc.) to make an informed choice.

Forward resolution, described for disjunction notation

- In +ve Horn clauses, the +ve literal must be written *rightmost*
  e.g. $\neg p_1 \vee p_2$ OK; $p_2 \vee \neg p_1$ not OK
- Search rule
  - choose any pair for parents
- Selection rule
  - again, resolve on *leftmost* literals only
  - the literals in the resolvent preserve the order from the parent

Forward resolution, described for rule notation

- The equivalent
  - any parents
  - resolved so that a literal in the antecedent of a rule or in a goal resolves with a fact (whether this be a fact from the initial clauses or one established by resolving away all antecedents in a rule)
- You can see this as *forwards chaining* from antecedents to facts
- Also called *data-driven* reasoning

Forward resolution

- This still leaves a choice
  - how to choose the two parents
  - breadth-first, depth-first, other?
  - the role of heuristics, etc. to make informed choices

Which is more efficient?

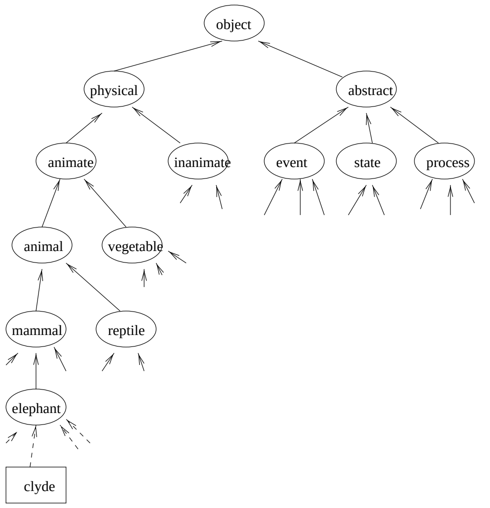
- Sometimes forward, sometimes backward, or even a mixture
  — depends on the 'shape' of search space (branching factor)

A different way to restrict expressiveness

- We can restrict the 'content' instead of (or as well as) the 'format'
  — if there is a certain kind of knowledge that you frequently encode and for which you want efficient inference.

- *Taxonomic knowledge* is one such example:
  - objects fall into classes/ categories/ collections/ kinds/ types/ concepts by virtue of sharing properties
  - much reasoning takes place at the level of classes
  - subclass relations organise classes into a class hierarchy/ taxonomy/ taxonomic hierarchy
    e.g. in biology, in medicine, in the Dewey Decimal system.

Taxonomy: simple example



One way to capture the simple taxonomy using FOPC

- Classes, e.g. `elephant`, are represented by unary predicates, e.g. *elephant*

- Individuals (instances/ objects), e.g. `clyde`, are represented by constants, e.g. *c*

Capturing in FOPC, cont'd

- Membership of a class, e.g.
  Clyde is an elephant: *elephant(c)*

- Subclasses, e.g.
  elephants are mammals
  $$\forall x(\textit{elephant}(x) \rightarrow \textit{mammal}(x))$$
  mammals are animals
  $$\forall x(\textit{mammal}(x) \rightarrow \textit{animal}(x))$$
  etc.

- By normal proof, from the three wffs, we can conclude:

  mammal(c)

  animal(c)

  etc.

  (And, BTW, all three are positive Horn clauses.)

Another way to encode using FOPC

- *Reification* turns properties, relationships, facts or other things in a logic's ontology into objects.

- Here we reify classes. Semantically, they stop being properties and become objects. Syntactically, therefore, we no longer use predicates, we use constants.

- Membership of a class, e.g.
  Clyde is an elephant

  instanceOf(c, elephant)

- Subclasses, e.g.
  elephants are mammals

  subclass(elephant, mammal)

  mammals are animals

  subclass(mammal, animal)

Reification, cont'd

- Now, from the three wffs, we can no longer prove that Clyde is a mammal and an animal. We need at least one more wff:
  $$\forall x \forall c_1 \forall c_2((\textit{instanceOf}(x, c_1) \wedge \textit{subclass}(c_1, c_2)) \rightarrow \textit{instanceOf}(x, c_2))$$
  Now, with these four, we can prove

  instanceOf(c, mammal)

  instanceOf(c, animal)

  To get *exactly* the same results as the previous encoding we also need:
  $$\forall c_1 \forall c_2 \forall c_3((\textit{subclass}(c_1, c_2) \wedge \textit{subclass}(c_2, c_3)) \rightarrow \textit{subclass}(c_1, c_3))$$
  (Again, all are positive Horn clauses.)

- The extra two wffs are the price of reification.

A special logic

- We could instead invent a special logic (knowledge representation language) tailored to representing and reasoning with taxonomic knowledge. We would want this to be *exactly* expressive enough, hoping that by restricting expressiveness in this way we will gain
  - conciseness and clarity in the syntax
  - the same semantics
  - more efficient proof

- Some people have developed so-called *description logics* or *terminological logics* for exactly this.

- But there's an even more popular logic (KRL):
  — **semantic networks** (semantic nets, associational networks/nets, conceptual graphs)

Semantic nets

- **Syntax**: a graphical syntax; ovals, rectangles, lines, dashed lines, labels
  — concise and clear

- **Semantics**: a sorry history

  – too often relied on imputation of meaning by the human reader rather than formal definition

  – in early days, failure to distinguish oval and rectangle (class and object), and line and dashed line (subclass and instance relationships), using only one symbol for each

  – nowadays, we recognise nets are just a logic

  – formal semantics can be given by specifying $\mathcal{U}$, $\mathcal{I}$, $[\![\ ]\!]$ or by showing how to translate nets into a logic that already has a formal semantics, which we have already done (twice).

Semantic nets, cont'd

- **Proof theory**:

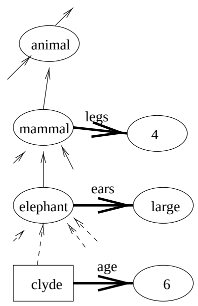
  – by storing the network as a node-and-pointer structure the limited kind of inference that we want can be done by special-purpose algorithms for following links
  — faster than using more general inference

  – but this needs care: should only traverse links in ways that correspond to what you'd get by traditional inference

  – example of doing otherwise: *spreading activation*
  — pass markers over the network, spreading out from activated notes, will retrieve you all 'related' classes and objects: not sound!

Extending the example: properties

- We might want to specify properties that hold for all instances of a class, e.g., mammals have 4 legs and elephants have large ears, and properties that hold for specific instances, e.g., Clyde is 6 years old.



Properties, cont'd

- What is the semantics? The semantics ought to make clear that these properties are instance properties, not class properties. Again we do this by showing equivalent FOPC. (And again we give two formulations).

- Version 1

$$\forall x(\textit{mammal}(x) \rightarrow \textit{legs}(x, 4))$$
$$\forall x(\textit{elephant}(x) \rightarrow \textit{ears}(x, \textit{large}))$$
$$\textit{age(clyde, 6)}$$

Reifying properties

- Version 2

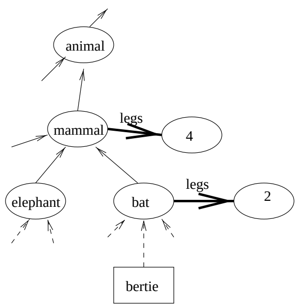
  valueOf(mammal, legs, 4)
  valueOf(elephant, ears, large)
  valueOf(clyde, age, 6)
  $\forall x \forall c \forall p \forall v(($*instanceOf*$(x, c) \wedge$
  valueOf$(c, p, v)) \rightarrow$ *valueOf*$(x, p, v))$
  $\forall c_1 \forall c_2 \forall p \forall v(($*subclass*$(c_1, c_2) \wedge$
  valueOf$(c_2, p, v)) \rightarrow$ *valueOf*$(c_1, p, v))$

  The standard logical semantics for these gives our network a semantics too.

- **Proof**: Clearly properties are *inherited* from superclasses:
  — again a procedural link-following algorithm will derive an instance's properties more quickly than general inference will.

More extensions: defaults and the over-riding of inheritance

- Example:



- The intention here is that bats being bipedal *over-rides* the inherited *default* that bats, being mammals, are quadrupeds.

- But we need a formal semantics.

Over-riding, cont'd

- The formal semantics is tricky. The treatment we've got so far, e.g.

  valueOf(mammal, legs, 4)
  valueOf(bat, legs, 2)
  subclass(bat, mammal)
  instanceOf(bertie, bat)
  $\forall x \forall c \forall p \forall v(($*instanceOf*$(x, c) \wedge$
  valueOf$(c, p, v)) \rightarrow$ *valueOf*$(x, p, v))$
  $\forall c_1 \forall c_2 \forall p \forall v(($*subclass*$(c_1, c_2) \wedge$
  valueOf$(c_2, p, v)) \rightarrow$ *valueOf*$(c_1, p, v))$

  is one that enable us to derive both:

  valueOf(bertie, legs, 2)

  valueOf(bertie, legs, 4)

  The problem is how to inhibit an inference.

Over-riding, cont'd

Inhibiting inferences has motivated new logics, e.g. *default logics*, *non-monotonic logics*. But it *can* be done in FOPC. We use the formulation where classes have been reified; the result is ugly:

valueOf(mammal, legs, 4)
exceptionTo(bat, mammal, legs)
valueOf(bat, legs, 2)
subclass(bat, mammal)
instanceOf(bertie, bat)
$\forall x \forall c \forall p \forall v(($*instanceOf*$(x, c) \wedge$ *valueOf*(c, p, v)
$\wedge \neg$*exceptionTo*$(x, c, p)) \rightarrow$ *valueOf*$(x, p, v))$
$\forall c_1 \forall c_2 \forall p \forall v(($*subclass*$(c_1, c_2) \wedge$ *valueOf*(c_2, p, v)
$\wedge \neg$*exceptionTo*$(c_1, c_2, p)) \rightarrow$ *valueOf*$(c_1, p, v))$
$\forall x \forall c_1 \forall c_2 \forall p(($*instanceOf*$(x, c_1) \wedge$
exceptionTo$(c_1, c_2, p)) \rightarrow$ *exceptionTo*$(x, c_2, p))$
$\forall c_1 \forall c_2 \forall c_3 \forall p(($*subclass*$(c_1, c_2) \wedge$
exceptionTo$(c_2, c_3, p)) \rightarrow$ *exceptionTo*$(c_1, c_3, p))$

Proof: Now we don't have Horn clauses so general inference will be even less efficient. But pointer-following will still be very efficient.

Frames: another alternative

- Frame languages simply give us yet another syntax. We'll look at this syntax. (The semantics and proof theory are so similar to that of nets that we will skip over them. What a disgrace.)

- In frame systems, classes and instances are shown as frames: boxes which have a *name*, a number of named *slots* and *fillers* or values in the slots.

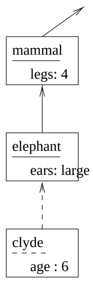
mammal		
legs	:	4
subclass	:	**animal**

elephant		
ears	:	large
subclass	:	**mammal**

clyde		
age	:	6
instanceOf	:	**elephant**

Frames, cont'd

- Sometimes where a slot has another frame as its value, graphical links are used instead: this brings in ideas from nets.



- Inasmuch as the encapsulation suggested by the frame language has an effect on storage structures (i.e. by ensuring properties are stored close to classes/instances, e.g. in contiguous record structures), inferencing efficiency may be further improved.

Procedural attachment

- Some systems allow slots to contain procedures: *procedural attachment*.

- As-needed: When a value for the slot is needed, the procedure is run (to compute the value).

- When-added: When a value is placed into a slot, the procedure is run (to check the value, e.g. consistency checks, or to propagate results to other slots).

Object-oriented representations

- Frames takes us closer to object-oriented approaches.

- E.g. if we also have some procedural knowledge that we want to represent, we could include methods. These can be inherited or over-ridden in the same way that properties are.

- Thus we get a meeting of AI and OO programming/OO databases.

More extensions

- There are many further extensions we can make:
  — class properties (such as population size)
  — multiple inheritance (especially what to do if there's conflict)

- Some have added negation, disjunction, conditionals and quantifiers —efficiency problems return.

- Others have represented knowledge other than taxonomic in this way, e.g. *conceptual dependency theory* is a semantic net which has nodes and arcs designed for representing descriptions of actions

- You can also mix a net/frame language with a traditional logic in a single system, using the net/frames for taxonomic stuff and the traditional logic for other assertions.