Theorem Proving using Resolution

Example:

(1) If it is hot(t) and humid(h), then it will rain(r).
(2) If it is humid(h) then it is hot(t)
(3) It is humid(h) now.

(?) Will it rain?(r)

Resolution

- works with clauses (disjunctions of predicates).
- makes a proof by contradiction
- uses the following inference rule:

$$\frac{Q \vee \sim R, S \vee R}{Q \vee S}$$

Theoretical Problems of Resolution

- The resolution procedure does not necessary halt, if a statement cannot be proven.
- However, the resolution procedure will find a proof, if a proof exists — however, some syntactical restrictions have to be imposed on theorem provers to guarantee this property*.
- Another problem is, how long it will take to find a proof. In general, it is hard to incorporate heuristic knowledge into resolution problem solvers, because they tend to work too low-level. Another related problem is, if this method is applied to more complex problems, to prevent that the theorem prover infers clauses that are trivial or completely irrelevant for the solution of the particular problem.

* which are quite ugly and might lead to an inefficient system performance due to using a breadth-first style search strategy.

Horn-Clause Resolution

Horn-clause: A clause that contains at most one positive literal.

Properties of Horn-Clause Resolution

- The implication-problem is decidable for Horn-clauses; that is, you can find an algorithm that finds a proof, if it exists, and which also terminates, when no proof can be found.
- However due to the syntacal restrictions in Horn-clauses, it is complicated to express knowledge that involves negations in Horn-clause resolution.
    - Therefore, systems that rely on Horn-clause resolution use negation as failure: *all assertions that neither are stored in the knowledge base nor can be inferred from the information stored in the knowledge base are assumed to be false."*
    - Negation as failure relies on the *closed world assumption* that might cause serious knowledge representation problems.
- PROLOG uses Horn-clause resolution relying on a depth-first, left-to-write control strategy, similar to backtracking.

Non-Finite Number of Clauses

When applying the resolution method to first order predicate calculus, the number of clause you can generate from a set of ground clauses is not necessary finite.

Consider the following example:

(A) Every human being loves another human being.

$\forall h(human(h) \longrightarrow \exists y(human(y) \wedge loves(x, y)))$

(B) Fred is a human being.

$human(Fred)$

We receive the following three clauses:

(1) $\sim human(\$h) \vee human(F(\$h))$

(2) $\sim human(\$h) \vee loves(\$h, F(\$h))$

(3) $human(Fred)$

(4) $human(F(Fred))$ using (3) and (1)

(5) $human(F(F(Fred))$ using (4) and (1)

... *continuing to resolve (1) with the last clause.*

(n+3) $human(F^n(Fred))$

Analysis of the Example

- The example shows that the number of clauses we can receive by resolving clauses is not finite, which explains somehow why the resolution algorithm not necessary terminates when no proof exists for the particular case — the algorithm might generate clauses through the end of its life.
- It also shows that a theorem prover has to take special precautions to prevent the above situation in the case that a proof does exist.
- Classical search techniques such as backtracking, graph-search, and hill-climbing(?) can be applied to the search process of finding the empty clause.

Classical Reasoning Systems

- They are based on classical logic.
- They are complete. All facts that are necessary to solve a particular problem are present in the system, either they are stored or they can be derived.
- They are consistent. The application of inference rules will never lead to a contradiction.
- They are monotonic. The only way of change tolerated by these systems is the addition of new facts; no other changes such as modifications and retractions are tolerated.
- They rely on two-valued logic.

Nonmonotonic Systems

Nonmonotonic system intend to solve problems for which the first three of the above assumptions are violated. Namely, problems that require

- inferences that are based on the lack of knowledge.
- inferences in a changing environment.
- inferences in the presence of inconsistencies.

Nonmonotic systems still rely on two-valued logic.

Nonmonotonic Logic

M-operator: M P evaluates to true if it is possible to believe P; that is, if the assumption of P does not lead to a contradiction with other facts and/or other assumptions.

More specifically, inference rules in NML look as follows:

$A \wedge M\ B \rightarrow C$

Semantics: If A is true and it possible to assume B then infer C, if the assumption of C does not lead to a contractiction.

Example:

$\forall x(Republican(x) \wedge M \sim Pacifist(x) \rightarrow \sim Pacifist(x))$

$\forall x(Quaker(x) \wedge M\ Pacifist(x) \rightarrow Pacifist(x))$

Problems of Nonmonotonic Logic

- does not resolve conflicts in one way or the other; does not support the notion of strength of assumptions and or inferences.
- inferencing relies on inferring the absense of contradictions, a problem which in general is not decidable for (full) first order predicate calculus.
- needs a powerful theorem prover.

Default Logic

$$\frac{A : B}{C}$$

Semantics: If A is true and it is possible to assume B, then infer C.

Remark: In default logic inference rules compute a set of plausible extensions of a knowledge base. Each extension corresponds to one maximal augmentation of the knowledge base. DL does not make any statement which of the extensions should be chosen in the case that the above set contains more than one member.

$$\frac{Republican(\$x) : \sim Pacifist(\$x)}{\sim Pacifist(\$x)}$$

$$\frac{Quaker(\$x) : \ Pacifist(\$x)}{Pacifist(\$x)}$$

Inheritance

idea: An object inherits the attribute values of all classes of which it is member unless doing so leads to a contradiction, in which case the value of the more restrictive class takes preference over the value of the broader class.

Remarks:

- Inheritance can be provided by using rules of DL; although this does not lead to elegant solutions — because the lack of preference rules in DL.
- Inheritance is supported by object-oriented programming languages, frame-cased systems, and by advanced data models for data- and knowledge bases.

Problems with the Closed-World Assumption

- ignorance ("I don't know (now).") cannot be expressed.
- its results depend on the form in which assertions are expressed; e.g. if the knowledge base contains $A \vee B$ we receive a contradiction.

Dependency-Directed Backtracking

Context: Constraint-satisfaction problems.

Idea: Revise/change only those assumptions (operator applications) that have been responsible for the particular failure (constraint violation) in the backtracking process. That is the algorithm might backtrack skipping several levels. Otherwise, the algorithm works like regular backtracking.

Preconditions: all inferences have to be stored explicitly in an *inference network* in order to be able to decide which decisions have been responsible for a particular failure. In other words, justifications why something is believed are stored explicitly in a network.