Reasoning Systems

Rule-Based Programming Languages

- Both forward and backward chaining with rules form the basis of programming languages.
- Prolog (PROgramming in LOGic) represents programs as logical Horn clauses and treats execution as answering queries with backward chaining.
- Production system languages (OPS5, CLIPS) represent programs as rules that add and/or delete elements from working memory and treat execution as forward chaining inference.

Prolog

- Prolog programs are stated as Horn clauses (facts and rules)

```
member(X, [X | L]).
member(X, [Y | L]) :- member(X, L).

append([ ], L, L).
append([X | L1], L2, [X | L3]) :- append(L1, L2, L3).
```

- Programs are executed by making queries.

```
? member(a [a,b,c])
  Yes.

? append([a,b], [c,d], X)
X = [a,b,c,d].
```

- Queries can generate “output” from “input”

```
? member(X, [a,b,c])
  a;
  b;
  c;
  No.
```


Prolog (cont)

- More query examples:

```
? append(X, [c], [a,b,c])
   X=[a,b].

? append(X, Y, [a,b])
  X=[],
  Y=[a,b];

  X=[a],
  Y=[b];

  X=[a,b],
  Y=[];

  No.

? member(a, X)
   [a | Z1];
   [Z2, a | Z3];
   [Z4, Z5, a | Z6];
   ....
```

Prolog Search

- Prolog uses depth-first search, pursuing conjuncts in the body of a clause in left to right order.

- Not guaranteed to terminate:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).
parent(Tom, John).

? ancestor(X, John)
X=Tom;
.....
```

- Programs must be written carefully to guarantee efficiency and termination, as in any other programming language.

Negation as Failure

- Since it uses Horn clause inference, Prolog cannot handle true negation.

- However, it does include negation as failure, not(P), which is assumed to be true unless P can be proven.

```
sibling(X,Y) :- parent(P,X), parent(P,Y), not(X=Y).

bachelor(X) :- male(X), adult(X), not(married(X,Y)).
married(X,Y) :- husband(X,Y).
married(X,Y) :- husband(Y,X).
married(X,Y) :- wife(X,Y).
married(X,Y) :- wife(Y,X).
```

- Unless all relevant knowledge is in the KB (**closed world assumption**, CWA), this type of inference is unsound.

  Not proving P is not the same as proving ¬P!

```
?sibling(mark-twain,samuel-clemens)
Yes.
```


Production Systems

- Forward chaining systems used to construct many expert systems and as a model of human cognition.

- Basis of several rule-based programming languages such as OPS5 and CLIPS.

- Maintains a **working memory** of positive ground literals (facts)

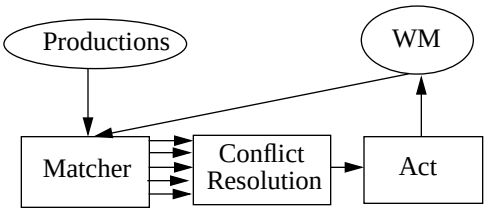
- Maintains a **production memory** or **rule memory** of rules of the form:

  $p_1 \wedge p_2 .... \wedge p_n \Rightarrow act_1 \wedge act_2 .... \wedge act_m$

  where p_i are positive literals and act_i are actions that can add or delete elements from working memory (and perhaps perform I/O)

Production System Execution

Until no more rules fire do
Match: Find all instantiations (variable bindings) of rules whose conditions match working memory
Conflict Resolution: Pick one of these rules to actually fire
Act: Execute the instantiated actions for this rule



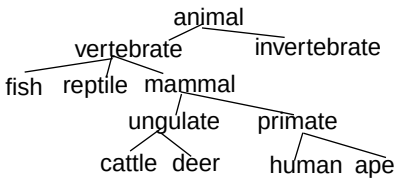
Production System Phases

- **Match**: Repeatedly attempting to match all rules every time is too inefficient. Better to maintain a list of currently "active" rules and update it each time working memory is changed.
  - Rete net is a standard approach.
- **Conflict Resolution**: Pick a rule to fire based on:
  - **No duplication**: Don't fire the same rule instantiation twice.
  - **Recency**: Prefer rules whose conditions rely on recently created elements of working memory.
  - **Specificity**: Prefer rules with more specific conditions

    sneezing $\Rightarrow$ cold
    sneezing $\wedge$ itching $\Rightarrow$ allergies

Semantic Networks

- Use graphs to represent concepts and the relations between them.
- Simplest networks are **ISA heirarchies**



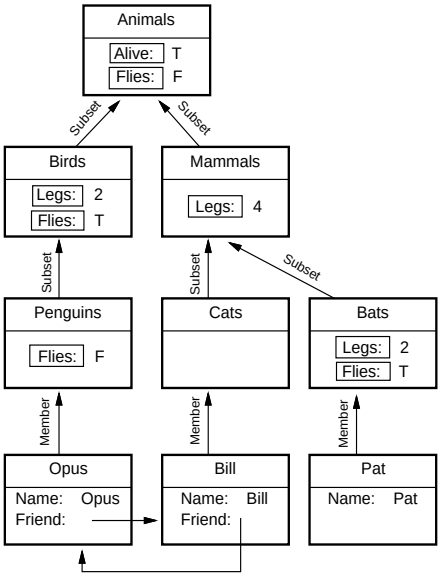
- Must be careful to make a **type/token distinction**

Bevo isa Cattle	Cattle(Bevo)
Cattle isa Ungulate	$\forall x$ (Cattle(x) $\Rightarrow$ Ungulate(x))

- Restricted shorthand for a logical representation.

Semantic Nets / Frames

- Labelled links can represent arbitrary relations between objects and/or concepts.
- Nodes with links can also be viewed as **frames** with **slots** that point to other objects and/or concepts.



Rel(Alive,Animals,T)
Rel(Flies,Animals,F)

Birds $\subset$ Animals
Mammals $\subset$ Animals

Rel(Flies,Birds,T)
Rel(Legs,Birds,2)
Rel(Legs,Mammals,4)

Penguins $\subset$ Birds
Cats $\subset$ Mammals
Bats $\subset$ Mammals
Rel(Flies,Penguins,F)
Rel(Legs,Bats,2)
Rel(Flies,Bats,T)

Opus $\in$ Penguins
Bill $\in$ Cats
Pat $\in$ Bats

Name(Opus,"Opus")
Name(Bill,"Bill")
Friend(Opus,Bill)
Friend(Bill,Opus)
Name(Pat,"Pat")

(a) A frame−based knowledge base **(b) Translation into first−or**

Inheritance

- Inheritance is a specific type of inference that allows properties of objects to be inferred from properties of categories to which the object belongs.

  Is Bill alive?

  Yes, since Bill is a cat, cats are mammals, mammals are animals, and animals are alive.
- Such inference can be performed by a simple graph traversal algorithm and implemented very efficiently.
- However, it is basically a form of logical inference

  $\forall x$ (Cat(x) $\Rightarrow$ Mammal(x))
  $\forall x$ (Mammal(x) $\Rightarrow$ Animal(x))
  $\forall x$ (Animal(x) $\Rightarrow$ Alive(x))
  Cat(Bill)

  $\vdash$ Alive(Bill)

Backward or Forward?

- Backward reasoning is more goal directed and can therefore be more efficient at answering specific queries.
- However, it can be very inefficient for some inferences like inheritance.

  ?Alive(Bill)
  Animal(Bill)? Bird(Bill)? Penguin(Bill)? Robin(Bill)? Grackle(Bill),...Mammal(Bill)?, Ungulate(Bill)?.....
- In this case, forward reasoning is more efficient but still not directed towards a particular goal.

  Cat(Bill) $\Rightarrow$ Mammal(Bill) $\Rightarrow$ Animal(Bill) $\Rightarrow$ Alive(Bill)
- Which is more efficient depends on whether the forward or backward branching factor is worse.
- Inheritance methods allow goal-directed efficient reasoning for a specific, restricted type of inference.

Semantics of Links

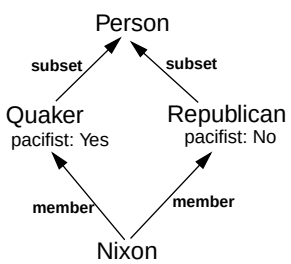
- Must be careful to distinguish different types of links.
- Links between tokens and tokens are different than links between types and types and links between tokens and types.

Link Type	Semantics	Example
$A \xrightarrow{Subset} B$	$A \subset B$	$Cats \subset Mammals$
$A \xrightarrow{Member} B$	$A \in B$	$Bill \in Cats$
$A \xrightarrow{R} B$	$R(A, B)$	$Bill \xrightarrow{Age} 12$
$A \xrightarrow{\boxed{R}} B$	$\forall x \; x \in A \Rightarrow R(x, B)$	$Birds \xrightarrow{\boxed{Legs}} 2$
$A \xrightarrow{\boxed{R}} B$	$\forall x \; \exists y \; x \in A \Rightarrow y \in B \wedge R(x, y)$	$Birds \xrightarrow{\boxed{Parent}} Birds$

Inheritance with Exceptions and Multiple Inheritance

- Information specified for a type gives the **default value** for a relation, but this may be over-ridden by a more specific type.
- Tweety is a bird. Does Tweety fly?
  Birds fly. Yes.

  Opus is a penguin. Does Opus fly?
  Penguin's don't fly. No.
- If hierarchy is not a tree but a directed acyclic graph (DAG) then different inheritance paths may result in different defaults being inherited.



Nonmonotonicity

- In normal **monotonic** logic, adding more sentences to a KB only entails more conclusions.

  if KB |– P then KB $\cup$ {S} |– P
- Inheritance with exceptions is not monotonic (it is **nonmonotonic**)

  Bird(Opus)
  Fly(Opus)? yes

  Penguin(Opus)
  Fly(Opus)? no
- Nonmonotonic logics attempt to formalize such reasoning by allow **default rules** of the form:

  If P and concluding Q is consistent, then conclude Q.

  If Bird(X) then if consistent Fly(x)

Defaults with Negation as Failure

- Prolog negation as failure can be used to implement default inference.

- fly(X) :- bird(X), not(ab(X)).
  ab(X) :- penguin(X).
  ab(X) :- ostrich(X).

  bird(opus).
  ? fly(opus)
  Yes.

  penguin(opus)
  ? fly(opus)
  No.