1943–1956 The Beginnings of AI

1943 McCulloch and Pitts show networks of neurons can compute and learn any function

1950 Shannon (1950) and Turing (1953) wrote chess programs

1951 Minsky and Edmonds build the first neural network computer (SNARC)

1956 Dartmouth Conference – Newell and Simon brought a reasoning program The Logic Theorist which proved theorems.

1952–1969 The Good Years

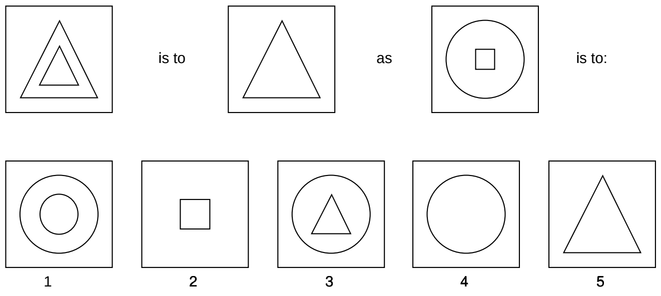
1952 Samuel's checkers player

1958 McCarthy designed **LISP**, helped invent time-sharing, and created Advice Taker (a domain independent reasoning system)

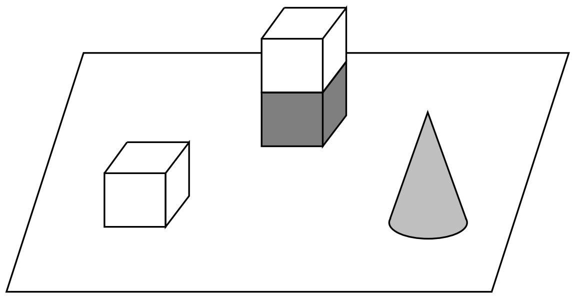
1960's Microworlds – solving limited problems: SAINT (1963), ANALOGY (1968) STUDENT (1967), blocksworld invented.

1962 Perceptron Convergence Theorem is proved.

Example ANALOGY Problem



Blocksworld



History of AI
1966–1974 A Dose of Reality

- Herb Simon (1957)
- Machine translation
- Knowledge-poor programs
- Intractable problems, lack of computing power (Lighthill Report, 1973)
- Limitations in knowledge representation (Minsky and Papert,1969)

Knowledge Representation

- Human intelligence relies on a lot of background knowledge (the more you know, the easier many tasks become / **"knowledge is power"**)
- E.g. SEND + MORE = MONEY puzzle.
- Natural language understanding
  - — Time flies like an arrow.
  - — Fruit flies like bananas.
  - — The spirit is willing but the flesh is weak. (English)
  - — The vodka is good but the meat is rotten. (Russian)

OR: Plan a trip to L.A.

- Q. How did we encode (domain) knowledge so far?
  Search knowledge?

Fine for limited amounts of knowledge / well-defined domains.

Otherwise: **knowledge-based systems approach**.

Knowledge-Based Systems / Programs

General idea:

- represent knowledge in declarative statements
- use inference / reasoning mechanism to derive new information or make decisions

Natural candidate: logical language (propositional / first-order) combined with a logical inference mechanism

How close to human thought?

In any case, appears reasonable strategy for machines.

“Advice-Taker”

- John McCarthy, 1958: “Programs with Common Sense”
- Useful to impart knowledge to a program in the form of declarative (logical) statements (“what” instead of “how”).
- Useful to represent how an agent’s actions affect the world and to employ general reasoning mechanisms to process and act on this information.

E.g. Formalize *“x is at y”* using predicate *at*, i.e., *at(x,y)* *at* **defined** by its properties, e.g., $at(x,y) \wedge at(y,z) \rightarrow at(x,z)$

Agent / Intelligent System Design

Kenneth Craik (1943) *The Nature of Explanation*

> If the organism carries a “small-scale model” of external reality and of its own small possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of the past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to the emergencies which face it.

Alternative view: against representations — Brooks (1989)

Representation Language

preferably:

— expressive and concise

— unambiguous and independent of context

— have an effective procedure to derive new information

not easy to meet these goals . . .

propositional and first-order logic meet some of the criteria

Logical Representation

Three components:

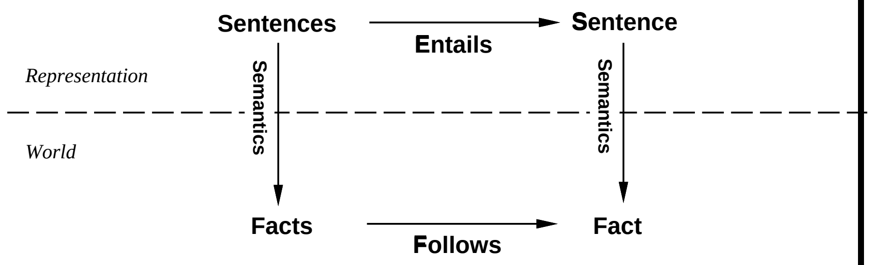
syntax

semantics (link to the world)

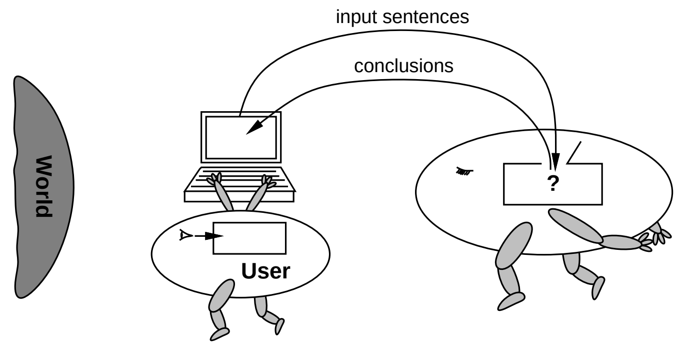
proof theory (“pushing symbols”)

To make it work: **soundness** and **completeness**.

Connecting Sentences to the World



Tenuous Link to Real World



All computer has are sentences (hopefully about the world).

Sensors can provide some grounding.

More Concrete: Propositional Logic

Syntax: build sentences from atomic propositions, using connectives $\land, \lor, \neg, \Rightarrow, \Leftrightarrow$.
(and / or / not / implies / equivalence (biconditional))

E.g.: $((\neg P) \lor (Q \land R)) \Rightarrow S$

p. 167 R&N.

Semantics

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Note: $\Rightarrow$ somewhat counterintuitive.
What's the truth value of "5 is even implies Sam is smart"?

Validity and Inference

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

Truth table for: *Premises* $\Rightarrow$ *Conclusion.*

Shows $((P \vee H) \wedge (\neg H)) \Rightarrow P$ is valid
(True in all interpretations)
We write $\models ((P \vee H) \wedge (\neg H)) \Rightarrow P$

Proof Theory

Purely syntactic rules for deriving the logical consequences of a set of sentences.

We write: $KB \vdash \alpha$, i.e., α can be **deduced** from KB or α is **provable** from KB.

If inference procedure is sound and complete, computer can ignore semantics and "just push symbols"!

First-Order Logic as a Knowledge Representation

Formalism for storing knowledge as:

propositions: "It is raining" becomes RAINING

operators: $\vee$, $\wedge$, $\neg$ or $\sim$, $=$, $\rightarrow$

predicates: Man(SOCRATES) for "Socrates is a man."

quantifiers:

All men are mortal.

$$\forall x : Man(x) \rightarrow Mortal(x)$$

Some man is mortal.

$$\exists x : Man(x) \rightarrow Mortal(x)$$

Reasoning Methods: Rules of Inference

1. Modus Ponens:

Assume:	$P \rightarrow Q$	If raining, then soggy courts.
and	P	It is raining.
Then:	Q	Soggy Courts.

2. Modus Tollens:

Assume:	$P \rightarrow Q$	If raining, then soggy courts.
and	$\neg Q$	No soggy courts.
Then:	$\neg P$	It is not raining.

Representing Facts

1. Pingali is a CS professor.
2. All CS professors are ENG professors.

3. Hopcroft is the dean.
4. All ENG professors are a friend of the dean or don't know him.
5. Everyone is a friend of someone.
6. People only criticize deans they are not friends of.

7. Pingali criticized Hopcroft.

Representing Subset Hierarchies

Member:

$$CSPROF(Pingali)$$

or

$$member(Pingali, CSPROF)$$

subset:

$$\forall x : CSPROF(x) \rightarrow ENGPROF(x)$$

or

$$isa(CSPROF, ENGPROF)$$

Is Pingali a friend of Hopcroft?

$\neg$ friend-of (Pingali, Hopcroft)

Forward Chaining

Given a fact p to be added to the KB,

1. Find all implications I that have p as a premise
2. For each i in I, if the other premises in i are already known to hold
   (a) Add the consequent in i to the KB

Continue until no more facts can be inferred.

Backward Chaining

Given a fact q to be "proven",

1. See if q is already in the KB. If so, return TRUE.
2. Find all implications, I, whose conclusion "matches" q.
3. Establish the premises of all i in I via backward chaining.

Resolution Rule of Inference

Assume:	$E_1 \vee E_2$	playing tennis or raining
and	$\neg E_2 \vee E_3$	not raining or working
Then:	$E_1 \vee E_3$	playing tennis or working

General Resolution Rule

$$(L_1 \vee L_2 \vee \ldots L_k)$$
$$(\neg L_k \vee L_{k+1} \vee \ldots L_m)$$
$$\longrightarrow$$
$$(L_1 \vee L_2 \vee \ldots L_{k-1} \vee L_{k+1} \vee \ldots \vee L_m)$$

Algorithm: Resolution Proof

- Negate the theorem to be proved, and add the result to the list of axioms.
- Put the list of axioms into clause form.
- Until there is no resolvable pair of clauses,
  - Find resolvable clauses and resolve them.
  - Add the results of resolution to the list of clauses.
  - Nil is produced, stop and report that the theorem is true.

- Stop and report that the negated theorem is false (and the original theorem is true).

Resolution Example

Example: Prove $\neg P$

Axioms:

	Regular	Clause Form
Axiom 1:	$P \rightarrow Q$	$\neg P \vee Q$
2:	$Q \rightarrow R$	$\neg Q \vee R$
3:	$\neg R$	

Resolution Example (cont.)

1.	P	Assume opposite
2.	$\neg P \vee Q$	Axiom 1
3.	$\neg Q \vee R$	Axiom 2
4.	$\neg R$	Axiom 3
5.	Q	Resolve 1 and 2
6.	R	Resolve 3 and 5
7.	nil	Resolve 4 with 6

Resolution Example: FOL

Example: Prove bird(tweety)

Axioms:

	Regular	Clause Form
1:	$\neg bird(tweety)$	
2:	$\forall x : feathers(x) \rightarrow bird(x)$	
3:	$feathers(tweety)$	
	$feathers(tweety) \rightarrow bird(tweety)$	
	$bird(tweety)$	

Resolution Theorem Proving

- sound (for propositional and FOL)
- complete (for propositional and FOL)

Procedure may seem cumbersome but note that can be easily automated. Just “smash” clauses till empty clause or no more new clauses.

Resolution

I **put in clausal form**
all variables universally quantified
main trick: “skolemization” to remove existentials
idea: invent names for unkown objects known to exist

II **use unification** to match atomic sentences

III **apply resolution rule** to the clausal set combined with negated goal. Attempt to generate empty clause.

Converting more complicated axioms

Axiom:

$$\begin{aligned} \forall x : (brick(x) \rightarrow \quad & \exists y : on(x,y) \wedge \neg pyramid(y) \\ & \wedge \neg\exists y : on(x,y) \wedge on(y,x) \\ & \wedge \forall y : \neg brick(y) \rightarrow \neg equal(x,y)) \end{aligned}$$

$\neg brick(x) \vee on(x, support(x))$
$\neg brick(w) \vee \neg pyramid(support(w))$
$\neg brick(u) \vee \neg on(u,y) \vee \neg on(y,u)$
$\neg brick(v) \vee brick(z) \vee \neg equal(v,z)$

1. Eliminate Implications

Substitute $\neg E_1 \vee E_2$ for $E_1 \rightarrow E_2$

$$\begin{aligned} \forall x : brick(x) \rightarrow \quad & (\exists y : on(x,y) \wedge \neg pyramid(y) \\ & \wedge \neg\exists y : on(x,y) \wedge on(y,x) \\ & \wedge \forall y : \neg brick(y) \rightarrow \neg equal(x,y)) \end{aligned}$$

$$\begin{aligned} \forall x : \neg brick(x) \vee \quad & (\exists y : on(x,y) \wedge \neg pyramid(y) \\ & \wedge \neg\exists y : on(x,y) \wedge on(y,x) \\ & \wedge \forall y : \neg(\neg brick(y)) \vee \neg equal(x,y)) \end{aligned}$$

2. Move negations down to the atomic formulas

$\neg(E_1 \wedge E_2) = (\neg E_1) \vee (\neg E_2)$
$\neg(E_1 \vee E_2) = (\neg E_1) \wedge (\neg E_2)$
$\neg(\neg E_1) = E_1$
$\neg\forall x : E_1(x) = \exists x : \neg E_1(x)$
$\neg\exists x : E_1(x) = \forall x : \neg E_1(x)$

$\forall x : \neg brick(x) \vee$

$(\exists y : on(x,y) \wedge \neg pyramid(y)$

$\wedge\ \neg\exists y :\ on(x,y) \wedge on(y,x)$

$\wedge\ \forall y : \neg(\neg brick(y)) \vee \neg equal(x,y))$

3. Eliminate Existential Quantifiers

Skolemization

Harder cases:
$\forall x : \exists y : father(y,x)$ becomes $\forall x : father(S1(x),x)$

There is one argument for each universally quantified variable whose scope contains the Skolem function.

Easy case:
$\exists x : President(x)$ becomes $President(S2)$

$\forall x : \neg brick(x) \vee (\exists y : on(x,y) \wedge \neg pyramid(y) \wedge \ldots$

4. Rename variables as necessary

We want no two variables of the same name.

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & ((on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge\ \forall y : (\neg on(x,y) \vee \neg on(y,x)) \\ & \wedge\ \forall y : (brick(y) \vee \neg equal(x,y)))\end{aligned}$$

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & ((on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge\ \forall y : (\neg on(x,y) \vee \neg on(y,x)) \\ & \wedge\ \forall z : (brick(z) \vee \neg equal(x,z)))\end{aligned}$$

5. Move the universal quantifiers to the left

This works because each quantifier uses a unique variable name.

$$\begin{aligned}\forall x : \neg brick(x)\ \vee & ((on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge\ \forall y : (\neg on(x,y) \vee \neg on(y,x)) \\ & \wedge\ \forall z : (brick(z) \vee \neg equal(x,z)))\end{aligned}$$

$$\begin{aligned}\forall x \forall y \forall z : (\neg brick(x) \vee & ((on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge\ (\neg on(x,y) \vee \neg on(y,x)) \\ & \wedge\ (brick(z) \vee \neg equal(x,z))))\end{aligned}$$

6. Move disjunctions down to the literals

$E_1 \vee (E_2 \wedge E_3) = (E_1 \vee E_2) \wedge (E_1 \vee E_3)$

$$\begin{aligned} \forall x \forall y \forall z : [&(\neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x)))) \\ &\wedge (\neg brick(x) \vee \neg on(x,y) \vee \neg on(y,x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x,z))] \end{aligned}$$

$$\begin{aligned} \forall x \forall y \forall z : [&(\neg brick(x) \vee on(x, S1(x))) \\ &\wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ &\wedge (\neg brick(x) \vee \neg on(x,y) \vee \neg on(y,x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x,z))] \end{aligned}$$

Slide CS472 – Knowledge-Based Systems 41

7. Eliminate the conjunctions

$$\begin{aligned} \forall x \forall y \forall z : [&(\neg brick(x) \vee on(x, S1(x))) \\ &\wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ &\wedge (\neg brick(x) \vee \neg on(x,y) \vee \neg on(y,x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x,z))] \end{aligned}$$

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$

$\forall x \forall y : \neg brick(x) \vee \neg on(x,y) \vee \neg on(y,x)$

$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x,z)$

Slide CS472 – Knowledge-Based Systems 42

8. Rename all variables, as necessary, so no two have the same name

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$

$\forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)$

$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)$

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall w : \neg brick(w) \vee \neg pyramid(S1(w))$

$\forall u \forall y : \neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$

$\forall v \forall z : \neg brick(v) \vee brick(z) \vee \neg equal(v, z)$

9. Eliminate the universal quantifiers

$\neg brick(x) \vee on(x, S1(x))$

$\neg brick(w) \vee \neg pyramid(S1(w))$

$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$

$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$

Algorithm: Putting Axioms into Clausal Form

- Eliminate the implications.
- Move the negations down to the atomic formulas.
- Eliminate the existential quantifiers.
- Rename the variables, if necessary.
- Move the universal quantifiers to the left.
- Move the disjunctions down to the literals.
- Eliminate the conjunctions.
- Rename the variables, if necessary.
- Eliminate the universal quantifiers.

Unification

UNIFY (P,Q) takes two atomic sentences P and Q and returns a substitution that makes P and Q **look the same.**

Rules for substitutions:

- Can replace a variable by a constant.
- Can replace a variable by a variable.
- Can replace a variable by a function expression, as long as the function expression does not contain the variable.

Unifier: a substitution that makes two clauses resolvable.
$v_1 \rightarrow C; v_2 \rightarrow v_3; v_4 \rightarrow f(...)$

Unification — Purpose

Given:

$Knows(John, x) \rightarrow Hates(John, x)$

$Knows(John, Jim)$

Derive

$Hates(John, Jim)$

Need **unifier** $\{x/Jim\}$ before resolution.
(simplest case)

$\neg Knows(John, x) \vee Hates(John, x)$ and $Knows(John, Jim)$

How do we resolve? First, match them.
Solution:

UNIFY($Knows(John, x)$,$Knows(John, Jim)$) $= \{x/Jim\}$

Gives

$\neg Knows(John, Jim) \vee Hates(John, Jim)$ and
$Knows(John, Jim)$

Conclude by resolution

$Hates(John, Jim)$

Unification (example)

general rule:

$Knows(John, x) \rightarrow Hates(John, x)$

facts:

$Knows(John, Jim)$

$Knows(y, Leo)$

$Knows(y, Mother(y))$

$Knows(x, Jane)$

"matching facts to general rules"

Slide CS472 – Knowledge-Based Systems 49

UNIFY($Knows(John, x)$,$Knows(John, Jim)$) $= \{x/Jim\}$

UNIFY($Knows(John, x)$,$Knows(y, Leo)$) $= \{x/Leo, y/John\}$

UNIFY($Knows(John, x)$,$Knows(y, Mother(y))$) $=$
$\{y/John, x/Mother(John)\}$

UNIFY($Knows(John, x)$,$Knows(x, Jane)$) $= fail$

Slide CS472 – Knowledge-Based Systems 50

Most General Unifier

In cases where there is more than one substitution choose the one that makes the least commitment (most general) about the bindings.

UNIFY$(Knows(John, x), Knows(y, z))$

$= \{y/John, x/z\}$

or $\{y/John, x/z, z/Freda\}$

or $\{y/John, x/John, z/John\}$

or ....

See R&N p. 303 for general unification algorithm. $O(n^2)$

Example

Jack owns a dog.

Every dog owner is an animal lover.

No animal lover kills an animal.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

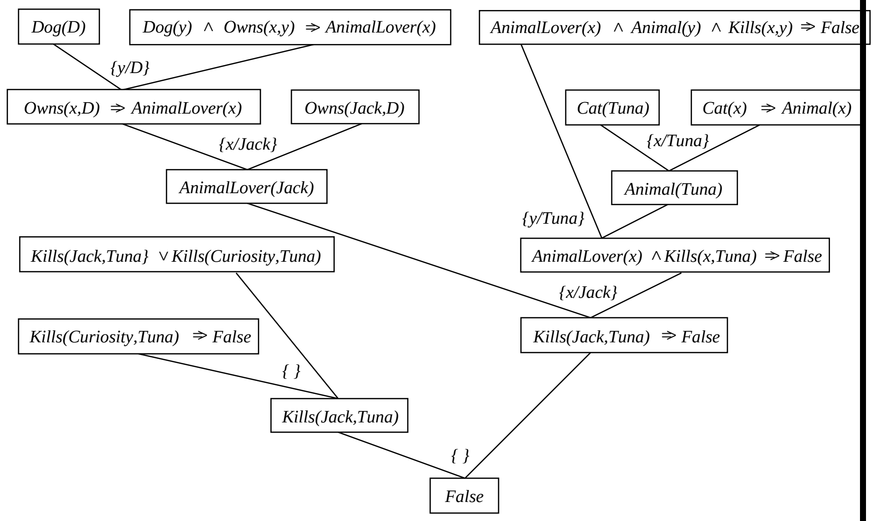
Original Sentences (Plus Background Knowledge)

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$
2. $\forall x \ \ (\exists y \ \ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
3. $\forall x \ \ AnimalLover(x) \rightarrow \forall y \ \ Animal(y) \rightarrow \neg Kills(x, y)$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x \ \ Cat(x) \rightarrow Animal(x)$

Clausal Form

1. $Dog(D)$ (D is the function that finds Jack's dog)
2. $Owns(Jack, D)$
3. $\neg Dog(S(x)) \vee \neg Owns(x, S(x)) \vee AnimalLover(x)$
4. $\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$
5. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
6. $Cat(Tuna)$
7. $\neg Cat(z) \vee Animal(z)$

Proof by Resolution with Refutation


Dog(D)
Dog(y) ∧ Owns(x,y) ⇒ AnimalLover(x)
AnimalLover(x) ∧ Animal(y) ∧ Kills(x,y) ⇒ False
{y/D}
Owns(x,D) ⇒ AnimalLover(x)
Owns(Jack,D)
Cat(Tuna)
Cat(x) ⇒ Animal(x)
{x/Jack}
{x/Tuna}
AnimalLover(Jack)
Animal(Tuna)
{y/Tuna}
Kills(Jack,Tuna} ∨ Kills(Curiosity,Tuna)
AnimalLover(x) ∧ Kills(x,Tuna) ⇒ False
{x/Jack}
Kills(Curiosity,Tuna) ⇒ False
Kills(Jack,Tuna) ⇒ False
{ }
Kills(Jack,Tuna)
{ }
False

Slide CS472 – Knowledge-Based Systems 55

Completeness

Resolution with **unification** applied to **clausal form**, is **refutation** complete.

In practice, still significant search problem!
Many different search strategies: **resolution strategies**.

Slide CS472 – Knowledge-Based Systems 56

Strategies for Selecting Clauses

unit-preference strategy: Give preference to resolutions involving the clauses with the smallest number of literals.

set-of-support strategy: Try to resolve with the negated theorem or a clause generated by resolution from that clause.

subsumption: Eliminates all sentences that are subsumed (i.e., more specific than) an existing sentence in the KB.

May still require **exponential** time.

Schubert Steamroller

```
Wolves, foxes, birds, caterpillars, and snails are animals,
and there are some of each of them. Also there are some
grains, and grains are plants.

Every animal either likes to eat all plants or
all animals much smaller than itself that like
to eat some plants.

Caterpillars and snails are much smaller than birds,
which are much smaller than foxes, which are much
smaller than wolves.
\newpage
```

```
Wolves do not like to eat foxes or grains, while
birds like to eat caterpillars but not snails.

Caterpillars and snails like to eat some plants.
```

Some of the necessary logical forms:

$\forall x\ (Wolf(x) \Rightarrow animal(x))$
$\forall x\ \forall y\ ((Caterpillar(x) \wedge Bird(y)) \Rightarrow Smaller(x, y).$
$\exists x\ \ bird(x)$

To prove:

```
There is an animal that likes to eat
         a grain-eating animal.
```

Requires almost 150 resolution steps (minimal).
Significant challenge for early systems.

A relatively straightforward KB can quickly
overwhelm general resolution methods.
Resolution strategies reduce the problem somewhat,
but not completely.
As a consequence, many **practical** Knowledge Representation
formalisms in AI use a **restricted form**
and **specialized inference.**

Practical Knowledge-Based Systems (Chap. 10, R&N)

- **Theorem provers / logic programming**
- **Production systems**
  forward chaining / if-then-rules / **expert** systems
- **Frame systems and semantic networks**
- **Description logics**

Theorem provers / logic programming

Theorem provers: generally based on resolution
many different strategies to improve efficiency
Logic programming: program statements directly in restricted FOL.
Execution: search for proof of goal/query
using backward chaining with depth first-search.
In certain cases too inefficient.

Production systems

- rich history in AI
- **"expert system"** boom in 70's / 80's

Basic idea:

capture knowledge of human expert in a large set of "if-then" rules

(really, logical implication $\Rightarrow$)

"production rules"

Components of Rule-Based Systems

working memory: set of positive literals with no variables

rule memory: set of inference rules

$$p_1 \wedge p_2 \ldots \rightarrow a_1 \wedge a_2 \ldots$$

where the p_i are literals, and the a_i are actions to take when the p_i are all satisfied

rule interpreter: inference engine

Sample Knowledge Base

Working Memory

(in robot room1)
(armempty robot)
(in table room1)
(on cup table)
(object table)
(object cup)
(room room1)
(room room2)

1. *(in robot ?x)* ∧ *(room ?y)* →
   (walk ?x ?y) ∧ (add (in robot ?y))∧ (delete (in robot ?x))

2. *(in robot ?x)* ∧ *(in ?y ?x)* ∧ *(object ?y)* ∧ *(not (at robot ?y))*
   → (walk ?x ?y) ∧ (add (at robot ?y))

3. *(in robot ?x)* ∧ *(at robot ?y)* ∧ *(clear ?y)* ∧ *(armempty robot)*
   ∧ *(room ?z)* → (push ?y ?z) ∧ (add (in robot ?z)) ∧ (add (in ?y ?z)) ∧ (delete (in robot ?x)) ∧ (delete (in ?y ?x))

4. *(at robot ?x)* ∧ *(armempty robot)* ∧ *(on ?y ?x)* →
   (pickup ?y) ∧ (add (holding robot ?y)) ∧ (add (clear ?x)) ∧ (delete (armempty robot)) ∧ (delete (on ?y ?x))

5. *(holding robot ?x)* → (putdown ?x) ∧ (add (armempty robot)) ∧ (delete (holding robot ?x)) ∧ (add (on ?x floor))

Reasoning with Rules

Forward Reasoning

1. Until no rule can fire or goal state is achieved,
   - (a) Find all rules whose *left* sides match assertions in working memory.
   - (b) Pick some to execute; modify working memory by applying the *right* sides of the rules.

Three Parts to Forward-Chaining Rule Interpreter

Match: identifying which rules are applicable at any given point in the reasoning

Conflict Resolution: selecting which of many rules should be applied at any given point in the reasoning

Execute: execute the right-hand side of the rule

Example: forward chaining

Goal: (in table room2)

1.	Rule 1	x = room1, y = room1, y = room2
	Rule 2	x = room1, y = table

Choose randomly; assume rule 2.

Walk to table.

Add: (at robot table)

2.	Rule 1	same bindings
	Rule 2?	

	Rule 4	x = table, y = cup

Choose randomly; assume rule 4.

Pick-up cup.

Add: (holding robot cup)
(clear table)

Delete: (armempty robot)

3.	Rule 1	same bindings
	Rule 5	x = cup

Choose randomly; assume rule 5.

Putdown cup.

Add: (armempty robot)

Add:	(on cup floor)	
Delete:	(holding robot cup)	
4.	Rule 1	same bindings
	Rule 3	x = room1, y = table, z = room1 or room2

Choose randomly; assume rule 3.

Push table (to) room2.

Add:	(in robot room2)
	(in table room2)
Delete:	(in robot room1)
	(in table room1)

Matching for Forward-Chaining

- requires smart indexing of the rules
- requires unification

Problem: For practical systems, applying unification in a straightforward manner will be very inefficient.

Rete Algorithm: Many to Many Matching

When matching many conditions to many WM elements, standard unification algorithm won't be efficient enough. Rete exploits:

- Temporal nature of data: each rule usually modifies only a small number of conditions
- Structural similarity in rules: different rules often share a number of preconditions.

$$marsupial(X) \wedge australian(X) \wedge hops(X) \rightarrow kangaroo(X)$$

$$marsupial(X) \wedge australian(X) \wedge climbs(X) \rightarrow koala(X)$$

Rete (cont.)

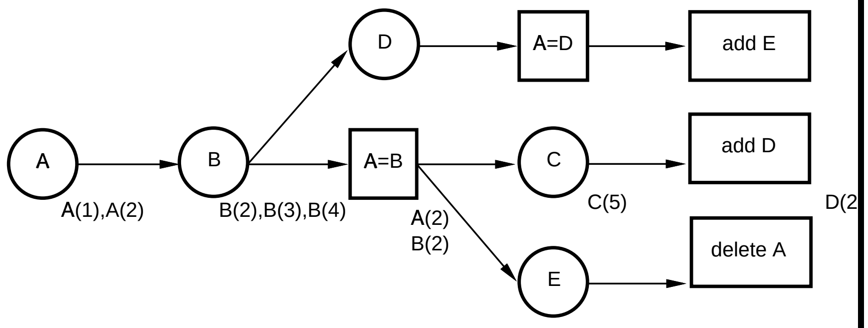
Rete exploits:

- Persistence of variable binding consistency: keeps track of variable bindings which may prevent the rule from firing.

$$son(X, Y) \wedge son(Y, Z) \rightarrow grandparent(X, Z)$$

$$son(Mary, Joe) \wedge son(Bill, Bob)$$

A Rete Network



Approximate Matching

Matching can be complicated by allowing partial or approximate matches.

For example, consider cases in which noisy signals are being matched (e.g., speech, vision and other sensory inputs). Rarely will exact matches be found.

Meta-Rules for Guiding the Search

(goal (in table room2))

Rule Memory:

1. (goal (in ?x ?y)) $\wedge$ (not (in ?x ?y)) $\rightarrow$
   (add (goal (change-loc ?x ?y)))
2. (goal ?x) $\wedge$?x $\rightarrow$ DONE

Control Strategies

Concerned with the general issue of how to **control** the reasoning process so that it efficiently finds a solution.

Control in Search refers to the order or selection of nodes/states to explore.

Control in Rule Bases refers to the order or selection of rules/actions to execute at each cycle of a rule-based system. *(conflict resolution)*

output of *matching* $\Rightarrow$ list of applicable rules and their variable bindings

output of *conflict resolution* $\Rightarrow$ *which* rule to apply

Syntactic Approaches to Conflict Resolution

- *Preferences based on the rules that matched*
  - Order the rules
  - Prefer special cases over more general ones
    - ∗ If one precondition set is a proper subset of another, then the second rule is a special case of the first.
    - ∗ If two precondition sets are identical except that one contains variables and the other constants, the second rule is a special case of the first.

Syntactic Approaches to Conflict Resolution

- *Preferences based on objects/assertions that matched*
  - Prefer recently added objects
- Combinations
  - LEX
    1. remove repeat firings from consideration,
    2. prefer rules that bind variables to recently added conditions,
    3. prefer the most complicated precondition,
    4. random choice

Semantic Approaches to Conflict Resolution

- *Preferences based on objects that matched*
  - Add information to rules to indicate more important objects
- *Preferences based on the action that the matched rule would perform*
- *Treat conflict resolution as (search) problem in its own right*

Successes in Rule-Based Reasoning

Expert systems

- DENDRAL (Buchanan *et al.*, 1969)
- MYCIN (Feigenbaum, Buchanan, Shortliffe)
- PROSPECTOR (Duda *et al.*, 1979)
- R1 (McDermott, 1982)

Successes in Rule-Based Reasoning

- DENDRAL (Buchanan *et al.*, 1969)
  - infers molecular structure from the information provided by a mass spectrometer
  - generate-and-test method

  -
    ```
    if  there are peaks at x_1 and x_2 s.t.
           x_1 + x_2 = M + 28
           x_1 - 28 is a high peak
           x_2 - 28 is a high peak
           At least one of x_1 and x_2 is high
    then there is a ketone subgroup
    ```

- MYCIN (Feigenbaum, Buchanan, Shortliffe)
  - diagnosis of blood infections
  - 450 rules; performs as well as experts
  - incorporated **certainty factors**

```
If: (1) the stain of the organism is
        gram-positive, and
    (2) the morphology of the organism is
        coccus, and
    (3) the growth conformation of the organism
        is clumps,
then there is suggestive evidence (0.7) that the
  identity of the organism is staphylococcus.
```

- PROSPECTOR (Duda *et al.*, 1979)
  - correctly recommended exploratory drilling at a geological site
  - rule-based system founded on probability theory
- R1 (McDermott, 1982)
  - designs configurations of computer components
  - about 10,000 rules
  - uses meta-rules to change context

```
If: current context is ?x
then: deactivate ?x context
      and activate ?y context
```

Cognitive Modeling with Rule-Based Systems

SOAR is a general architecture for building intelligent systems.

- Long term memory consists of rules.
- Working memory describes current state.
- All problem solving, including deciding what rule to execute, is state space search.
- Successful rule sequences are *chunked* into new rules.
- Control strategy embodied in terms of meta-rules.

Example Syntax for Control Rule

Meta-rule

Under conditions A and B,
Rules that do {*not*} mention X

{*at all,*
in their LHS,
in their RHS }

will

{*definitely be useless,*
probably be useless, ...
probably be especially useful,
definitely be especially useful }

Advantages of Rule-Based Systems

1. Expressibility*
2. Simplicity of control*
3. Modifiability*
4. Explainability
5. Machine readability
6. Parallelism*

Disadvantages of Rule-Based Systems

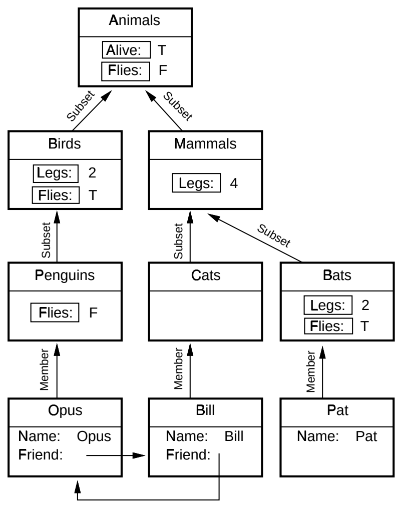
1. Difficulties in expressibility
2. Obscure control
3. Undesirable interactions among rules
4. Non-transparent behavior
5. Difficult debugging
6. Slowness

Knowledge Engineering

If we're acquiring the rules from an expert, there are two heuristics to keep in mind:

1. Ask about *specific situations* to learn the expert's general knowledge.
2. Ask about situation pairs that look identical, but that are handled differently. *situation comparison*

Semantic Networks and Frame-Based KB's



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

Birds ⊂ Animals
Mammals ⊂ Animals

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

Penguins ⊂ Birds
Cats ⊂ Mammals
Bats ⊂ Mammals

Rel(Flies,Penguins,F)
Rel(Legs,Bats,2)
Rel(Flies,Bats,T)

Opus ∈ Penguins
Bill ∈ Cats
Pat ∈ Bats

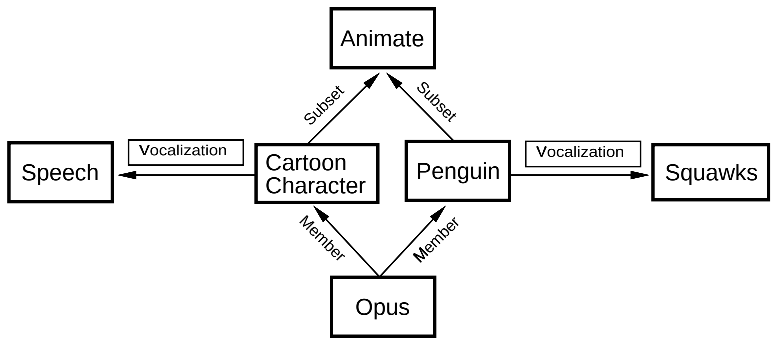
Name(Opus,"Opus")
Name(Bill,"Bill")
Friend(Opus,Bill)
Friend(Bill,Opus)
Name(Pat,"Pat")

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

Link Types

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

Multiple Inheritance



Slide CS472 – Knowledge-Based Systems 95

Advantages and Disadvantages of Semantic Networks

- Extremely limited in expressiveness
- Capture inheritance information in a modular way
- Simplicity makes them easy to understand
- Efficient inference procedures

Slide CS472 – Knowledge-Based Systems 96