Automated Reasoning: Logical Approaches

UMaine COS 470/570 – Introduction to AI

Spring 2019

Automated reasoning

Reasoning

- *Reasoning* = ability to make decision or infer something from existing facts
- *Automated reasoning*:
  - Search is one (very simple) kind
  - Neural networks: *non-symbolic*
  - Here: *symbolic* reasoning
    - Encode *knowledge* in some *representation*
    - Apply inference mechanisms ⇒ new knowledge

Why not just search for everything?

- Realistic problems: search spaces *very* large, potentially infinite
- Difficult to find heuristics
- Often problem has structure that can be exploited
- Often: $\exists$ much knowledge about world, problem
  - E.g., medicine
- Search: example of *weak method*:
  - general purpose
  - little knowledge
- Knowledge-based methods: *strong methods*

Knowledge representation

Knowledge

- Need way to *represent* & use the knowledge
- Many different representation schemes, inference methods
  - Theorem proving:
    - Represent knowledge in a logical formalism
    - Inference methods that knowledge $\Rightarrow$ new knowledge
  - Rule-based reasoners:
    - Represent knowledge as “if–then” rules
    - Apply the rules $\Rightarrow$ new knowledge
  - Planners:
    - Represent knowledge as plan schemas, rules/logic, ...
    - Use specialized planning techniques $\Rightarrow$ plans
  - Many others

Kinds of knowledge

- Problem-specific: start, goal states, map, . . .
- Domain
- Problem-solving, other domain-independent
- Meta-knowledge: for explanation, learning, etc.

Knowledge & agents

- *All* agents have knowledge
- Some: built in to the agent's structure
  - e.g., reflex agent
  - *implicit* knowledge
- Some augment with verbatim history
- Some: *explicit* knowledge representation
  - Search agents
  - Goal-based, utility-based agents

Why explicit knowledge?

- Agent reuse: just replace knowledge
- Knowledge acquisition from humans
- Reasoning about it:
  - by humans: proving properties about behavior, e.g.
  - by agent itself: introspection, machine learning, explanation, . . .

Knowledge representation

- *Knowledge representation*:
  1. system of representation, or...
  2. way to represent particular concepts, or...
  3. collection of knowledge an agent has (informally; really *knowledge base*)
- Representations often *formal*:
  - Rules about what can be stored
  - Particular syntax, semantics
- Others interested in knowledge representation:
  - psycologists
  - philosophers

Models and abstraction

- Knowledge representation *models* a world
  - Abstraction of a world: some things are left out
  - Focuses, limits reasoning
- Model's creator:
  - Determines salient features
  - Determines *granularity* of model

Knowledge representation criteria

- Criteria
  - Easy for humans to understand
  - Concise
  - Context-independent
  - Context-dependent
  - Compositional
  - Canonical
  - Appropriate granularity
  - Representational adequacy
  - Inferential adequacy
  - Acquisitional adequacy
- Trade-offs!

Syntax, semantics, pragmatics

- Knowledge representation is a *language*
- Syntax: valid structure of sentences
- Semantics: meaning of sentences
- Pragmatics (sometimes): what the sentences mean in context

Kinds of knowledge representations

- Implicit/structural
- Procedural, but explicit:
  - *how* to do something – like program
  - good for instructions
  - may be hard for humans to understand
  - may be hard for the agent to understand and/or learn
- Declarative/explicit:
  - represents *what* something is, what to do
  - easy to extend, understand
  - program can access its own knowledge: introspection, learning
  - harder to represent sometimes than procedural
  - less efficient to "execute" than procedural
- Structured vs. unstructured

First-order logic

Formal logic

- A *logic* is a representation language with precisely-defined syntax and semantics
- Sentences represent facts
- *Syntax:* describes the possible legal configurations of elements that form valid sentences
- *Semantics:* one interpretation is facts to which the sentences refer
- $\exists$ many logics

Inference

- *Inference:* creates new knowledge from old
- Human inferences – can be very broad, complex
- Machine inferences:
  - smaller than might usually count
  - anything that is not a direct match with the knowledge base requires an inference

Inference

- A *logic* has associated reasoning mechanisms:
- *Inference rules:* create new sentence from existing sentences
- *Inference procedure:* Produces new facts from old:

$$S_0, S_1, \cdots, S_n \vdash A$$

- Theorem prover: uses inference rules to prove some sentence

Entailment

- Want to know:
  - Does sentence A follow from a knowledge base K of sentences?
  - I.e., is A true if K is true?
- *Entailment:*
  - K entails A iff A is necessarily true given K
  - Written $K \models S$
  - Note: $\models$ could take ≥ 1 inference
  - For inference procedure i, written: $KB \models_i S$
- *Sound (truth-preserving) inference procedure:* produces only entailed sentences

Proof

- *Proof:* record of operation of a sound inference procedure

Proof

- *Proof:* record of operation of a sound inference procedure
- *Complete* inference procedure P:

$$\forall s\, K \models s \Rightarrow K \models_P s$$

Proof

- *Proof:* record of operation of a sound inference procedure
- *Complete* inference procedure P:

$$\forall s\, K \models s \Rightarrow K \models_P s$$

- *Proof theory:* set of rules for deducing the entailments of set of sentences (R&N)

Proof

- *Proof:* record of operation of a sound inference procedure
- *Complete* inference procedure P:

$$\forall s\, K \models s \Rightarrow K \models_P s$$

- *Proof theory:* set of rules for deducing the entailments of set of sentences (R&N)

Proof

- *Proof:* record of operation of a sound inference procedure
- *Complete* inference procedure P:

$$\forall s\, K \models s \Rightarrow K \models_P s$$

- *Proof theory:* set of rules for deducing the entailments of set of sentences (R&N)

$$\boxed{\text{Logic} = \text{syntax} + \text{semantics} + \text{proof theory}}$$

Models

- Natural language sentences:
  - Shared conventions, knowledge among speakers
  - Meaning of sentence from these $\Rightarrow$ truth, falsehood
- Truth in logic:
  - One kind of truth: entailment – s is true given K iff $K \models s$
  - But what about the normal meaning of “true”?
- Meaning/truth beyond entailment:
  - No inherent meaning of sentences
  - Meaning (truth) of sentence S depends on some *interpretation*
- *Model:* a world in which sentence is true given some interpretation

Models

- Natural language sentences:
  - Shared conventions, knowledge among speakers
  - Meaning of sentence from these $\Rightarrow$ truth, falsehood
- Truth in logic:
  - One kind of truth: entailment – s is true given K iff $K \models s$
  - But what about the normal meaning of "true"?
- Meaning/truth beyond entailment:
  - No inherent meaning of sentences
  - Meaning (truth) of sentence S depends on some *interpretation*
- *Model:* a world in which sentence is true given some interpretation

$$K \models s \text{ iff all models of } K \text{ are also models of } s$$

Validity

- Valid sentence: true in all possible worlds (i.e., a *tautology*)
- Valid inference: if premise true, conclusion *must* be true in any world:

  All humans are mortal and I am a human $\Rightarrow$ I am mortal All birds live underground and Tweety is a bird $\Rightarrow$ Tweety lives underground

Soundness

- Tend to use *sound* interchangeably with valid, but not really same
- Inference is sound if premises true *and* inference is valid
- Argument (proof) is sound if all inferences are valid *and* premises are true
- I.e., soundness is with respect to a model (world)

Satisfiability

- *Satisfiable sentence:*
  - Some interpretation in some world for which sentence is true
  - E.g.: My cat hates dogs.
- Non-satisfiable sentence
  - No world in which sentence is true
  - E.g.:
    - I am mortal and I am not mortal.
    - Every cat hates dogs and there is a cat that does not hate dogs.

Propositional Logic

Propositional logic (calculus)

- Simplest kind of logic: “zeroth-order logic”
- Sentences = *propositions*
- Symbols stand for propositions
- Symbols, connectives $\Rightarrow$ compound propositions
- No variables, $\therefore$ no quantification
- *Ontological commitment:* there are facts in world that are true
- *Epistemological commitment:* a sentence is true or false

Syntax

- Elements of language:
  - Symbols
  - True, False
  - Logical connectives, parentheses
- Recursive definition:
  - True, False, symbol are propositions (*atomic* sentences)
  - If S, P and Q are sentences, then so are:

$$(S),\quad P \wedge Q,\quad P \vee Q,\quad \neg P,\quad P \Rightarrow Q,\quad \text{and } P \Leftrightarrow Q$$

- *Literal*: atomic sentence or negated atomic sentence
- Precedence rules: $\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow$

Semantics

- True, False: fixed interpretation
- Propositions + connectives: “standard” compositional semantics
- Propositions: whatever interpretation they are given

Connectives

A	$\neg A$
F	T
T	F

A	B	$A \vee B$
F	F	F
F	T	T
T	F	T
T	T	T

Connectives

A	B	$A \wedge B$
F	F	F
F	T	F
T	F	F
T	T	T

Implication

A	B	$A \Rightarrow B$
F	F	T
F	T	F
T	F	T
T	T	T

- Seems odd
- Think of it as: If A True, then I claim B is true, else I make no claim
- Only time $A \Rightarrow B$ is false is if B is false
  - E.g.: Trump is president $\Rightarrow$ he didn't win the election
- Implication true when antecedent is false:
  - E.g.: Clinton is president $\Rightarrow$ she won the election
- Definition: $P \Rightarrow Q \equiv \neg P \vee Q \equiv \neg(P \wedge \neg Q)$

Inference rules for propositional logic

- Double negation elimination:

$$\frac{\neg\neg A}{A}$$

- AND elimination (unidirectional only):

$$\frac{A_1 \wedge A_2 \wedge \ldots \wedge A_n}{A_i}$$

- OR introduction (unidirectional only):

$$\frac{A_i}{A_1 \vee A_2 \vee \ldots \vee A_n}$$

Inference rules for propositional logic

- De Morgan's laws:

$$\frac{\neg(A \wedge B)}{\neg A \vee \neg B}$$

$$\frac{\neg(A \vee B)}{\neg A \wedge \neg B}$$

- Distributive:

$$\frac{A \vee (B \wedge C)}{(A \vee B) \wedge (A \vee C)}$$

$$\frac{A \wedge (B \vee C)}{(A \wedge B) \vee (A \wedge C)}$$

Inference rules for propositional logic

- Various others: (0 = false, 1 = true)
  - Null law:

$$\frac{A \wedge 0}{0}, \frac{A \vee 1}{1}$$

  - Identity law:

$$\frac{A \wedge 1}{A}, \frac{A \vee 0}{A}$$

  - Idempotent law:

$$\frac{A \wedge A}{A}, \frac{A \vee A}{A}$$

Deduction

- Sound form of inference
- **Modus ponens**
  - Form:

$$\frac{\begin{array}{l} A \Rightarrow B \\ A \end{array}}{B}$$

  - Example:

$$\frac{\begin{array}{l} \text{Bird} \Rightarrow \text{Fly} \\ \text{Bird} \end{array}}{\text{Fly}}$$

Deduction

- **Modus tolens**
  - Form:

$$\frac{\begin{array}{l} A \Rightarrow B \\ \neg B \end{array}}{\neg A}$$

  - Example:

$$\frac{\begin{array}{l} \text{Bird} \Rightarrow \text{Fly} \\ \neg\text{Fly} \end{array}}{\neg\text{Bird}}$$

Complexity of propositional inference

- Could build a truth table to prove conclusion
- 2^n rows – n propositional symbols – can we do better?
- General case: no – NP-complete problem
- Horn clauses: one class for which P-time algorithm exists

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow Q$$

– P_i, Q – non-negated atoms

Problems with propositional calculus

- Too many propositions!
- No variables – no quantification

Predicate Calculus

First-order predicate calculus

- Various names: first-order logic (FOL), first-order predicate calculus (FOPC), . . .
- Ontological commitment
  - world consists of objects that have properties
  - various relations hold among objects
  - $\exists$ functions arguments (objects) $\rightarrow$ objects
- FOPC can represent anything that can be programmed

Parts of predicate calculus

- *Term*: something signifying an object
  - Symbol
  - Variable
  - *Function* (N.B.: *not* like function in programs!)
- *Negation:* NOT
- *Connectives:* AND ($\wedge$), OR ($\vee$), IMPLIES ($\Rightarrow$), and sometimes $\Leftrightarrow$ or $\equiv$, $=$
- *Quantifiers:* existential ($\exists$) & universal ($\forall$)

Literals, clauses, and sentences

- *Literal*: a term, a predicate applied to term(s), or negated predicate applied to term(s)
- *Well-formed formulas* (*wffs*): statements in the logic
  - Literals are wffs
  - If A & B are wffs so are:

$$A \vee B \qquad A \wedge B \qquad A \Rightarrow B$$

$$\exists A \qquad \forall A$$

- *Clause* - a wff consisting of solely of a disjunction of literals
- *Sentence:* a wff with no *free variables*

Computable functions

- Problem:
  - When proving a theorem, need to check truth/falsehood of predicates
  - Ultimately, predicates have to match against knowledge base (possibly after some number of inferences)
  - Some predicates: need infinite number of facts in the knowledge base! E.g., numeric predicates:

    $$\forall x, y\ \texttt{Pompeian}(x) \wedge \texttt{born}(x, y) \wedge \texttt{less}(y, 79) \Rightarrow \texttt{dead}(x)$$

    For this, we'd have to have an infinite number of facts in our KB:

    $$\texttt{less}(78, 79), \texttt{less}(77, 79), \texttt{less}(76, 79) \ldots$$

- Solution: Evaluate as T or F by running a function on the computer, not matching to a knowledge base

Representing knowledge in FOPC

- Remember: symbols are just symbols and have no additional meaning
- Have a *corpus* of knowledge
  - depends on domain, task, goals, etc.
  - do not attempt to represent everything
  - first specified in English, usually
  - corpus will probably change as work on system
- Identify predicates that will be used

Representing an example corpus

- John likes carrots. `likes(John,Carrots)`
- Mary likes carrots.
- John grows the vegetables he likes.
- Carrots are vegetables.
- When you like a vegetable, you grow it.
- To eat something, you have to own it.
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. `likes(John,Carrots)`
- Mary likes carrots. `likes(Mary,Carrots)`
- John grows the vegetables he likes.
- Carrots are vegetables.
- When you like a vegetable, you grow it.
- To eat something, you have to own it.
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables.
- When you like a vegetable, you grow it.
- To eat something, you have to own it.
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables. $\texttt{vegetables}(\mathit{Carrots})$
- When you like a vegetable, you grow it.
- To eat something, you have to own it.
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables. $\texttt{vegetables}(\mathit{Carrots})$
- When you like a vegetable, you grow it.
  $\forall x, y\ \texttt{vegetable}(x) \wedge \texttt{person}(y) \wedge \texttt{like}(y, x) \rightarrow \texttt{grows}(y, x)$
- To eat something, you have to own it.
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables. $\texttt{vegetables}(\mathit{Carrots})$
- When you like a vegetable, you grow it.
  $\forall x, y\ \texttt{vegetable}(x) \wedge \texttt{person}(y) \wedge \texttt{like}(y, x) \rightarrow \texttt{grows}(y, x)$
- To eat something, you have to own it.
  Which (if either) of these:
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{owns}(x, y) \rightarrow \texttt{eats}(x, y)$
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{eats}(x, y) \rightarrow \texttt{owns}(x, y)$
- When you grow something, you own it.
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables. $\texttt{vegetables}(\mathit{Carrots})$
- When you like a vegetable, you grow it.
  $\forall x, y\ \texttt{vegetable}(x) \wedge \texttt{person}(y) \wedge \texttt{like}(y, x) \rightarrow \texttt{grows}(y, x)$
- To eat something, you have to own it.
  Which (if either) of these:
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{owns}(x, y) \rightarrow \texttt{eats}(x, y)$
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{eats}(x, y) \rightarrow \texttt{owns}(x, y)$
- When you grow something, you own it.
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{grows}(x, y) \rightarrow \texttt{owns}(x, y)$
- In order to grow something, you must own a garden.

Representing an example corpus

- John likes carrots. $\texttt{likes}(\texttt{John}, \texttt{Carrots})$
- Mary likes carrots. $\texttt{likes}(\texttt{Mary}, \texttt{Carrots})$
- John grows the vegetables he likes.
  $\forall x\ \texttt{vegetable}(x) \wedge \texttt{likes}(\texttt{John}, x) \rightarrow \texttt{grows}(\texttt{John}, x)$
- Carrots are vegetables. $\texttt{vegetables}(\mathit{Carrots})$
- When you like a vegetable, you grow it.
  $\forall x, y\ \texttt{vegetable}(x) \wedge \texttt{person}(y) \wedge \texttt{like}(y, x) \rightarrow \texttt{grows}(y, x)$
- To eat something, you have to own it.
  Which (if either) of these:
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{owns}(x, y) \rightarrow \texttt{eats}(x, y)$
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{eats}(x, y) \rightarrow \texttt{owns}(x, y)$
- When you grow something, you own it.
  $\forall x, y\ \texttt{person}(x) \wedge \texttt{grows}(x, y) \rightarrow \texttt{owns}(x, y)$
- In order to grow something, you must own a garden.
  Which?
  $\forall x \exists g, y\ \texttt{garden}(g) \wedge \texttt{owns}(x, g) \rightarrow \texttt{grows}(x, y)$
  $\forall x \exists g, y\ \texttt{garden}(g) \wedge \texttt{grows}(x, y) \rightarrow \texttt{owns}(x, g)$

Rules of inference

- *modus ponens:* If $(A \rightarrow B) \wedge A$ then B logically follows.
- *modus tolens:* If $(A \rightarrow B) \wedge \neg B$ then $\neg A$ logically follows
- *resolution:* If $(A \vee B) \wedge (\neg B \vee C)$ then $(A \vee C)$ logically follows
- *abduction:* If $(A \rightarrow B) \wedge B$ then $A \Leftarrow$ not sound
- *induction:* If $(instance(A, B) \wedge P) \wedge (instance(C, B) \wedge P)$, then $instance(x, B) \rightarrow P \Leftarrow$ not sound

Proof by deduction

- Put what you want to prove in the knowledge base
- Apply rules of inference in a systematic way
- Add inferences along the way to knowledge base since made from sound inferences
- Need to make sure that matching is done correctly

Miscellaneous FOPC topics

- Bijection ($\Leftrightarrow$): iff

$$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$$

- Equality
  - Often used in FOPC to link two descriptions as referring to the same object:

$$\texttt{FatherOf}(\texttt{John}) = \texttt{Henry}$$

  - Often used in formulae; sometimes to make sure that two things are not the same object:

$$\exists x, y\, \texttt{Dog}(x) \wedge \texttt{Dog}(y) \wedge \neg(x = y)$$

Artificial Intelligence

Miscellaneous FOPC topics

- Lambda (λ) expressions:
  - Temporary functions/predicate expressions (as in Lisp)

$$\lambda x, y\, \texttt{Nationality}(x) \neq \texttt{Nationality}(y) \wedge$$
$$\texttt{SchoolYear}(x) = \texttt{SchoolYear}(y)$$
$$(\lambda x, y\, \texttt{Nationality}(x) \neq \texttt{Nationality}(y) \wedge$$
$$\texttt{SchoolYear}(x) = \texttt{SchoolYear}(y))(\texttt{Joe}, \texttt{Pierre})$$

- Doesn't extend FOPC – can always replace lambda exp. with expansion

Miscellaneous FOPC topics

- Uniqueness quantifier $\exists!$
  - Ex:
    $$\exists! \mathtt{President}(x, \mathtt{USA})$$
  - Also doesn't extend FOPC – just *syntactic sugar* for:

$$\exists \mathtt{President}(x, \mathtt{USA}) \wedge \forall y\, \mathtt{President}(y, \mathtt{USA}) \Rightarrow x = y$$

Theorem proving

Theorem Proving

- What good is it?
- Axioms – more or less self-evident things that are "given"
- Theorems

  1. Must contain nothing that cannot be proven
  2. Must be implied entirely by propositions other than itself in or arising from the axioms
  3. Two theorems proven from the same set of (consistent) axioms cannot be contradictory

Theorem Proving

- What good is it?
- Axioms – more or less self-evident things that are "given"
- Theorems
  1. Must contain nothing that cannot be proven
  2. Must be implied entirely by propositions other than itself in or arising from the axioms
  3. Two theorems proven from the same set of (consistent) axioms cannot be contradictory
- Theorem proving in this course:
  - Unification
  - Axioms
  - Forward and backward proof
  - Resolution theorem proving

Matching in Theorem Proving

- Where is matching needed?
  - Determining if something is trivially true – i.e., in the KB
  - Determining if something matches the antecedent (consequent)of an implication

Matching in Theorem Proving

- Where is matching needed?
  - Determining if something is trivially true – i.e., in the KB
  - Determining if something matches the antecedent (consequent)of an implication
- What properties should our match function have?
  - Identical things match.
  - Variables can match constants, unless the variable is already bound in an inconsistent way
  - Should keep track of *bindings* so variables consistency can be checked, so *instantiation* of axioms can be done

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$	no	no syntactic match

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$	no	no syntactic match
$dog(x)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$	no	no syntactic match
$dog(x)$	yes	Pluto can subsitute for variable: x/Pluto

Artificial Intelligence

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$	no	no syntactic match
$dog(x)$	yes	Pluto can subsitute for variable: x/Pluto
$\neg dog(x)$		

Unification

- A particular kind of *matching* – Allow variables, track substitutions of things for variables
- Thing to match: $dog(Pluto)$

Proposition	Match?	Why?
$dog(Pluto)$	yes	identical
$\neg dog(Pluto)$	no	negated literal
$dog(Fido)$	no	constant term mismatch
$\neg dog(Fido)$	no	no syntactic match
$cat(Pluto)$	no	predicate mismatch
$\neg cat(Pluto)$	no	no syntactic match
$dog(x)$	yes	Pluto can subsitute for variable: x/Pluto
$\neg dog(x)$	no	negated

Unification

- Basic idea for literals: check negation, check predicates, check arguments
- Matching rules:
  - symbols only match themselves
  - variable can match anything X unless:
    - X contains the variable
    - the variable has been bound to something that doesn't itself match X
  - Variable *binding*
  - Subsitutions — also called a *binding list* or a *unifier*

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	$\{$x/Pluto$\}$, $\{$x$\rightarrow$Pluto$\}$, or ((x Pluto))

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	{x/Pluto}, {x→Pluto}, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	{x/Pluto}, {x$\rightarrow$Pluto}, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	{x/A, y/A}

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	$\{$x/Pluto$\}$, $\{$x$\rightarrow$Pluto$\}$, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	$\{$x/A, y/A$\}$
(P ?x ?x)	(P ?y ?z)	

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	{x/Pluto}, {x→Pluto}, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	{x/A, y/A}
(P ?x ?x)	(P ?y ?z)	{x/y, y/z}

Artificial Intelligence

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	{x/Pluto}, {x→Pluto}, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	{x/A, y/A}
(P ?x ?x)	(P ?y ?z)	{x/y, y/z}
(owns Minnie ?y)	(owns ?z Pluto)	

Substitution in Unification

- Substitution $\equiv$ *unifier*
- Examples: Assume ?z is already bound to Mickey

A	B	unify(A,B)
(dog ?x)	(dog Pluto)	$\{$x/Pluto$\}$, $\{$x$\rightarrow$Pluto$\}$, or ((x Pluto))
(equalto A A)	(equalto ?x ?y)	$\{$x/A, y/A$\}$
(P ?x ?x)	(P ?y ?z)	$\{$x/y, y/z$\}$
(owns Minnie ?y)	(owns ?z Pluto)	nil

Substitution in Unification

- Order doesn't matter: $\{x/y\} \equiv \{y/x\}$
- Could have more complex substitutions:
  - unify loves(x, y) with loves(Pluto,z)

Substitution in Unification

- Order doesn't matter: $\{x/y\} \equiv \{y/x\}$
- Could have more complex substitutions:
  - unify loves(x, y) with loves(Pluto,z)
  - One possibility: $\{$x/Pluto, y/z$\}$

Substitution in Unification

- Order doesn't matter: $\{$x/y$\} \equiv \{$y/x$\}$
- Could have more complex substitutions:
  - unify loves(x, y) with loves(Pluto,z)
  - One possibility: $\{$x/Pluto, y/z$\}$
  - Another: $\{$x/Pluto, y/Mickey, z/Mickey$\}$

Substitution in Unification

- Order doesn't matter: $\{$x/y$\} \equiv \{$y/x$\}$
- Could have more complex substitutions:
  - unify loves(x, y) with loves(Pluto,z)
  - One possibility: $\{$x/Pluto, y/z$\}$
  - Another: $\{$x/Pluto, y/Mickey, z/Mickey$\}$
  - Still another: $\{$x/Pluto, y/ice-cream, z/ice-cream$\}$

Substitution in Unification

- Order doesn't matter: $\{$x/y$\} \equiv \{$y/x$\}$
- Could have more complex substitutions:
  - unify loves(x, y) with loves(Pluto,z)
  - One possibility: $\{$x/Pluto, y/z$\}$
  - Another: $\{$x/Pluto, y/Mickey, z/Mickey$\}$
  - Still another: $\{$x/Pluto, y/ice-cream, z/ice-cream$\}$
- Want *most general unifier* – Don't over-commit!

Unify Algorithm

```
Unify(lit1,lit2,{blist}):
begin
    if eql(lit1,lit2) then
        return t, blist;
    elsif lit1 is a variable then
        if lit1 appears in lit2 then
            return nil, blist;
        elsif lit1 is bound in blist then
            Unify(binding(lit1,blist),lit2,blist);
        else
            return t, blist+{lit1/lit2};
        fi
    elsif lit2 is a variable then
        Unify(lit2,lit1,blist);
    elsif lit1 or lit2 are both atoms or lists of different lengths
        then return nil, blist;
    else
        match = t;
        temp-blist = blist;
        loop for i = 1 to length(lit1) do
            match,temp-blist = Unify(lit1[i],lit2[i],temp-blist);
            if match = nil then retun nil, blist;
            else apply temp-blist to remainder of lit1 and lit2;
            fi;
        end loop;
        return t, temp-blist;
    fi;
end Unify;
```

Theorem Proving

Theorem Proving as Search

- State: axioms at the current moment
- Operators:
  - Modus ponens, modus tolens, resolution
  - Apply to axiom set $\Rightarrow$ new axiom set (new state)
- Forward, backward search/proof

Example Axiom Set

1. $human(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\forall x\ human(x) \Rightarrow mortal(x)$
5. $\forall x\ Pompeian(x) \Rightarrow died(x, 79)$
6. $erupted(volcano, 79)$
7. $\forall x, t_1, t_2\ mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$
8. $now = 2014$
9. $\forall x, t\ [alive(x, t) \Rightarrow \neg dead(x, t)] \wedge [\neg dead(x, t) \Rightarrow alive(x, t)]$
10. $\forall x, t_1, t_2\ died(x, t_1) \wedge gt(t_2, t_1) \Rightarrow dead(x, t_2)$

Is Marcus dead?

- Forward proof:

Is Marcus dead?

- Forward proof:

1.	human(Marcus)	axiom 1

Is Marcus dead?

- Forward proof:

1.	human(Marcus)	axiom 1
2.	born(Marcus,40)	axiom 3

Is Marcus dead?

- Forward proof:

1. human(Marcus)	axiom 1
2. born(Marcus,40)	axiom 3
3. mortal(Marcus)	1 & axiom 4 $\forall x \; human(x) \Rightarrow mortal(x)$, {x/Marcus}

Is Marcus dead?

- Forward proof:

1. human(Marcus)	axiom 1
2. born(Marcus,40)	axiom 3
3. mortal(Marcus)	1 & axiom 4 $\forall x\ human(x) \Rightarrow mortal(x)$, {x/Marcus}
4. now = 2014	axiom 8

Is Marcus dead?

- Forward proof:

1. human(Marcus)	axiom 1
2. born(Marcus,40)	axiom 3
3. mortal(Marcus)	1 & axiom 4 $\forall x\ human(x) \Rightarrow mortal(x)$, {x/Marcus}
4. now = 2014	axiom 8
5. dead(Marcus,2014)	3 & 2 & 4 & axiom 7 $\forall x, t_1, t_2\ mortal(x) \wedge born(x, t_1) \wedge$ $gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$ {x/Marcus, t1/40, t2/now, now/2014}

Forward vs Backward Proof

- May be difficult to constrain search:
  - branching factor large
  - no direction on which branch to take
- Backward proof – easier to constrain search (usually)

Backward Proof Example

Prove: Marcus is dead.

1. $human(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\forall x\ human(x) \Rightarrow mortal(x)$
5. $\forall x\ Pompeian(x) \Rightarrow died(x, 79)$
6. $erupted(volcano, 79)$
7. $\forall x, t_1, t_2\ mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$
8. $now = 2014$
9. $\forall x, t\ [alive(x, t) \Rightarrow \neg dead(x, t)] \wedge [\neg dead(x, t) \Rightarrow alive(x, t)]$
10. $\forall x, t_1, t_2\ died(x, t_1) \wedge gt(t_2, t_1) \Rightarrow dead(x, t_2)$

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow$ $\neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow \neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow$ $\neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

$\neg$Cloudy

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. $Raining \Rightarrow Cloudy$	2. $Rainbow \Rightarrow \neg Cloudy$
3. $Rainbow$	4. $Raining$

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

$\neg$Cloudy $\neg$Cloudy $\vee$ exist(Leprechauns)	since $1 \vee A = A$

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow \neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

$\neg$Cloudy	
$\neg$Cloudy $\vee$ exist(Leprechauns)	since 1 $\vee$ A = A
Cloudy $\Rightarrow$ exist(Leprechauns)	definition of $\Rightarrow$

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow \neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

$\neg$Cloudy	
$\neg$Cloudy $\vee$ exist(Leprechauns)	since 1 $\vee$ A = A
Cloudy $\Rightarrow$ exist(Leprechauns)	definition of $\Rightarrow$
exist(Leprechauns)	Modus ponens with Cloudy

Contradictions in the Knowledge Base

- What happens if your KB is inconsistent?
- Suppose your knowledge base is:

1. Raining $\Rightarrow$ Cloudy	2. Rainbow $\Rightarrow \neg$Cloudy
3. Rainbow	4. Raining

- Is this inconsistent?
- If so, is this a problem?
  - Suppose we conclude both $\neg$Cloudy & Cloudy

$\neg$Cloudy	
$\neg$Cloudy $\vee$ exist(Leprechauns)	since 1 $\vee$ A = A
Cloudy $\Rightarrow$ exist(Leprechauns)	definition of $\Rightarrow$
exist(Leprechauns)	Modus ponens with Cloudy

If your axiom set is inconsistent, can prove anything!

Resolution Theorem Proving

Resolution Theorem Proving (RTP)

- A proof by refutation: Try to prove A by proving $\neg A$ is false
- Prove false by showing a contradiction
- Uses only one inference rule
- Repeatedly apply *resolution*:

$$(A \vee B) \wedge (\neg B \vee C) \equiv A \vee C$$

  - Need standardized knowledge base: *conjunctive normal form* or *implicative normal form*
  - Finding nil means contradiction ($A \wedge \neg A$ resolves to nil)
- Cannot use on an inconsistent knowledge base because can prove anything

Conjunctive Normal Form (CNF)

- Need to make the all clauses in the same form so easy to apply
- Clauses contain only OR's as operators
- Clauses are interpreted as ANDed together
- Use sound rules of inference, so consistency of the knowledge base remains the same

Converting a Knowledge Base to CNF

1. Eliminate implications ($\rightarrow$)
2. Reduce scope of $\neg$
3. Standardize (separate) variable names
4. Move quantifiers to the left
5. *Skolemize* existential quantifiers
6. Drop universal quantifiers
7. Change KB to conjunction of disjunctions
8. Standardize (separate) variable names (again)

Converting the Garden Example to CNF

- John likes carrots.
  Like(John, Carrots)
- Mary likes carrots.
  Like(Mary, Carrots)
- John grows the vegetables he likes.
  $\forall$ x Like(John, x) $\bigwedge$ Vegetable(x) $\longrightarrow$ Grow(John, x)
- Carrots are vegetables.
  Vegetable(Carrots)
- When you like a vegetable and you own it, you eat it.
  $\forall$ x $\forall$ y Like(x, y) $\bigwedge$ Vegetable(y) $\bigwedge$ Own(x, y) $\longrightarrow$ Eat(x, y)
- To eat something, you have to own it.
  $\forall$ x $\forall$ y Eat(x,y) $\longrightarrow$ Own(x, y)
- When you grow something, you own it.
  $\forall$ x $\forall$ y Grow(x,y) $\longrightarrow$ Own(x ,y)
- In order to grow something, you must own a garden.
  $\forall$ x $\forall$ y $\exists$ g Grow(x, y) $\longrightarrow$ Own(x, g) $\bigwedge$ Garden(g)

Eliminate Implications: $a \rightarrow b \equiv \neg a \vee b$

$\forall x \forall y \mathrm{Eat}(x,y) \rightarrow \mathrm{Own}(x,y)$	$\forall x \forall y \neg \mathrm{Eat}(x,y) \vee \mathrm{Own}(x,y)$
$\forall x \forall y \mathrm{Grow}(x,y) \rightarrow \mathrm{Own}(x,y)$	$\forall x \forall y \neg \mathrm{Grow}(x,y) \vee \mathrm{Own}(x,y)$
$\forall x \forall y \exists g \mathrm{Grow}(x,y) \rightarrow \mathrm{Own}(x,g) \wedge \mathrm{Garden}(g)$	$\forall x \forall y \exists g \neg \mathrm{Grow}(x,y) \vee [\mathrm{Own}(x,$ $\mathrm{Garden}(g)]$
$\forall x [\mathrm{Like}(\mathrm{John},x) \wedge \mathrm{Vegetable}(x)] \rightarrow \mathrm{Grow}(\mathrm{John},x)$	$\forall x \neg [\mathrm{Like}(\mathrm{John},x) \wedge \mathrm{Vegetable}(x)]$ $\vee \mathrm{Grow}(\mathrm{John},x)$
$\forall x \forall y [\mathrm{Like}(x,y) \wedge \mathrm{Vegetable}(y) \wedge \mathrm{Own}(x,y)] \rightarrow \mathrm{Eat}(x,y)$	$\forall x \forall y \neg [\mathrm{Like}(x,y) \wedge \mathrm{Vegetable}(y)$ $\mathrm{Own}(x,y)] \vee \mathrm{Eat}(x,y)$

Reduce scope of $\neg$

- Use DeMorgan's laws, $\neg(\neg p) = p$
- For quantifiers:
  - $\neg \forall x P(x) = \exists x \neg P(x)$
  - $\neg \exists x P(x) = \forall x \neg P(x)$
- $\forall x \neg[\text{Like}(\text{John}, x) \wedge \text{Vegetable}(x)] \vee \text{Grow}(\text{John}, x) \equiv$

$$\forall x \neg \text{Like}(\text{John}, x) \vee \neg \text{Vegetable}(x) \vee \text{Grow}(\text{John}, x)$$

- $\forall x \forall y \neg[\text{Like}(x, y) \wedge \text{Vegetable}(y) \wedge \text{Own}(x, y)] \vee \text{Eat}(x, y) \equiv$

$$\forall x \forall y \neg \text{Like}(x, y) \vee \neg \text{Vegetable}(y) \vee \neg \text{Own}(x, y) \vee \text{Eat}(x, y)$$

Standardize Variable Names

- Give each variable in scope of quantifier a different name
- $\forall x \forall y \neg \text{Eat}(x, y) \vee \text{Own}(x, y)$
- $\forall x_1 \forall y_1 \neg \text{Grow}(x_1, y_1) \vee \text{Own}(x_1, y_1)$
- $\forall x_2 \forall y_2 \exists g \neg \text{Grow}(x_2, y_2) \vee [\text{Own}(x_2, g) \wedge \text{Garden}(g)]$
- $\forall x_3 \neg \text{Like}(\text{John}, x_3) \vee \neg \text{Vegetable}(x_3) \vee \text{Grow}(\text{John}, x_3)$
- $\forall x_4 \forall y_4 \neg \text{Like}(x_4, y_4) \vee \text{Vegetable}(y_4) \vee \neg \text{Own}(x_4, y_4) \vee \text{Eat}(x_4, y_4)$

Move quantifiers to the left

- Names are different, so scoping is no problem
- This does not require any changes to our example knowledge base

Skolemize Existential Quantifiers

- Since $\exists\, x$ means "there exists some x", just invent a constant for it – a Skolem constant
- Generally use sk1..skn for Skolem constants
- If inside universal quantifier, use *Skolem function*: a function of that variable: e.g., sk1(x)
- $\forall x_2 \forall y_2 \exists g \neg \mathrm{Grow}(x_2, y_2) \vee [\mathrm{Own}(x_2, g) \wedge \mathrm{Garden}(g)]$

$$\equiv$$

$$\forall x_2 \forall y_2 \neg \mathrm{Grow}(x_2, y_2) \vee [\mathrm{Own}(x_2, sk(x_2, y_2)) \wedge \mathrm{Garden}(sk(x_2, y_2))]$$

Drop ∀

- Can do this, since all variables are now universally quantified
- Like(John, Carrots)
- Like(Mary, Carrots)
- Vegetable(Carrots)
- $\neg$Eat$(x, y) \lor$ Own(x, y)
- $\neg$Grow$(x_1, y_1) \lor$ Own(x_1, y_1)
- $\neg$Grow$(x_2, y_2) \lor [$Own$(x_2, sk(x_2, y_2)) \land$ Garden$(sk(x_2, y_2))]$
- $\neg$Like(John, $x_3) \lor \neg$Vegetable$(x_3) \lor$ Grow(John, x_3)
- $\neg$Like$(x_4, y_4) \lor$ Vegetable$(y_4) \lor \neg$Own$(x_4, y_4) \lor$ Eat(x_4, y_4)

Change to a conjunct of disjuncts

- Change the whole set of statements to a conjunction of disjunction by applying distributive property and dropping ANDs between disjunctive clauses
  - $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$
- $\neg\text{Grow}(x_2, y_2) \vee [\text{Own}(x_2, sk(x_2, y_2)) \wedge \text{Garden}(sk(x_2, y_2))] \equiv$
  $\neg\text{Grow}(x_2, y_2) \vee \text{Own}(x_2, sk(x_2, y_2))$
  and
  $\neg\text{Grow}(x_2, y_2) \vee \text{Garden}(sk(x_2, y_2))$

Give each variable a different name

- $\neg \text{Grow}(x_2, y_2) \vee \text{Own}(x_2, sk(x_2, y_2))$
- $\neg \text{Grow}(x_5, y_5) \vee \text{Garden}(sk(x_5, y_5))$

Algorithm for Resolution Theorem Proving

1. Convert statements to conjunctive normal form
2. Pick two clauses and "resolve" them
   - need to worry about matching variables
   - don't need to undo steps – steps are ignorable since only making sound inferences
3. If resolvent is not nil, add resolvent to KB and go to 2. Otherwise, have proved original statement by contradiction of negation of that statement

RTP as Search

- Operators:
- Choice points:
- Backtracking:
- Search strategy:
- Heuristics:

How would we use unify in resolution?

- Suppose we want to resolve W(A,B) and $\neg W(A,x) \vee S(x) \vee R(A,x)$
- Can unify W(A,B) and W(A,x) if x = B, so have substitution instance of B/x
- Using the substitution for the whole clause, we get $\neg W(A,B) \vee S(B) \vee R(A,B)$
- When resolve the two clauses, get: $S(B) \vee R(A,B)$

Unifying Two Clauses

- Predicates must match (easiest thing to eliminate on)
- Arguments must match:
  - if constant, or one in previous substitution, bound to that in the clause
  - if a variable, can try all possibilities

Resolution Theorem Proving Example

- Put knowledge base in CNF
  - $S(A, B)$
  - $S(C, B)$
  - $T(B)$
  - $\neg Q(x, y) \vee P(x, y)$
  - $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - $\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
  - $\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
  - $\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
  - $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
- Negate the clause that you are trying to prove
  - want to prove $Q(A, B)$ – add $\neg Q(A, B)$ to knowledge base
- Resolve clauses until come to nil

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

– prove $\neg Q(A, B)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \lor P(x, y)$
$\neg R(x_1, y_1) \lor P(x_1, y_1)$
$\neg R(x_2, y_2) \lor P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \lor W(sk1(x_3, y_3))$
$\neg S(A, x_4) \lor \neg T(x_4) \lor R(A, x_4)$
$\neg S(x_5, y_5) \lor \neg T(y_5) \lor \neg P(x_5, y_5)$
$\lor Q(x_5, y_5)$

– prove $\neg Q(A, B)$
– resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \lor \neg T(y_5) \lor \neg P(x_5, y_5) \lor Q(x_5, y_5)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

– prove $\neg Q(A, B)$
– resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  – substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil

Resolving on the Example

$S(A,B)$
$S(C,B)$
$T(B)$
$\neg Q(x,y) \vee P(x,y)$
$\neg R(x_1,y_1) \vee P(x_1,y_1)$
$\neg R(x_2,y_2) \vee P(x_2, sk1(x_2,y_2))$
$\neg R(x_3,y_3) \vee W(sk1(x_3,y_3))$
$\neg S(A,x_4) \vee \neg T(x_4) \vee R(A,x_4)$
$\neg S(x_5,y_5) \vee \neg T(y_5) \vee \neg P(x_5,y_5)$
$\vee Q(x_5,y_5)$

- prove $\neg Q(A,B)$
- resolve $\neg Q(A,B)$ with $\neg S(x_5,y_5) \vee \neg T(y_5) \vee \neg P(x_5,y_5) \vee Q(x_5,y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A,B) \vee \neg T(B) \vee \neg P(A,B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A,B)$

Resolving on the Example

$S(A,B)$
$S(C,B)$
$T(B)$
$\neg Q(x,y) \vee P(x,y)$
$\neg R(x_1,y_1) \vee P(x_1,y_1)$
$\neg R(x_2,y_2) \vee P(x_2, sk1(x_2,y_2))$
$\neg R(x_3,y_3) \vee W(sk1(x_3,y_3))$
$\neg S(A,x_4) \vee \neg T(x_4) \vee R(A,x_4)$
$\neg S(x_5,y_5) \vee \neg T(y_5) \vee \neg P(x_5,y_5)$
$\vee Q(x_5,y_5)$

- prove $\neg Q(A,B)$
- resolve $\neg Q(A,B)$ with $\neg S(x_5,y_5) \vee \neg T(y_5) \vee \neg P(x_5,y_5) \vee Q(x_5,y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A,B) \vee \neg T(B) \vee \neg P(A,B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A,B)$
- resolve with: T(B)

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: A/x_5, B/y_5 - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$

Resolving on the Example

$S(A,B)$
$S(C,B)$
$T(B)$
$\neg Q(x,y) \lor P(x,y)$
$\neg R(x_1,y_1) \lor P(x_1,y_1)$
$\neg R(x_2,y_2) \lor P(x_2, sk1(x_2,y_2))$
$\neg R(x_3,y_3) \lor W(sk1(x_3,y_3))$
$\neg S(A,x_4) \lor \neg T(x_4) \lor R(A,x_4)$
$\neg S(x_5,y_5) \lor \neg T(y_5) \lor \neg P(x_5,y_5)$
$\lor Q(x_5,y_5)$

- prove $\neg Q(A,B)$
- resolve $\neg Q(A,B)$ with $\neg S(x_5,y_5) \lor \neg T(y_5) \lor \neg P(x_5,y_5) \lor Q(x_5,y_5)$
  - substitutions: A/x_5, B/y_5 - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A,B) \lor \neg T(B) \lor \neg P(A,B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \lor \neg P(A,B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A,B)$
- resolve with: $\neg R(x_1,y_1) \lor P(x_1,y_1)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: A/x_5, B/y_5 - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: A/x_1, B/y_5
  - resolvent: $\neg R(A, B)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4
  - resolvent: $\neg S(A, B) \vee \neg T(B)$

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: A/x_5, B/y_5 - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: A/x_1, B/y_5
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4
  - resolvent: $\neg S(A, B) \vee \neg T(B)$
- resolve with: $S(A, B)$

Artificial Intelligence

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: A/x_5, B/y_5 - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: A/x_1, B/y_5
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4
  - resolvent: $\neg S(A, B) \vee \neg T(B)$
- resolve with: $S(A, B)$
  - substitution: nil

Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4
  - resolvent: $\neg S(A, B) \vee \neg T(B)$
- resolve with: $S(A, B)$
  - substitution: nil
  - resolvent: $\neg T(B)$

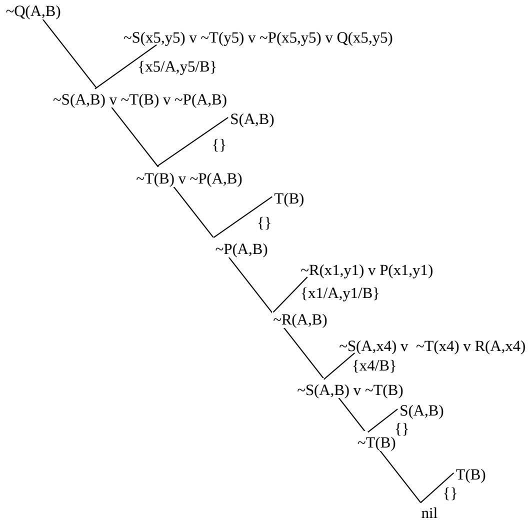
Resolving on the Example

$S(A, B)$
$S(C, B)$
$T(B)$
$\neg Q(x, y) \vee P(x, y)$
$\neg R(x_1, y_1) \vee P(x_1, y_1)$
$\neg R(x_2, y_2) \vee P(x_2, sk1(x_2, y_2))$
$\neg R(x_3, y_3) \vee W(sk1(x_3, y_3))$
$\neg S(A, x_4) \vee \neg T(x_4) \vee R(A, x_4)$
$\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5)$
$\vee Q(x_5, y_5)$

- prove $\neg Q(A, B)$
- resolve $\neg Q(A, B)$ with $\neg S(x_5, y_5) \vee \neg T(y_5) \vee \neg P(x_5, y_5) \vee Q(x_5, y_5)$
  - substitutions: $A/x_5, B/y_5$ - only looking at the Q's and then must apply throughout when resolve
  - resolvent: $\neg S(A, B) \vee \neg T(B) \vee \neg P(A, B)$
- resolve resolvent with S(A,B)
  - substitutions: nil
  - $\neg T(B) \vee \neg P(A, B)$
- resolve with: T(B)
  - substitutions: nil
  - resolvent: $\neg P(A, B)$
- resolve with: $\neg R(x_1, y_1) \vee P(x_1, y_1)$
  - substitution: $A/x_1, B/y_5$
  - resolvent: $\neg R(A, B)$
- resolve with $\neg S(A, x_4) \vee T(x_4) \vee R(A, x_4)$
  - substitution: B/x_4
  - resolvent: $\neg S(A, B) \vee \neg T(B)$
- resolve with: $S(A, B)$
  - substitution: nil
  - resolvent: $\neg T(B)$
- resolve with T(B) $\longrightarrow$ nil

Artificial Intelligence

Proof Tree


~Q(A,B)
~S(x5,y5) v ~T(y5) v ~P(x5,y5) v Q(x5,y5)
{x5/A,y5/B}
~S(A,B) v ~T(B) v ~P(A,B)
S(A,B)
{}
~T(B) v ~P(A,B)
T(B)
{}
~P(A,B)
~R(x1,y1) v P(x1,y1)
{x1/A,y1/B}
~R(A,B)
~S(A,x4) v ~T(x4) v R(A,x4)
{x4/B}
~S(A,B) v ~T(B)
S(A,B)
{}
~T(B)
T(B)
{}
nil

Another example

	FOL	CNF
1	$human(Marcus)$	
2	$Pompeian(Marcus)$	
3	$born(Marcus, 40)$	
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	
3	$born(Marcus, 40)$	
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	$\neg human(x_1) \vee mortal(x_1)$
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	$\neg human(x_1) \vee mortal(x_1)$
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	$\neg Pompeian(x_2) \vee died(x_2, 79)$
6	$erupted(volcano, 79)$	
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Artificial Intelligence

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	$\neg human(x_1) \vee mortal(x_1)$
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	$\neg Pompeian(x_2) \vee died(x_2, 79)$
6	$erupted(volcano, 79)$	$erupted(volcano, 79)$
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	$\neg human(x_1) \vee mortal(x_1)$
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	$\neg Pompeian(x_2) \vee died(x_2, 79)$
6	$erupted(volcano, 79)$	$erupted(volcano, 79)$
7	$\forall x, t_1, t_2 \quad mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	$\neg mortal(x_3) \vee \neg born(x_3, t_1) \vee \neg gt(t_2 - t_1, 150) \vee dead(x_3, t2)$
8	$now = 2014$	

Another example

	FOL	CNF
1	$human(Marcus)$	$human(Marcus)$
2	$Pompeian(Marcus)$	$Pompeian(Marcus)$
3	$born(Marcus, 40)$	$born(Marcus, 40)$
4	$\forall x \quad human(x) \Rightarrow mortal(x)$	$\neg human(x_1) \lor mortal(x_1)$
5	$\forall x \quad Pompeian(x) \Rightarrow died(x, 79)$	$\neg Pompeian(x_2) \lor died(x_2, 79)$
6	$erupted(volcano, 79)$	$erupted(volcano, 79)$
7	$\forall x, t_1, t_2 \quad mortal(x) \land born(x, t_1) \land gt(t_2 - t_1, 150) \Rightarrow dead(x, t_2)$	$\neg mortal(x_3) \lor \neg born(x_3, t_1) \lor \neg gt(t_2 - t_1, 150) \lor dead(x_3, t2)$
8	$now = 2014$	$now = 2014$

Artificial Intelligence

Another example

9 FOL: $\forall x,t\ [alive(x,t) \Rightarrow \neg dead(x,t)] \wedge [\neg dead(x,t) \Rightarrow alive(x,t)]$

10 FOL: $\forall x,t_1,t_2\ died(x,t_1) \wedge gt(t_2,t_1) \Rightarrow dead(x,t_2)$

Another example

9 FOL: $\forall x, t\ [alive(x,t) \Rightarrow \neg dead(x,t)] \wedge [\neg dead(x,t) \Rightarrow alive(x,t)]$
CNF:
$[\neg alive(x_4,t_3) \vee \neg dead(x_4,t_3)] \wedge [dead(x_4,t_3) \vee alive(x_4,t_3)]$
$\equiv$
(a)$\neg alive(x_4,t_3) \vee \neg dead(x_4,t_3)$
(b)$dead(x_5,t_4) \vee alive(x_5,t_4)$

10 FOL: $\forall x, t_1, t_2\ died(x,t_1) \wedge gt(t_2,t_1) \Rightarrow dead(x,t_2)$

Artificial Intelligence

Another example

9 FOL: $\forall x, t\ [alive(x,t) \Rightarrow \neg dead(x,t)] \wedge [\neg dead(x,t) \Rightarrow alive(x,t)]$

CNF:

$[\neg alive(x_4,t_3) \vee \neg dead(x_4,t_3)] \wedge [dead(x_4,t_3) \vee alive(x_4,t_3)]$

$\equiv$

(a)$\neg alive(x_4,t_3) \vee \neg dead(x_4,t_3)$

(b)$dead(x_5,t_4) \vee alive(x_5,t_4)$

10 FOL: $\forall x, t_1, t_2\ died(x,t_1) \wedge gt(t_2,t_1) \Rightarrow dead(x,t_2)$

CNF: $\neg died(x_6,t_5) \vee \neg gt(t_6,t_5) \vee dead(x_6,t_6)$

Marcus CNF

1. $human(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\neg human(x_1) \vee mortal(x_1)$
5. $\neg Pompeian(x_2) \vee died(x_2, 79)$
6. $erupted(volcano, 79)$
7. $\neg mortal(x_3) \vee \neg born(x_3, t_1) \vee \neg gt(t_2 - t_1, 150) \vee dead(x_3, t2)$
8. $now = 2014$
9. $\neg alive(x_4, t_3) \vee \neg dead(x_4, t_3)$
10. $dead(x_5, t_4) \vee alive(x_5, t_4)$
11. $\neg died(x_6, t_5) \vee \neg gt(t_6, t_5) \vee dead(x_6, t_6)$

Prove: *dead(Marcus)*

Control Strategies

- Only try clauses with complementary literals
- Unit preference strategy
- Set-of-support
- Eliminate clauses which cannot change value of knowledge base
  - tautologies
  - subsumed clauses
    - P(x) subsumes $P(y) \vee Q(z)$ since if P(x) is true it doesn't make any difference if Q(x) is true – assuming P(x) is true since in the knowledge base
    - P(x) subsumes P(A) since variable is more general than the constant

Properties of RTP

- Is it complete?
  - *Semi-decidable* – with appropriate control strategies (e.g., set-of-support and unit-preference)
- Time complexity?
- Space complexity?

Question Answering

- Yes/no questions
  - turn question into statement
  - if can prove, answer is “yes”
  - if can’t prove, try proving negation for “no”
- Fill in the blank questions (wh-questions)
  - use an existentially-quantified variable in the question
  - negate the question and see what variable is bound to
- Green’s trick:
  - do not negate, but mark so can distinguish from other clauses
  - when left with only clause, see what variable is bound to

Rule-based reasoning

Expert Systems

- What is an “expert system”?
- Also called *knowledge-based systems*
- *Strong* vs *weak* methods
- Feigenbaum, Shortliffe, Buchanan, J. McDermott, others: create specialists, not generalists

Characteristics

- Expert-level performance
- Clean separation of knowledge and program ("inference engine")
- Highly domain-specific, specialty very narrow
- Often: meta-knowledge
- Often: handles uncertainty
- *Highly knowledge-intensive*

Rule-based Expert Systems

- Based on *production systems* [Post, 1943]
- Rules:
  - productions: rewrite rules
  - if *condition*+ then *action*+
  - test/action pairs, antecedent/consequent, LHS/RHS
- Working memory – contains positive literals
- Control system
- *Forward chaining* of rules

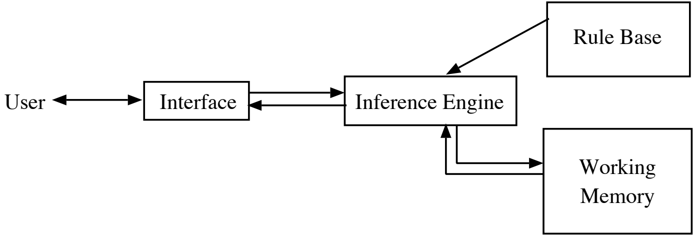
Benefits of production systems

- Equivalent to Turing machines
- Separates knowledge and program
- Modular
- Standard knowledge representation
- Simpler than full-fledged FOPC; more efficient than theorem prover
- *Physical symbol system*

Modifications to Production System

- Backward- as well as forward-chaining of rules
- Uncertainty management
  - Literals: (*predicate attribute value CF*)
    (IDENTITY $ORG1 STREPTOCOCCUS 700)
  - Rules: add a certainty associated with rule
    If it is cloudy and the barometer is falling
    Then there is suggestive evidence (.7) that it will rain
- User interface
- Meta knowledge

Modifications to Production System



Kinds of RBES

- Classified by domain

Kinds of RBES

- Classified by domain
- ...by type of task:
  - synthesis/construction
  - analysis/categorization

Kinds of RBES

- Classified by domain
- ...by type of task:
  - synthesis/construction
  - analysis/categorization
- ...by reasoning style:
  - Forward chaining
  - Backward chaining

Kinds of RBES

- Classified by domain
- ...by type of task:
  - synthesis/construction
  - analysis/categorization
- ...by reasoning style:
  - Forward chaining
  - Backward chaining
- ...by exact or probabilistic or fuzzy reasoning

Forward-Chaining RBES

Forward-Chaining RBES

- Control cycle:
  - Find rules whose antecedents are true: *triggered* rules
  - Select one: *conflict resolution*
  - *Fire* the rule to take some action
- Continue forever or until some goal is achieved
- Used for synthesis, often, or process control

Example: Winston's "Bagger" Program

- Toy forward chainer – domain = bagging groceries
- Steps in this process:
  1. Check what customer has and suggest additions
  2. Bag large items, putting large bottles in first
  3. Bag medium items, putting frozen food in freezer bags
  4. Bag small items wherever there is room
- Working memory:
  - Needs to have information about:
    - items already bagged
    - unbagged items
    - which step (context) we're in

Example: Winston's "Bagger" Program

- Representation: could be literals, could have more structure than that
- Initial state:

```
Step: check-order
Bagged: nil
Unbagged: bread, Glop brand cheese, granola,
          ice cream
```

- Also need information about the world; this might be in the form of a table for this problem:

Object	Size	Container	Frozen?
bread	M	bag	nil
Glop	S	jar	nil
granola	L	box	nil
ice cream	M	box	t
Pepsi	L	bottle	nil
potato chips	M	bag	nil

Example: Winston's "Bagger" Program

Conflict resolution strategies – possibilities:

- specificity ordering:
  - if two rules conflict and one is more specific than the other, use it
  - Rule 1 is more specific than Rule 2 if Rule 1's antecedent literals are a superset of Rule 2's (assuming conjunction)
- rule ordering – implicit in rule base (unless using a rete net)
- data ordering – look at some data first (rete does this, sort of)
- size of antecedent – prefer rules with larger antecedent, since it's likely to be more specific
- recency – least/most recently used (depending on needs of designer)
- context-limiting

Example: Winston's "Bagger" Program

- Rules in form of IF-THEN pairs
- Examples:

```
R1: if  step = check-order &
        exists bag of chips &
        not exists soft drink bottle
    then add bottle of pepsi to order

R2: if  step = check-order
    then step = bag-large-items

R3: if step = bag-large-items &
       exists large item to be bagged &
       exists large bottle to be bagged &
       exists bag with < 6 large items
    then put bottle in bag
```


Example: Winston's "Bagger" Program

- Initial state:

```
Step: check-order
Bagged: nil
Unbagged: bread, Glop brand cheese, granola,
          ice cream
```

- World info:

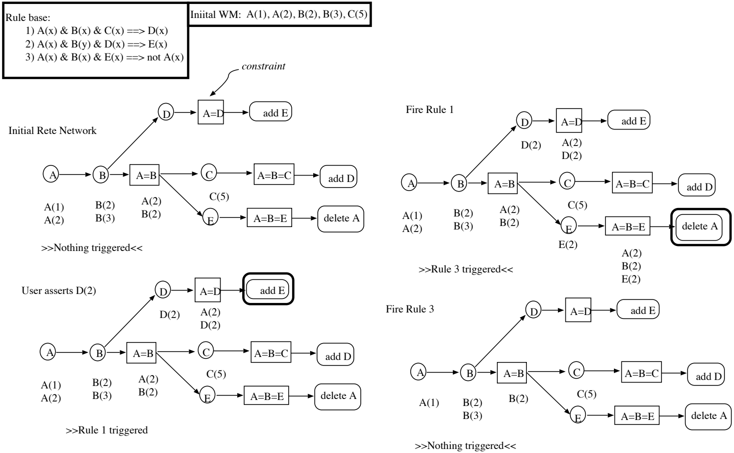
Object	Size	Container	Frozen?
bread	M	bag	nil
Glop	S	jar	nil
granola	L	box	nil
ice cream	M	box	t
Pepsi	L	bottle	nil
potato chips	M	bag	nil

Finding Triggered Rules

- Possibly very time-consuming
- Observations:
  - Rules often share LHS elements (literals)
  - Rules don't usually change over short term
  - When WM changes: usually only a few changes per cycle
- Forgy: build a *rete network* based on the rules
- Rete records state of WM, rules in network – update on change

Overview
Forward-Chaining RBES
- Overview
- Example
- Triggering
- Rete Network
Backward-Chaining RBES
Examples

Rete Network



Backward-Chaining RBES

Backward-Chaining RBES

- Synthesis: pick a solution
- Analysis: gather evidence, form best hypothesis – e.g., medical diagnosis
- Work backward from goal: focus question–asking on relevant facts, tests
- Need uncertainty management
- Follow all (relevant) lines of reasoning: no conflict resolution

How Does It Work?

- Sort of like a backward-chaining theorem prover
- Want to conclude something about x:
  - Is x in WM? Then conclude something from that.
  - Are there rules that conclude something about x? Then for each rule:
    - Try to conclude something about each antecedent (*backchain*).
    - If that's possible, fire the rule, giving some evidence for x.
  - Combine evidence for and against x.

Example: Zoo World

- Goal: id(Animal1,?x)
- Initial state 1:
  ```
  color(Animal1,tawny),
  eye-direction(Animal1,forward),
  teeth-shape(Animal1,pointed),
  eats(Animal1,meat),
  hair(Animal1), dark-spots(Animal1)
  ```
- Initial state 2:
  ```
  color(Animal1,tawny),
  eye-direction(Animal1,forward),
  teeth-shape(Animal1,pointed),
  eats(Animal1,meat),
  hair(Animal1)
  ```

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence
- Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

where $P(E) = P(E \mid H) \cdot P(H) + P(E \mid \neg H) \cdot P(\neg H)$

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence
- Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

where $P(E) = P(E \mid H) \cdot P(H) + P(E \mid \neg H) \cdot P(\neg H)$

- Example:
  - H: Joey has lung cancer
  - E: Joey smokes

$$P(lung\text{–}Ca \mid smoking) = \frac{P(smoking \mid lung\text{–}Ca) \cdot P(lung\text{–}Ca)}{P(smoking)}$$

Uncertainty Handling

- General form:

$$P(H_i \mid E) = \frac{P(E \mid H_i) \cdot P(H_i)}{\sum P(E \mid H_j) \cdot P(H_j)}$$

- And with some prior evidence E and a new observation e:

$$P(H \mid e, E) = P(H \mid e) \cdot \frac{P(E \mid e, H)}{P(E \mid e)}$$

Problems with Bayesian approach

- There are problems with Bayesian probability for expert systems (in dispute recently)
- Probabilities may be difficult to obtain
  - P(E), P(H), P(E| H) may be hard to get in general – for example, where E = cough, or H = AIDS
  - empirical evidence suggests that people are not very good at estimating probabilities [Tversky & Kahneman, e.g.]
- Size of set of probabilities needed $O(2^n)$
  - Even if we could obtain them – requires too much space
  - ...and too much time to use, and compute

Problems with Bayesian approach

- In the general case, we're interested in

$$P(H \mid E_1 \wedge E_2 \wedge \ldots \wedge E_n)$$

  which is completely impractical to get
- Also assumes that $P(H_1), P(H_2), \ldots$ are disjoint probability distributions, that is, that H_i are independent and that they cover the set of all hypotheses!
- *Bayesian nets* address many of these problems in a different formalism

A Kludge: Certainty Factors

- Approximation to probability theory
- MYCIN (e.g.): $CF[H,E] = MB[H,E] - MD[H,E]$
- Since rule only supports/denies one fact: need only one number to give CF for H given E
- One CF per literal, one per rule

Combining Certainty Factors

- Formally, when two rules give evidence about same literal:

$$MB[H, s_1 \wedge s_2] = 0 \text{ if } MD = 1,$$

$$MB[H, s_1] + MB[H, s_2] \cdot (1 - MB[H, s_1])$$

- Similarly for MD
- Simple update function!

Example

- Rule A: If x then s_1
  Rule B: If y then s_2
  Rule C: If s_1 then H
  Rule D: If s_2 then H
- suppose $MB[H, s_1] = 0.3$, $MD = 0 \Rightarrow CF = 0.3$
- now rule B fires, giving $MB[H, s_2]$ as, say, 0.2:

$$MB[H, s_1 \wedge s_2] = 0.3 + 0.2 \cdot 0.7 = 0.44$$

$$MD = 0$$

$$CF = 0.44$$

Certainty Factors

- How to compute $CF(A \wedge B)$ for rule antecedents?

$$MB[H_1 \wedge H_2, E] = \min(MB[H_1, E], MB[H_2, E]$$

and for $CF(A \vee B)$:

$$MB[H_1 \wedge H_2, E] = \max(MB[H_1, E], MB[H_2, E]$$

Certainty Factors

- How to update certainty based on rule firing?
  - Two things to consider: MB/MD in antecedents (computed as above) and the CF of the rule:

$$MB[H,S] = MB'[H,S] \cdot \max(0, CF[S,E])$$

    where $MB'[H,S]$ is how much you'd believe S if E were completely believed (i.e., the rule CF), and $CF[S,E]$ is the certainty you have in S given all the evidence.
  - Essentially: you multiply the CF of the rule times the CF of the evidence

Certainty Factors

- More recently (1986), it's been found that CFs aren't in conflict with basic probability theory
- Why, then, do they work and Bayesian techniques seem not to?

Certainty Factors

- More recently (1986), it's been found that CFs aren't in conflict with basic probability theory
- Why, then, do they work and Bayesian techniques seem not to?
  - Heuristics
  - They assume rule independence – conditional probabilities are 0
  - The knowledge engineer has to ensure this
  - Leads to compound antecedents, but...
  - ...makes it tractable and modular
- Many recent expert systems are based on *Bayesian networks*

Example Expert Systems

- DENDRAL
- R1/XCON [J. McDermott] – DEC
- MYCIN, EMYCIN, ONCOCIN, PUFF, VM, CENTAUR, MDX, MDX2,...
- Blackboard systems

Description Logic

Description logics

- Logic:
  - very general, good semantics, but:
  - cumbersome
  - intractable, not decidable
- Frames and semantic nets ("network representations"):
  - specialized reasoning, intuitive, but:
  - semantics lacking/inconsistent
- Brachman's KL-ONE system: attempted to add rigor to network representations
- Gave rise to what is now called *description logics*

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., `Car`, `Human`, etc.
  - equivalent to: $\texttt{Car}(x)$, etc., in FOL

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., `Car`, `Human`, etc.
  - equivalent to: $\mathtt{Car}(x)$, etc., in FOL
- Roles:
  - Like slots in frames
  - E.g., `hasChildren`

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., `Car`, `Human`, etc.
  - equivalent to: $\mathtt{Car}(x)$, etc., in FOL
- Roles:
  - Like slots in frames
  - E.g., `hasChildren`
- Complex (compound) concepts:
  - Built by composition from other concepts and roles
  - Often *intersection of concepts* ($\sqcap$) as operator
  - Different composition operators $\Rightarrow$ different logics

Tbox and Abox

- Knowledge in a DL system divided into two "boxes"
- *Tbox* (terminological box):
  - definitions – the ontology, i.e.
  - consists of concepts – e.g., `Human`
  - relatively static across problems

Tbox and Abox

- Knowledge in a DL system divided into two “boxes”
- *Tbox* (terminological box):
  - definitions – the ontology, i.e.
  - consists of concepts – e.g., `Human`
  - relatively static across problems
- *Abox* (assertion box):
  - facts about current problem
  - instances of concepts – e.g., `Human(Roy)`
  - dynamic across, even within problems

Tbox Examples

- Woman:

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

Tbox Examples

- Woman:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

- Parent:

$$\texttt{Parent} \equiv \texttt{Person} \sqcap \exists \texttt{hasChild.Person}$$

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

$$\mathtt{Mother} \equiv \mathtt{Parent} \sqcap \mathtt{Woman}$$

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

$$\mathtt{Mother} \equiv \mathtt{Parent} \sqcap \mathtt{Woman}$$

- Students who take COS 470:

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

$$\mathtt{Mother} \equiv \mathtt{Parent} \sqcap \mathtt{Woman}$$

- Students who take COS 470:

$$\mathtt{Student} \sqcap \exists \mathtt{classSchedule.}(\exists \mathtt{contains.COS470})$$

Abox Examples

- Joe is Harry's son:

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\texttt{Professor(Roy)}$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\mathtt{Professor(Roy)}$$

$$\mathtt{Person(Roy)} \sqcap \mathtt{hasRole(Roy,Professor)}$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\texttt{Professor(Roy)}$$

$$\texttt{Person(Roy)} \sqcap \texttt{hasRole(Roy,Professor)}$$

$$(\texttt{Person} \sqcap \exists\texttt{hasRole.Professor})(\texttt{Roy})$$

Counting

- Some logics can count, too
- E.g.: "A mother with two female and at least one male children":

Counting

- Some logics can count, too
- E.g.: “A mother with two female and at least one male children”:

$$\texttt{Mother} \sqcap = 2(\texttt{hasChild.Female}) \sqcap \geq 1(\texttt{hasChild.Male})$$

Inference in DL

- Reasoning in DL systems occurs in context of Tbox and Abox
- Tbox reasoning: *subsumption*
  - Is concept $A \sqsubseteq$ concept B?
  - E.g.:

$$
\begin{aligned}
\texttt{Mother} &\equiv \texttt{Person} \sqcap \texttt{Female} \sqcap \exists \texttt{hasChild.Person} \\
\texttt{Parent} &\equiv \texttt{Person} \sqcap \exists \texttt{hasChild.Person} \\
\texttt{Mother} &\sqsubseteq \texttt{Parent}
\end{aligned}
$$

  - Can be much more complicated and indirect
- Abox reasoning: *classification*
  - Is A an instance of concept B?
- Often other kinds of reasoning, too

Different DLs

- DL really comprised of a family of logics
- Basic is $\mathcal{AL}$ (ascription language)
- Add other operators, get new languages – e.g., $\mathcal{ALU}$ would be $\mathcal{AL}$ plus union, etc.
- Simple DLs: decidable, (relatively) efficient inferences
- More expressive DLs: give up efficiency, even decidability

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

- (From R&N) Men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments:

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

- (From R&N) Men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments:

```
And(Man,AtLeast(3,Son),AtMost(2,Daughter),
    All(Son,And(Unemployed, Married,
        All(Spouse,Doctor))),
    All(Daughter,And(Professor,
        Fills(Department,Physics,Math))))
```

Uses

- General-purpose knowledge representation
- Natural language processing
- Reasoning in intelligent databases: entity-relation models
- Web Ontology Language (OWL):
  - Part of *semantic Web*
  - Associate machine-understandable semantics with Web pages
  - One language is OWL-DL
  - Complete and decidable

Local DL example: Orca

Example Orca DL

```
-----------------------------------------
Definition=(SOME expectsPresenceOf Salinity)
Certainty=0.401
-----------------------------------------
Definition=(SOME expectsPresenceOf OceanSurface)
Certainty=0.436
-----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Thruster (SOME hasAdvisedValue ShoreBased)))
Certainty=0.769
-----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Location
                      (SOME hasNumber
                            (AND Float
                                 (D-FILLER hasNumericValue
```

```
                                  (D-LITERAL 19.115639 (D-BASE-TYPE float)))
                                 (D-FILLER hasUnitOfMeasure
                                  (D-LITERAL somerandomstring
                                   (D-BASE-TYPE string)))))
                      (SOME hasNumber
                            (AND Integer
                                 (D-FILLER hasNumericValue
                                  (D-LITERAL 31 (D-BASE-TYPE integer)))
                                 (D-FILLER hasUnitOfMeasure
                                  (D-LITERAL somerandomstring
                                   (D-BASE-TYPE string)))))))
Certainty=0.482
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Survey (SOME hasDegreeExpected Mine)
                      (SOME definesGoal ActiveMission)))
Certainty=0.125
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND DetectSubmarine
                      (D-FILLER hasEventDescription
```

```
                        (D-LITERAL somerandomstring
                         (D-BASE-TYPE
                          http://www.w3.org/2001/XMLSchema#string)))))
Certainty=0.243
----------------------------------------
Definition=(SOME hasFuzzyFeature
                 (AND Danger
                      (SOME hasFuzzyMembershipFunction
                            (AND TrapezoidalFunction
                                 (SOME hasLocalMaxAt Number)
                                 (SOME hasLocalMaxAt
                                       (AND Float
                                            (D-FILLER hasNumericValue
                                             (D-LITERAL 24.848389
                                              (D-BASE-TYPE
                                               http://www.w3.org/2001/XMLSchema#floa
                                            (D-FILLER hasUnitOfMeasure
                                             (D-LITERAL somerandomstring
                                              (D-BASE-TYPE
                                               http://www.w3.org/2001/XMLSchema#str
                                 (SOME hasLocalMinAt Number)
```

```
                                    (SOME hasLocalMinAt
                                          (AND Integer
                                               (D-FILLER hasNumericValue
                                                (D-LITERAL 5
                                                 (D-BASE-TYPE
                                                  http://www.w3.org/2001/XMLSchema#inte
                                               (D-FILLER hasUnitOfMeasure
                                                (D-LITERAL somerandomstring
                                                 (D-BASE-TYPE
                                                  http://www.w3.org/2001/XMLSchema#stri
Certainty=0.334
----------------------------------------
Definition=(AND (SOME hasActivePeriod EnteringContext)
                (SOME hasOperationalSetting
                      (AND SelfDepth (SOME hasAdvisedValue Medium))))
Certainty=0.943
----------------------------------------
Definition=(AND
            (SOME definesGoal
                  (AND SamplingComplete
                       (D-FILLER hasEventDescription
```

```
                      (D-LITERAL somerandomstring
                       (D-BASE-TYPE
                        http://www.w3.org/2001/XMLSchema#string)))))
            (SOME hasCost Medium) (SOME hasDegreeExpected High)
            (SOME hasImportance High)
            (SOME isAchievedBy (AND Maneuver (SOME hasActor PeerAgent))))
Certainty=0.559
----------------------------------------
Definition=(AND
            (SOME respondsWithAction
                  (AND CommunicateStatus
                       (SOME hasObject
                             (AND NavigationComputer
                                  (SOME hasCost
                                        (AND SelfBatteryLevel
                                             (SOME hasStateValue Medium)))))
                       (SOME hasActor AdversaryAgent)
                       (SOME isSampleTargetOf PeerAgent)))
            (SOME hasImportance Medium)
            (SOME handlesEvent
                  (AND SensorFailure
```

```
                        (D-FILLER hasEventDescription
                         (D-LITERAL somerandomstring
                          (D-BASE-TYPE
                           http://www.w3.org/2001/XMLSchema#string))))))
Certainty=0.124
----------------------------------------
Definition=(AND
            (SOME handlesEvent
                  (AND PowerFailure
                       (SOME hasStateValue
                             (AND ThrusterFailure
                                  (D-FILLER hasEventDescription
                                   (D-LITERAL somerandomstring
                                    (D-BASE-TYPE
                                     http://www.w3.org/2001/XMLSchema#string)))))))
            (SOME hasImportance Low)
            (SOME respondsWithAction
                  (AND MaintainPosition (SOME hasActor Agent))))
Certainty=0.904
----------------------------------------
Definition=(SOME definesAction
```

```
                  (AND Thruster
                       (SOME hasObject
                             (AND PeerAgent (SOME hasNumber Targeted)))
                       (SOME hasSpeed AdversaryAgent)))
Certainty=0.655
----------------------------------------
Definition=(SOME definesAction
                  (AND MaintainPosition
                       (SOME hasDirection
                             (AND Number (SOME handlesEvent Submarine)))
                       (SOME hasSpeed
                             (AND Float
                                  (SOME hasObject
                                        (AND Navigate
                                             (SOME hasActor AdversaryAgent)))))
                       (SOME definesGoal Thruster)))
Certainty=0.117
```