First Order Logic

Beyond Propositional logic

- Propositional logic not expressive enough
  - In Wumpus world we needed to explicitly write every case of Breeze & Pit relation
  - Facts = propositions
  - “All squares next to pits are breezy”
- “Regular” programming languages mix facts (data) and procedures (algorithms)
  - World[2,2]=Pit
  - Cannot deduce/compose facts automatically
  - Declarative vs. Procedural

Natural Language

- Natural language probably not used for representation
  - Used for communication
  - “Look!”

First-Order Logic

- Idea:
  - Don’t treat propositions as “atomic” entities.
- First-Order Logic:
  - Objects: cs4701, fred, ph219, emptylist …
  - Relations/Predicates: is_Man(fred), Located(cs4701, ph219), is_kind_of(apple, fruit)…
    - Note: Relations typically correspond to verbs
  - Functions: Best_friend(), beginning_of() : Returns object(s)
  - Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$
  - Quantifiers:
    - Universal: $\forall x$: (is_Man(x)) is_Mortal(x))
    - Existential: $\exists y$: (is_Father(y, fred))

Predicates

- In traditional grammar, a **predicate** is one of the two main parts of a sentence the other being the subject, which the predicate modifies.
- "John is yellow" *John* acts as the subject, and *is yellow* acts as the predicate.
- The predicate is much like a verb phrase.
- In linguistic semantics a **predicate** is an expression that can be *true of* something

Wikipedia

Types of formal mathematical logic

- Propositional logic
  - Propositions are interpreted as true or false
  - Infer truth of new propositions
- First order logic
  - Contains predicates, quantifiers and variables
    - E.g. Philosopher(a) → Scholar(a)
    - $\forall$x, King(x) $\wedge$ Greedy (x) $\Rightarrow$ Evil (x)
  - Variables range over individuals (domain of discourse)
- Second order logic
  - Quantify over predicates and over sets of variables

Other logics

- Temporal logic
    - Truths and relationships change and depend on time
- Fuzzy logic
    - – Uncertainty, contradictions

Wumpus

- Squares neighboring the wumpus are smelly
  - Objects: Wumpus, squares
  - Property: Smelly
  - Relation: neighboring
- Evil king john rules England in 1200
  - Objects: John, England, 1200
  - Property: evil, king
  - Relation: ruled

Example: Representing Facts in First-Order Logic

1. Lucy* is a professor
2. All professors are people.
3. John is the dean.
4. Deans are professors.
5. All professors consider the dean a friend or don’t know him.
6. Everyone is a friend of someone.
7. People only criticize people that are not their friends.
8. Lucy criticized John .

* Name changed for privacy reasons.

Same example, more formally

Knowledge base:

- is-prof(lucy)
- $\forall$ x (is-prof(x) $\rightarrow$ is-person(x))
- is-dean(John)
- $\forall$ x (is-dean(x) $\rightarrow$ is-prof(x))
- $\forall$ x ($\forall$ y (is-prof(x) $\wedge$ is-dean(y) $\rightarrow$ is-friend-of(y,x) $\vee$ $\neg$knows(x, y)))
- $\forall$ x ($\exists$ y (is-friend-of (y, x)))
- $\forall$ x ($\forall$ y (is-person(x) $\wedge$ is-person(y) $\wedge$ criticize (x,y) $\rightarrow$ $\neg$is-friend-of (y,x)))
- criticize(lucy, John)

Question: Is John no friend of Lucy?

$\neg$is-friend-of(John ,lucy)

How the machine “sees” it:

Knowledge base:

- $P1(A)$
- $\forall x\, (P1(x) \rightarrow P3(x))$
- $P4(B)$
- $\forall x\, (P4(x) \rightarrow P1(x))$
- $\forall x\, (\forall y\, (P1(x) \wedge P4(y) \rightarrow P2(y,x) \vee \neg P5(x, y)))$
- $\forall x\, (\exists y\, (P2(y, x)))$
- $\forall x\, (\forall y\, (P3(x) \wedge P3(y) \wedge P6(x,y) \rightarrow \neg P2(y,x)))$
- $P6(A, B)$

Question: $\neg P2(B, A)$?

Knowledge Engineering

1. Identify the task.
2. Assemble the relevant knowledge.
3. Decide on a vocabulary of predicates, functions, and constants.
4. Encode general knowledge about the domain.
5. Encode a description of the specific problem instance.
6. Pose queries to the inference procedure and get answers.
7. Debug the knowledge base.

Knowledge Engineering

1. All professors are people.
2. Deans are professors.
3. All professors consider the dean a friend or don’t know him.
4. Everyone is a friend of someone.
5. People only criticize people that are not their friends.

General Knowledge

6. Lucy* is a professor
7. John is the dean.
8. Lucy criticized John.

Specific problem

9. Is John a friend of Lucy’s?

Query

Inference Procedures: Theoretical Results

- There exist complete and sound proof procedures for propositional and FOL.
  - Propositional logic
    - Use the definition of entailment directly. Proof procedure is exponential in *n*, the number of symbols.
    - In practice, can be much faster…
    - Polynomial-time inference procedure exists when KB is expressed as **Horn clauses**: $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$

      where the P_i and *Q* are non-negated atoms.
  - First-Order logic
    - Godel's completeness theorem showed that a proof procedure exists…
    - But none was demonstrated until Robinson's 1965 *resolution algorithm*.
    - Entailment in first-order logic is *semidecidable*.

Types of inference

- Reduction to propositional logic
  - Then use propositional logic inference, e.g. enumeration, chaining
- Manipulate rules directly

Universal Instantiation

- $\forall x$, King(x) $\wedge$ Greedy (x) $\Rightarrow$ Evil (x)
  - King(John) $\wedge$ Greedy (John) $\Rightarrow$ Evil (John)
  - King(Richard) $\wedge$ Greedy (Richard) $\Rightarrow$ Evil (Richard)
  - King(Father(John)) $\wedge$ Greedy (Father(John)) $\Rightarrow$ Evil (Father(John))
- Enumerate all possibilities
  - All must be true

Existential Instantiation

- $\exists$ x, Crown(x) $\wedge$ OnHead(x, John)
  - Crown (C) $\wedge$ OnHead(C, John)
  - Provided C is not mentioned anywhere else
- Instantiate the one possibility
  - One must be true
  - Skolem Constant (skolemization)

Resolution Rule of Inference

Example:

$$\begin{array}{lll} \text{Assume:} & E_1 \vee E_2 & \text{playing tennis or raining} \\ \text{and} & \neg E_2 \vee E_3 & \text{not raining or working} \\ \hline \text{Then:} & E_1 \vee E_3 & \text{playing tennis or working} \end{array}$$

“Resolvent”

General Rule:

$$\begin{array}{ll} \text{Assume:} & E \vee E_{12} \vee \ldots \vee E_{1k} \\ \text{and} & \neg E \vee E_{22} \vee \ldots \vee E_{2l} \\ \hline \text{Then:} & E_{12} \vee \ldots \vee E_{1k} \vee E_{22} \vee \ldots \vee E_{2l} \end{array}$$

Note: E_{ij} can be negated.

Algorithm: Resolution Proof

- Negate the original theorem to be proved, and add the result to the knowledge base.
- Bring knowledge base into conjunctive normal form (CNF)
  - CNF: conjunctions of disjunctions
  - Each disjunction is called a clause.
- Repeat until there is no resolvable pair of clauses:
  - Find resolvable clauses and resolve them.
  - Add the results of resolution to the knowledge base.
  - If NIL (empty clause) is produced, stop and report that the (original) theorem is true.
- Report that the (original) theorem is false.

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$

$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan

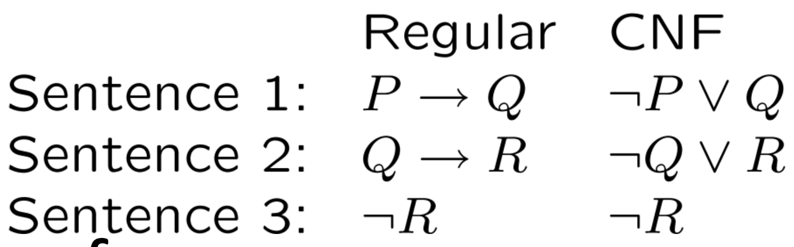
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$

Resolution Example: Propositional Logic

- To prove: ¬P
- Transform Knowledge Base into CNF



	Regular	CNF
Sentence 1:	$P \rightarrow Q$	$\neg P \lor Q$
Sentence 2:	$Q \rightarrow R$	$\neg Q \lor R$
Sentence 3:	$\neg R$	$\neg R$

- Proof

1.	$\neg P \lor Q$	Sentence 1
2.	$\neg Q \lor R$	Sentence 2
3.	$\neg R$	Sentence 3
4.	*P*	Assume opposite
5.	*Q*	Resolve 4 and 1
6.	*R*	Resolve 5 and 2
7.	nil	Resolve 6 with 3
8.	Therefore original theorem (¬P) is true	

Resolution Example: FOL

Axioms:	Regular	CNF
	$\forall x : feathers(x) \rightarrow bird(x)$	$\neg feathers(x) \vee bird(x)$
	$feathers(tweety)$	$feathers(tweety)$

Is bird(tweety)?

A: True **B: False**

Resolution Example: FOL

Example: Prove *bird(tweety)*

Axioms:	Regular	CNF
1:	$\forall x : feathers(x) \rightarrow bird(x)$	$\neg feathers(x) \vee bird(x)$
2:	$feathers(tweety)$	$feathers(tweety)$
3:	$\neg bird(tweety)$	$\neg bird(tweety)$
4:		$\neg feathers(tweety)$

Resolution Proof

1. Resolve 3 and 1, specializing (i.e. “unifying”) tweety for x.
   Add :feathers(tweety)
2. Resolve 4 and 2. Add NIL.

Resolution Theorem Proving

Properties of Resolution Theorem Proving:

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

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

Pigeon-Hole (PH) problem: you cannot place n+1 pigeons in n holes (one per hole)

A note on negation

- To prove theorem θ we need to show it is never wrong:
  - we test if there is an instance that satisfies $\neg\theta$
  - if so report that θ is false
- But we are not proving that $\neg\theta$ is true
  - Just that θ is false
  - Showing instance of $\neg\theta$ is not the same as showing that $\neg\theta$ is *always* true
- E.g. prove theorem θ that says “$x+y=4 \rightarrow x=2 \wedge y=2$”
  - We find a case $x=1 \wedge y=3$ so theorem is not true
  - But $\neg\theta$ is also not always true either

Substitutions

- Syntax:
  - SUBST (A/B, q) or SUBST (θ, q)
- Meaning:
  - Replace All occurrences of “A” with “B” in expression “q”
- Rules for substitutions:
  - Can replace a variable by a constant.
  - Can replace a variable by a variable.
  - Can replace a variable by a function expression, as long as the function expression does not contain the variable.

$$v_1/C; v_2/v_3; v_4/f(...)$$

Generalize Modus Ponens

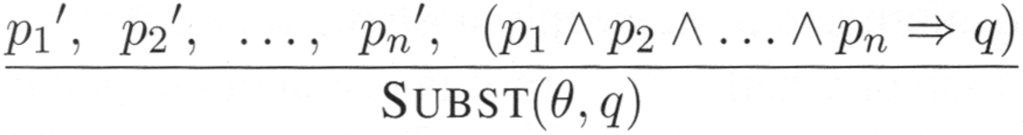
- Given n+1 terms: p_1’… p_n’, ($p_1 \wedge … \wedge p_n \Rightarrow$ ***q***)
- And a substitution θ that makes corresponding terms identical

$$\textsc{Subst}(\theta, p_i{}') = \textsc{Subst}(\theta, p_i), \text{ for all } i,$$

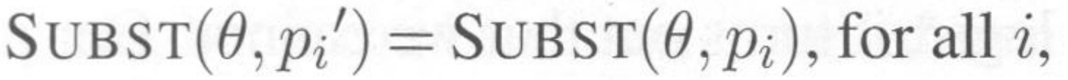
- Then apply the substitute also to ***q***

$$\frac{p_1{}',\ \ p_2{}',\ \ \ldots,\ \ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\textsc{Subst}(\theta, q)}$$

Generalize Modus Ponens

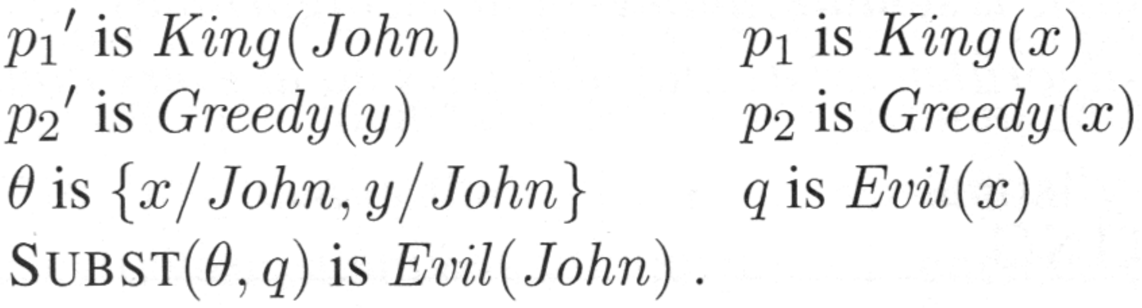


$$\frac{p_1{}',\ \ p_2{}',\ \ \ldots,\ \ p_n{}',\ \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{\textsc{Subst}(\theta, q)}$$



$$\textsc{Subst}(\theta, p_i{}') = \textsc{Subst}(\theta, p_i), \text{ for all } i,$$

- Example



$$\begin{array}{ll}
p_1{}' \text{ is } King(John) & p_1 \text{ is } King(x) \\
p_2{}' \text{ is } Greedy(y) & p_2 \text{ is } Greedy(x) \\
\theta \text{ is } \{x/John, y/John\} & q \text{ is } Evil(x) \\
\textsc{Subst}(\theta, q) \text{ is } Evil(John)\ . &
\end{array}$$

Unification

Unify procedure: Unify(P,Q) takes two atomic (i.e. single predicates) sentences P and Q and returns a substitution that makes P and Q identical.

Unifier: a substitution that makes two clauses resolvable.

Unification

- Knows(John,x), Knows(John, Jane)
  - {x/Jane)
- Knows(John,x), Knows(y, Bill)
  - {x/Bill,y/John)

Unification

- Which unification solves

  Knows(John,x), Knows(y, Mother(y))

- Choose
  - A: {y/Bill,x/Mother(John))
  - B: {y/John,x/Bill)
  - C: {y/John,x/Mother(John))
  - D: {y/Mother(John),x/John)
  - E: {y/John,x/Mother(y))

Unification

- Knows(John,x), Knows(John, Jane)
  - {x/Jane)
- Knows(John,x), Knows(y, Bill)
  - {x/Bill,y/John)
- Knows(John,x), Knows(x, Bill)
  - Fail
- Knows(John,x), Knows(z, Bill)
  - {x/Bill,z/John)
  - Standardizing apart

Unification - Purpose

Given:

$\neg$*Knows* (*John, x*) $\vee$ *Hates* (*John, x*)

Knows (*John, Jim*)

Derive:

Hates (*John, Jim*)

Unification:

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

Need unifier {*x/Jim*} for resolution to work.

Add to knowledge base:

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

Unification (example)

Who does John hate?

$\exists$ *x: Hates* (*John*, *x*)

Knowledge base (in clause form):

1. $\neg$*Knows* (*John*, *v*) $\vee$ *Hates* (*John*, *v*)
2. *Knows* (*John*, *Jim*)
3. *Knows* (*y*, *Leo*)
4. *Knows* (*z*, *Mother*(*z*))
5. $\neg$*Hates* (*John*, *x*) (since $\neg$ $\exists$*x: Hates* (*John*, *x*) $\Leftrightarrow$ $\forall$ x: $\neg$Hates(John,x))

   Resolution with 5 and 1:

   unify(*Hates*(*John*, *x*), *Hates*(*John*, *v*)) = {x/v}
6. $\neg$*Knows* (*John*, v)

   Resolution with 6 and 2:

   unify(*Knows*(*John*, v), *Knows*(*John*, *Jim*))= {v/Jim}

   or resolution with 6 and 3:

   unify(*Knows* (*John*, v), *Knows* (y, Leo)) = {y/John, v/Leo}

   or Resolution with 6 and 4:

   unify(*Knows* (*John*, v), *Knows* (z, Mother(z))) = {z/John, v/Mother(z)}

Answers:

1. Hates(John,x) with {x/v, v/Jim} (i.e. John hates Jim)
2. Hates(John,x) with {x/v, y/John, v/Leo} (i.e. John hates Leo)
3. Hates(John,x) with {x/v, v/Mother(z), z/John} (i.e. John hates his mother)

Most General Unifier

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

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

= {*y* / *John*, *x* / *z*}

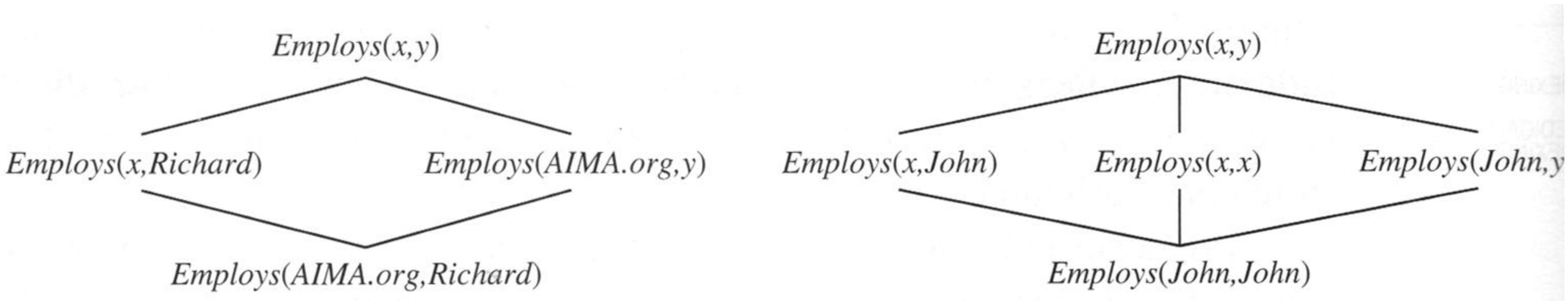
not {*y* / *John*, *x* / *John*, *z* / *John*}

not {*y* / *John*, *x* / *z*, *z* / *Freda*}

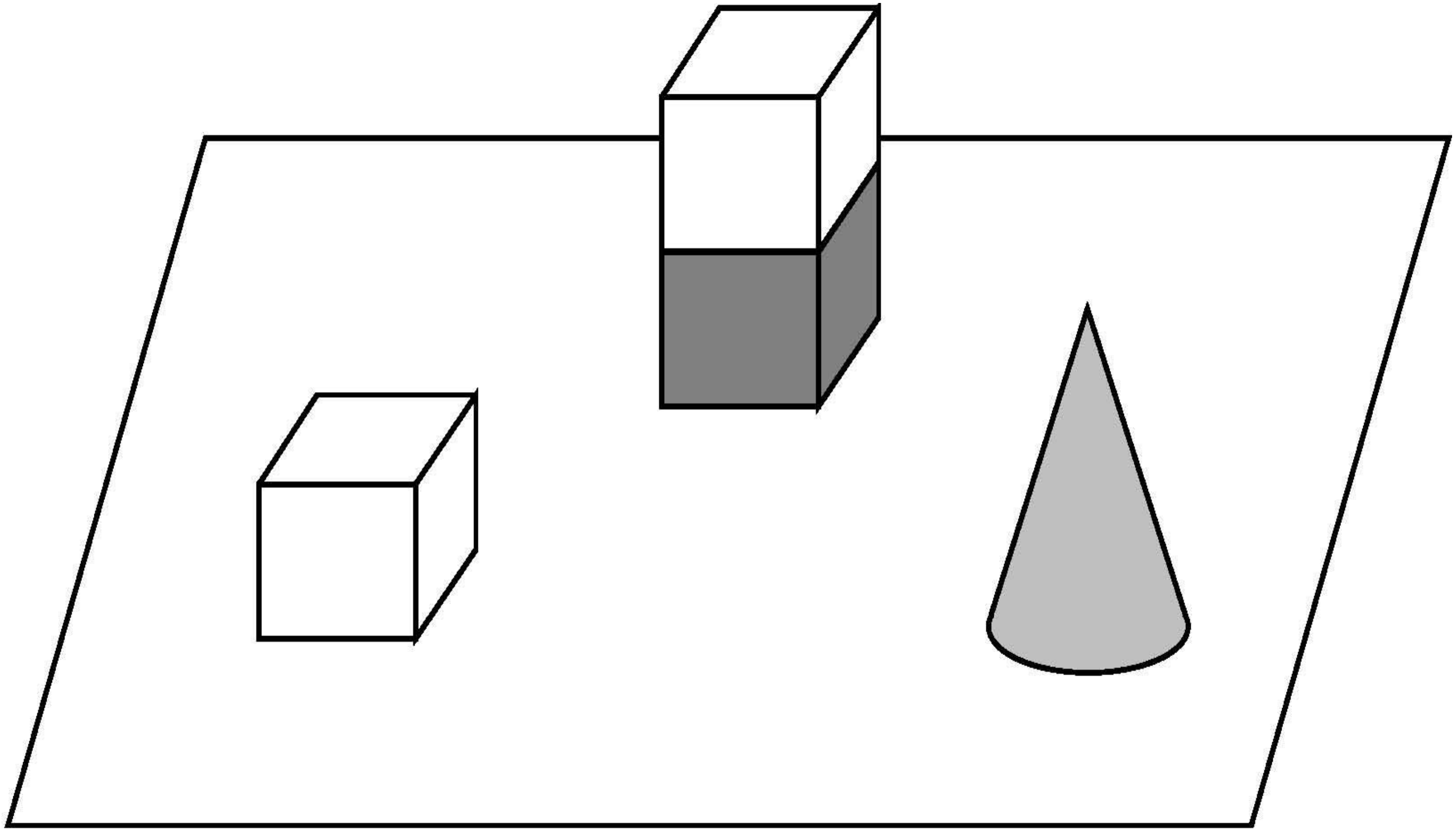
…

See R&N for general unification algorithm. O(n^2) with Refutation

Subsumption Lattice



Blocksworld



Converting More Complicated Sentences to CNF

Sentence:

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

1. First, bricks are on something else that is not a pyramid;
2. Second, there is nothing that a brick is on and that is on the brick as well.
3. Third, there is nothing that is not a brick and also is the same thing as the brick.

CNF:

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

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

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

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

Algorithm: Putting Axioms into Clausal Form

1. Eliminate the implications.
2. Move the negations down to the atomic formulas.
3. Eliminate the existential quantifiers.
4. Rename the variables, if necessary.
5. Move the universal quantifiers to the left.
6. Move the disjunctions down to the literals.
7. Eliminate the conjunctions.
8. Rename the variables, if necessary.
9. Eliminate the universal quantifiers.

1. Eliminate Implications

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

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

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

2. Move negations down to the atomic formulas

Equivalence Transformations:

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

Result:

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

3. Eliminate Existential Quantifiers: Skolemization

Harder cases:

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

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

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

Easy case:

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

4. Rename variables as necessary

We want no two variables of the same name.

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

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

5. Move the universal quantifiers to the left

This works because each quantifier uses a unique variable name.

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

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

6. Move disjunctions down to the literals

$$E_1 \lor (E_2 \land E_3) \iff (E_1 \lor E_2) \land (E_1 \lor E_3)$$

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

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

7. Eliminate the conjunctions

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

$$\forall x : \neg brick(x) \vee on(x, S1(x))$$
$$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$$
$$\forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)$$
$$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)$$

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

$$\forall x : \neg brick(x) \vee on(x, S1(x))$$
$$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$$
$$\forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)$$
$$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)$$

$$\forall x : \neg brick(x) \vee on(x, S1(x))$$
$$\forall w : \neg brick(w) \vee \neg pyramid(S1(w))$$
$$\forall u \forall y : \neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$
$$\forall v \forall z : \neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

9. Eliminate the universal quantifiers

$$\neg brick(x) \vee on(x, S1(x))$$
$$\neg brick(w) \vee \neg pyramid(S1(w))$$
$$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$
$$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

Algorithm: Putting Axioms into Clausal Form

1. Eliminate the implications.
2. Move the negations down to the atomic formulas.
3. Eliminate the existential quantifiers.
4. Rename the variables, if necessary.
5. Move the universal quantifiers to the left.
6. Move the disjunctions down to the literals.
7. Eliminate the conjunctions.
8. Rename the variables, if necessary.
9. Eliminate the universal quantifiers.

Resolution Proofs as Search

- Search Problem
  - States: Content of knowledge base in CNF
  - Initial state: Knowledge base with negated theorem to prove
  - Successor function: Resolution inference rule with unify
  - Goal test: Does knowledge base contain the empty clause 'nil'
- Search Algorithm
  - Depth first search (used in PROLOG)
    - Note: Possibly infinite state space
    - Example:
      - IsPerson(Fred)
      - IsPerson(y) → IsPerson(mother(y))
      - Goal: ∃ x: IsPerson(x)
      - Answers: {x/Fred} and {x/mother(Fred)} and {x/mother(mother(Fred))} and …

Strategies for Selecting Clauses

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

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

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

May still require exponential time.

Example

Jack owns a dog.

Every dog owner is an animal lover.

No animal lover kills an animal.

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

Did Curiosity kill the cat?

Original Sentences (Plus Background Knowledge)

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$

2. $\forall x;\ \ (\exists y\ \ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$

3. $\forall x;\ \ \ AnimalLover(x) \rightarrow (\forall y\ \ \ Animal(y) \rightarrow \neg Kills(x, y))$

4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

5. $Cat(Tuna)$

6. $\forall x :\ \ \ Cat(x) \rightarrow Animal(x)$

Conjunctive Normal Form

$$Dog(D)$$

(D is a placeholder for the dogs unknown name (i.e. Skolem symbol/function). Think of D like "JohnDoe")

$$Owns(Jack, D)$$

$$\neg Dog(y) \vee \neg Owns(x, y) \vee AnimalLover(x)$$

$$\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$$

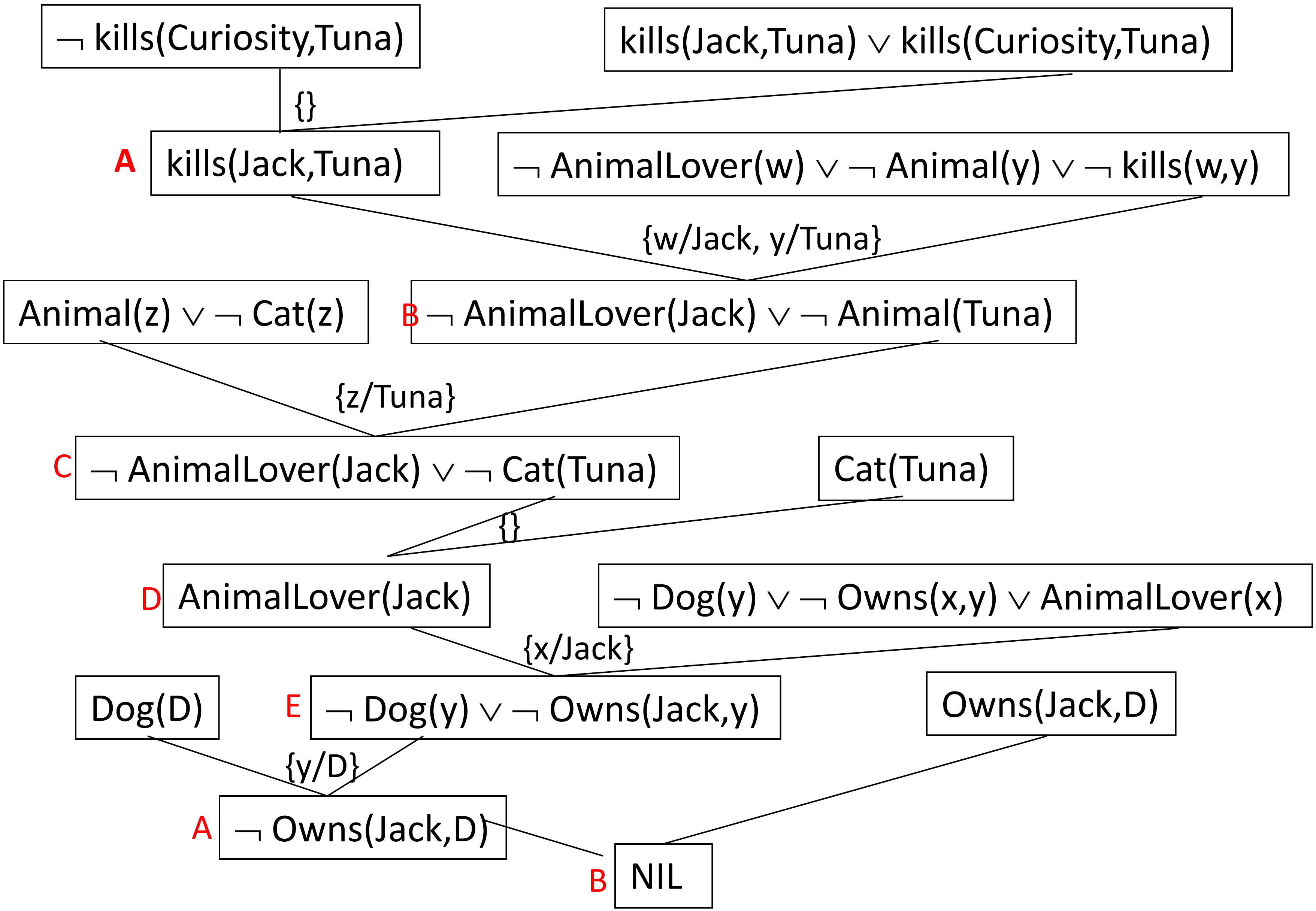
$$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$$

$$Cat(Tuna)$$

$$\neg Cat(z) \vee Animal(z)$$

$$\neg Kills(Curiosity, Tuna)$$

Find Mistake in proof by Resolution



¬ kills(Curiosity,Tuna)

kills(Jack,Tuna) ∨ kills(Curiosity,Tuna)

{}

A kills(Jack,Tuna)

¬ AnimalLover(w) ∨ ¬ Animal(y) ∨ ¬ kills(w,y)

{w/Jack, y/Tuna}

Animal(z) ∨ ¬ Cat(z)

B ¬ AnimalLover(Jack) ∨ ¬ Animal(Tuna)

{z/Tuna}

C ¬ AnimalLover(Jack) ∨ ¬ Cat(Tuna)

Cat(Tuna)

{}

D AnimalLover(Jack)

¬ Dog(y) ∨ ¬ Owns(x,y) ∨ AnimalLover(x)

{x/Jack}

Dog(D)

E ¬ Dog(y) ∨ ¬ Owns(Jack,y)

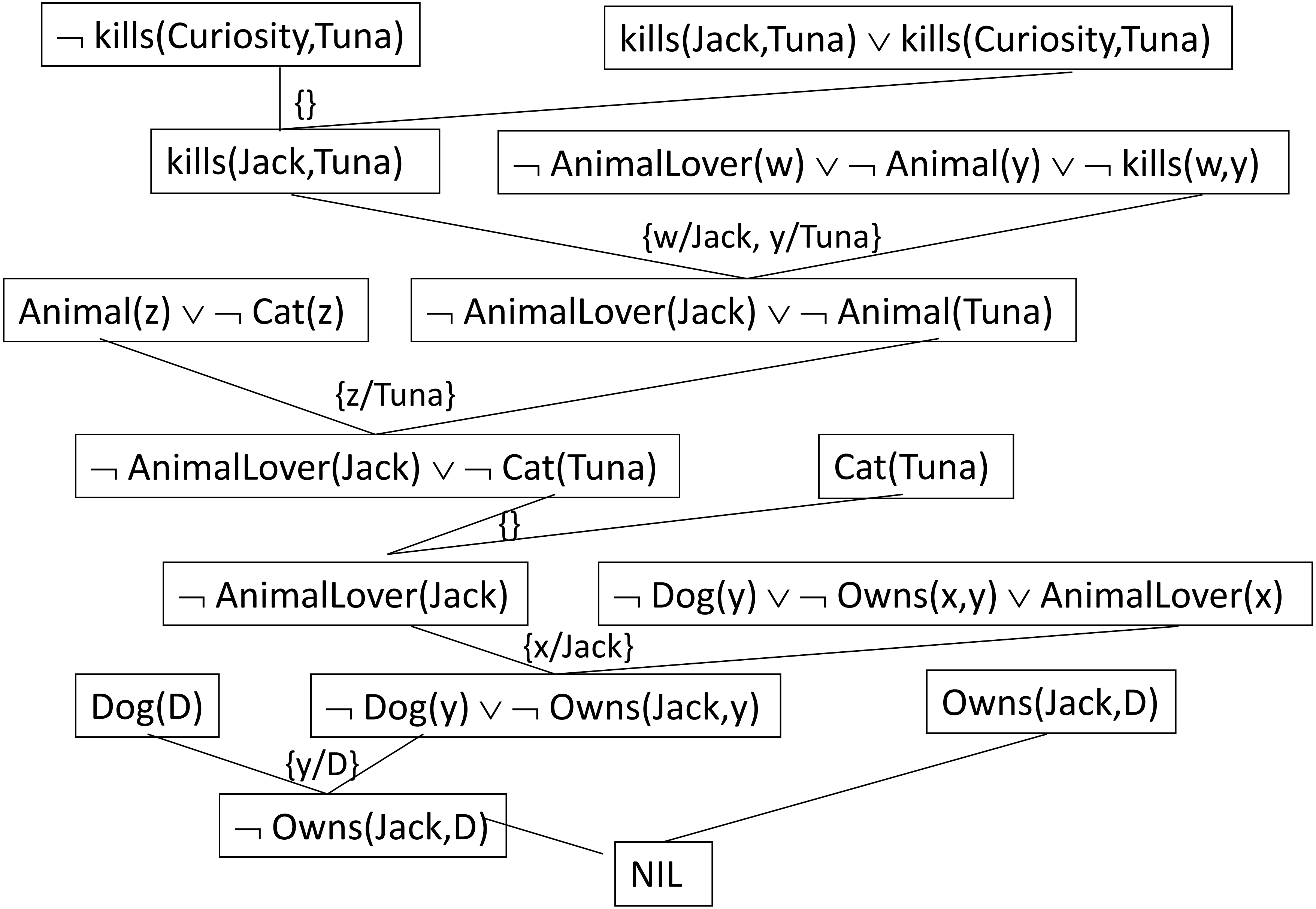
Owns(Jack,D)

{y/D}

A ¬ Owns(Jack,D)

B NIL

Proof by Resolution



¬ kills(Curiosity,Tuna)

kills(Jack,Tuna) ∨ kills(Curiosity,Tuna)

{}

kills(Jack,Tuna)

¬ AnimalLover(w) ∨ ¬ Animal(y) ∨ ¬ kills(w,y)

{w/Jack, y/Tuna}

Animal(z) ∨ ¬ Cat(z)

¬ AnimalLover(Jack) ∨ ¬ Animal(Tuna)

{z/Tuna}

¬ AnimalLover(Jack) ∨ ¬ Cat(Tuna)

Cat(Tuna)

{}

¬ AnimalLover(Jack)

¬ Dog(y) ∨ ¬ Owns(x,y) ∨ AnimalLover(x)

{x/Jack}

Dog(D)

¬ Dog(y) ∨ ¬ Owns(Jack,y)

Owns(Jack,D)

{y/D}

¬ Owns(Jack,D)

NIL

Simple problem

- Schubert Steamroller:
  - Wolves, foxes, birds, caterpillars, and snails are animals and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants.
- Prove: there is an animal that likes to eat a grain-eating animal
- Some of the necessary logical forms:
  - $\forall$ x (Wolf(x) $\rightarrow$ animal(x))
  - $\forall$ x $\forall$ y ((Caterpillar(x) $\vee$ Bird(y)) $\rightarrow$ Smaller(x,y)
  - $\exists$ x bird(x)
- Requires almost 150 resolution steps (minimal)

Proofs can be Lengthy

A relatively straightforward KB can quickly overwhelm general resolution methods.

Resolution strategies reduce the problem somewhat, but not completely.

As a consequence, many practical Knowledge Representation formalisms in AI use a restricted form and specialized inference.

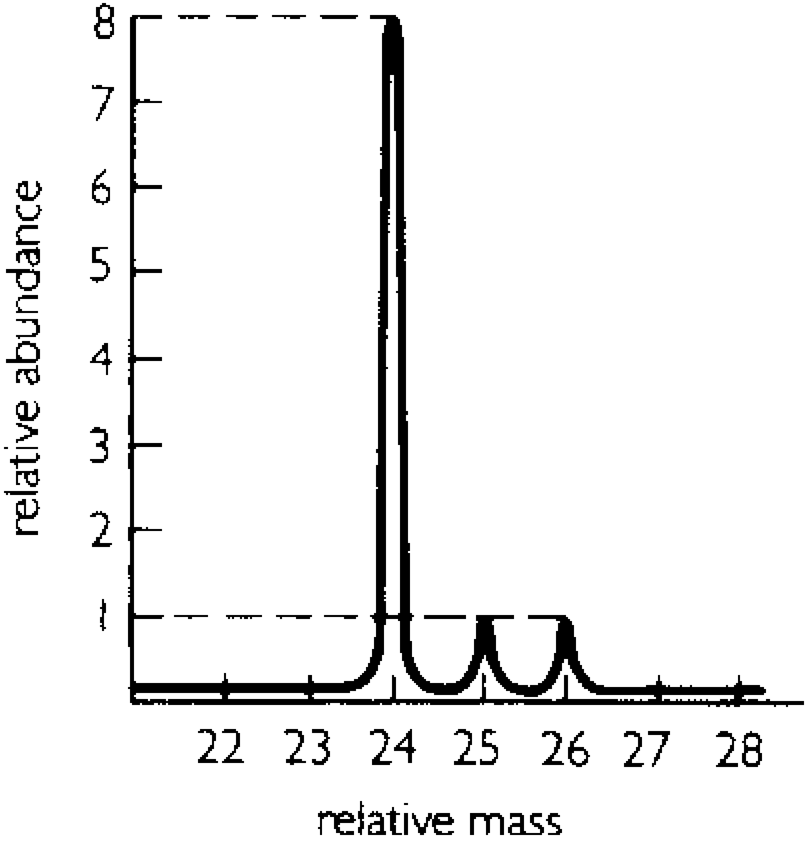
- Logic programming (Prolog)
- Production systems
- Frame systems and semantic networks
- Description logics

Successes in Rule-Based Reasoning

Expert systems

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

DENRAL: Mass Spectrometry



Successes in Rule-Based Reasoning

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

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

Successes in Rule-Based Reasoning

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

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

Successes in Rule-Based Reasoning

- PROSPECTOR (Duda *et al*., 1979)
  - Correctly recommended exploratory drilling at geological site
  - Rule-based system founded on probability theory
- R1 (McDermott, 1982)
  - Designs configurations of computer components
  - About 10,000 rules

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

Cognitive Modeling with Rule-Based Systems

SOAR is a general architecture for building intelligent systems.

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

Properties of Knowledge-Based Systems

Advantages

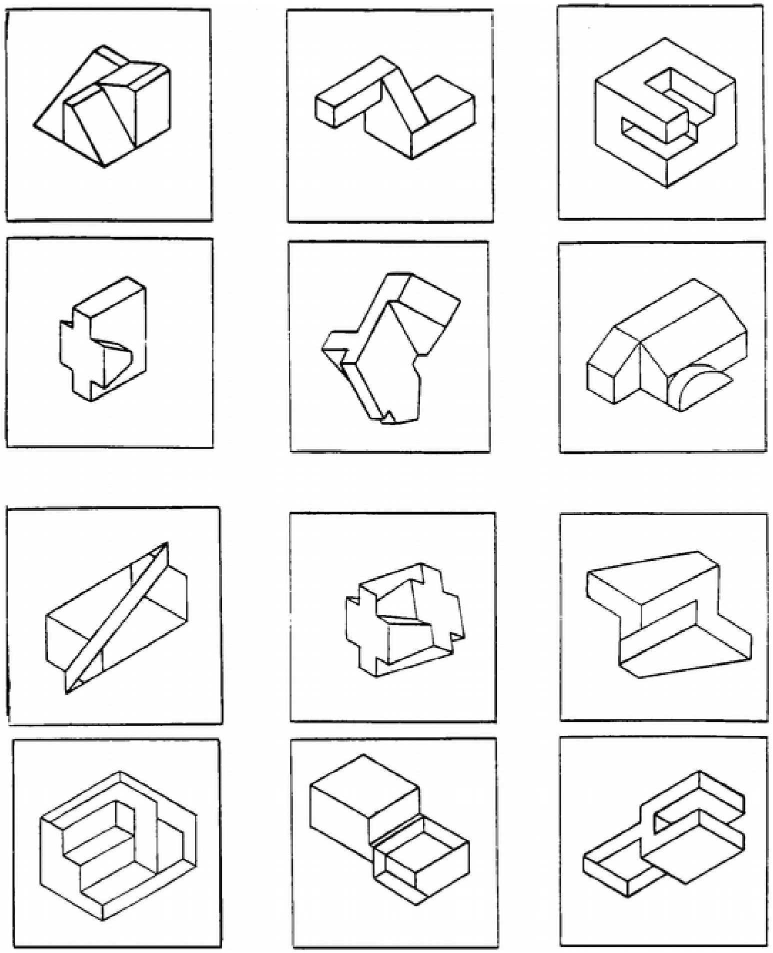
1. Expressibility*: Human readable
2. Simplicity of inference procedures*: Rules/knowledge in same form
3. Modifiability*: Easy to change knowledge
4. Explainability: Answer “how” and “why” questions.
5. Machine readability
6. Parallelism*

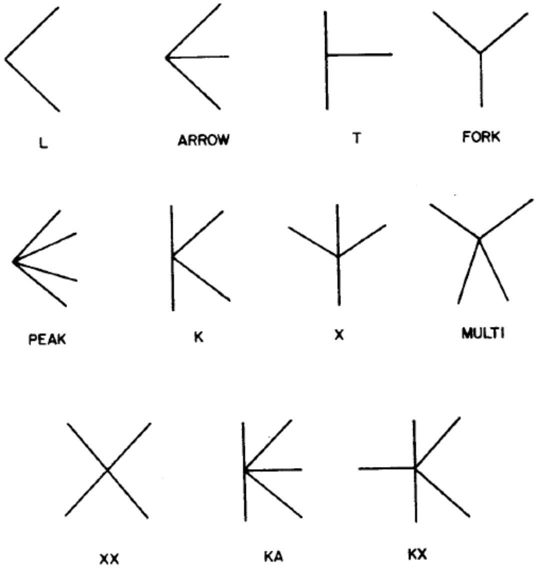
Disadvantages

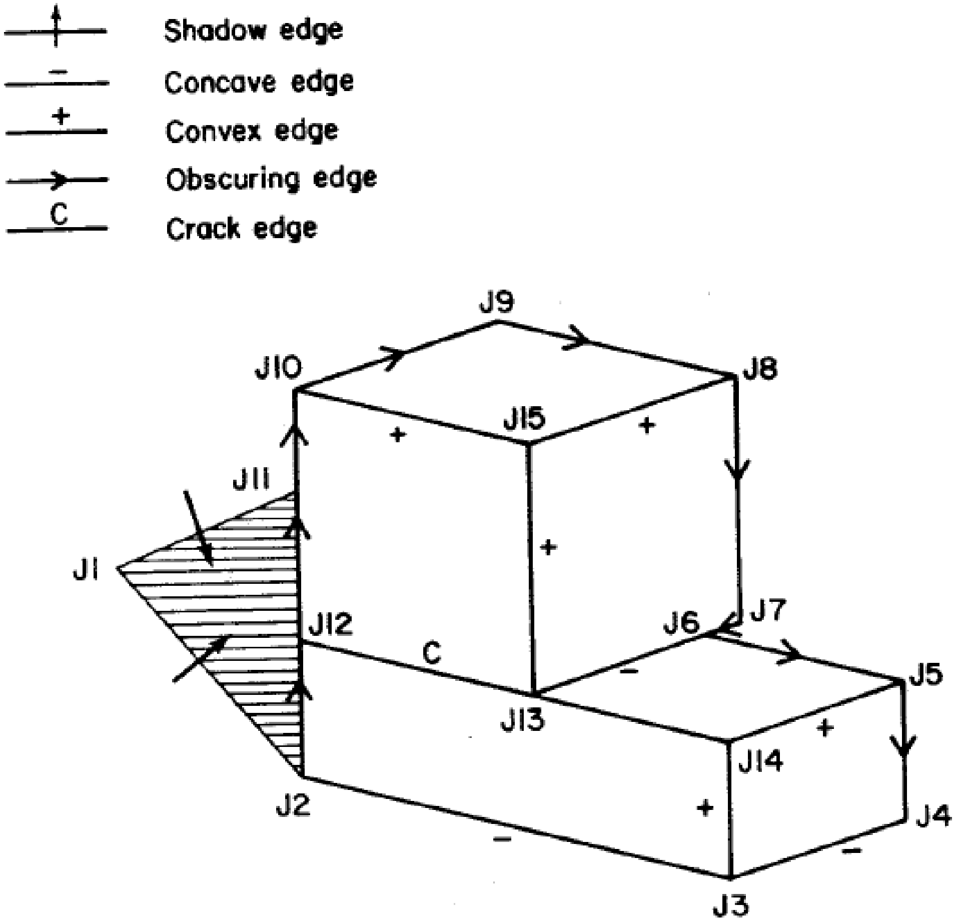
1. Difficulties in expressibility
2. Undesirable interactions among rules
3. Non-transparent behavior
4. Difficult debugging
5. Slow
6. Where does the knowledge base come from???

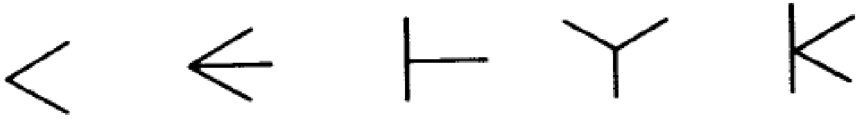
Impossible Objects as Nonsense Sentences

D. A. Huffman, *Machine Intelligence*, Vol. 6 (1971), pp. 295-323

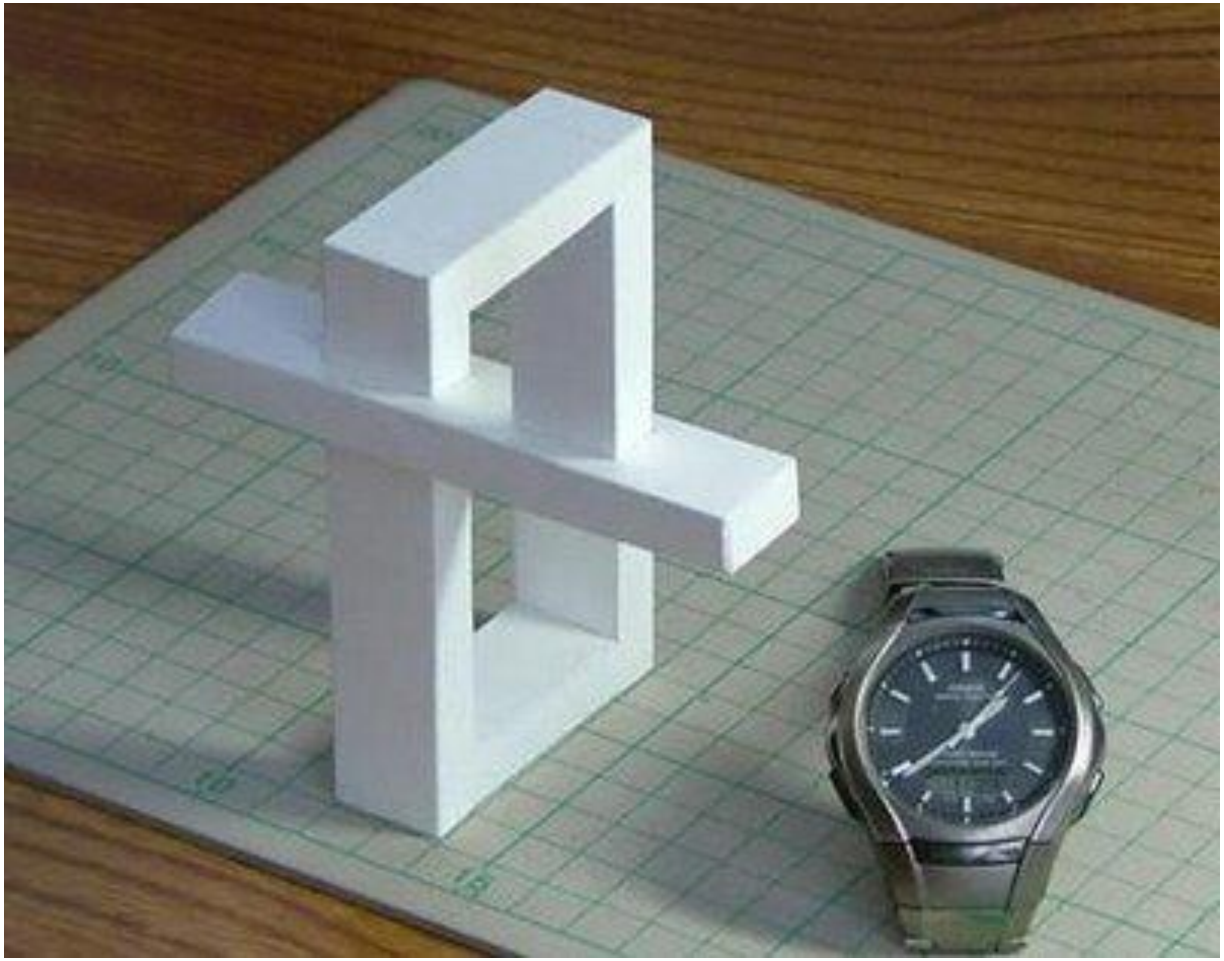








(L)	(ARROW)	(T)	(FORK)	(K)
J1	J2	J6	J14	J13
J4	J3	J11	J15	
J7	J5	J12		
J9	J8			
	J10			



Other types of “logic” KB

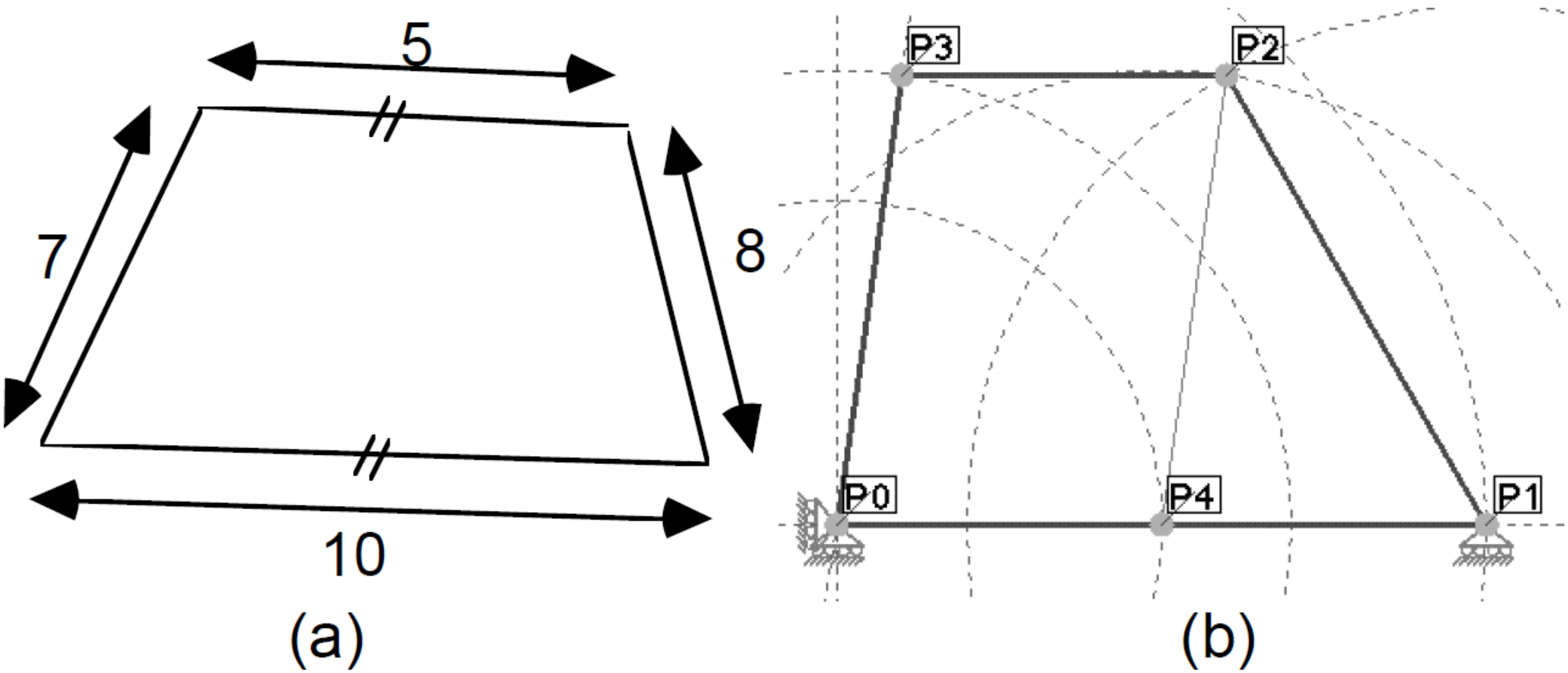
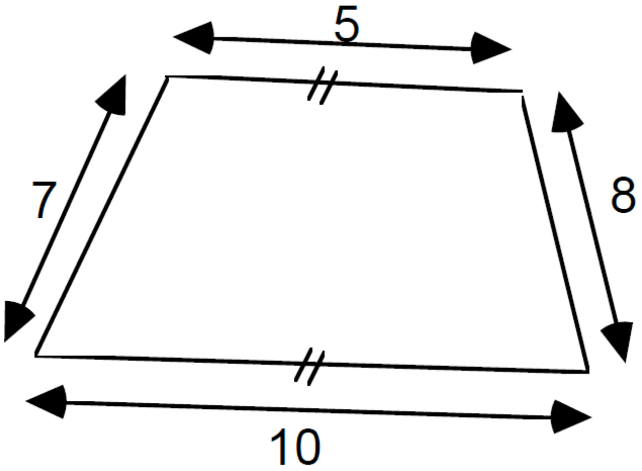
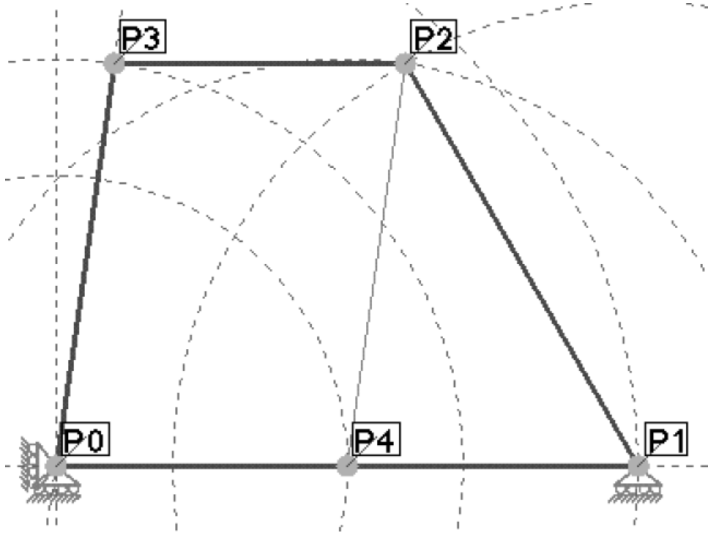


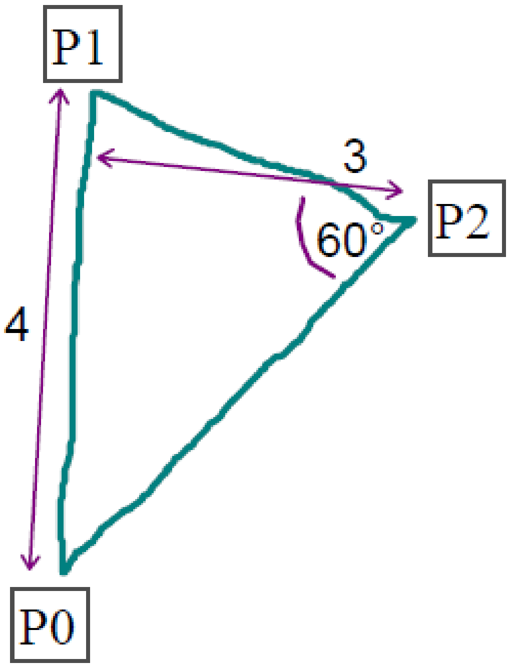
Fig 4. Constraint relocation: (a) Initial problem; (b) solution. Note the automatic generation of point 4.

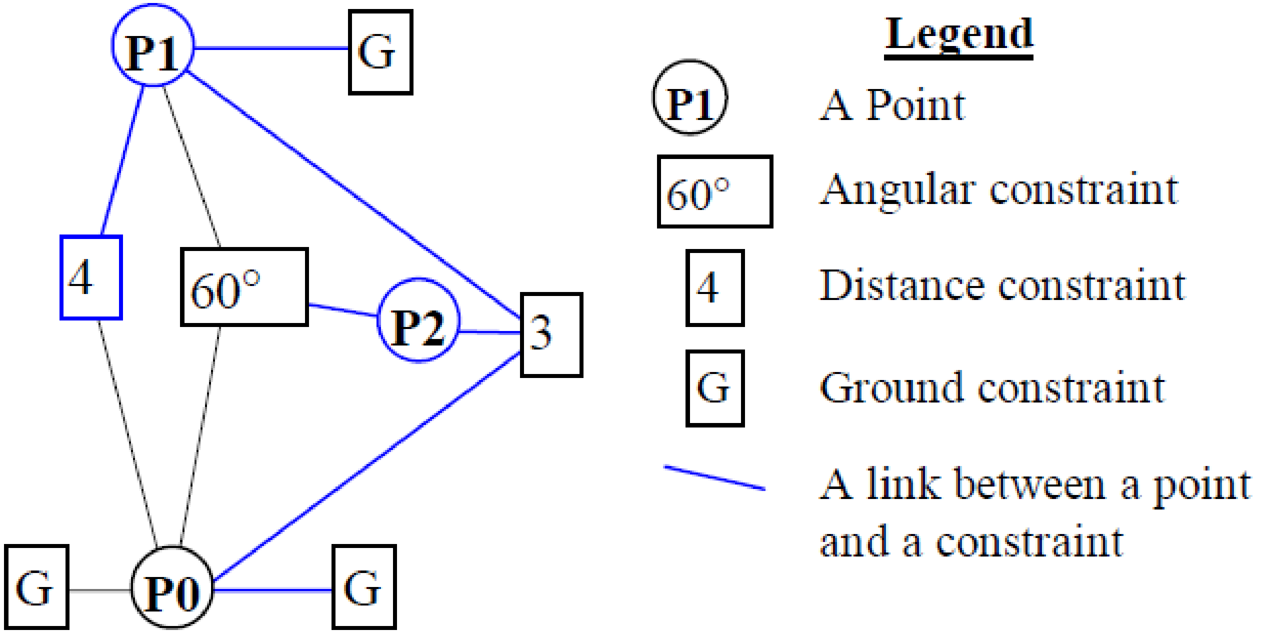
Geometric KB

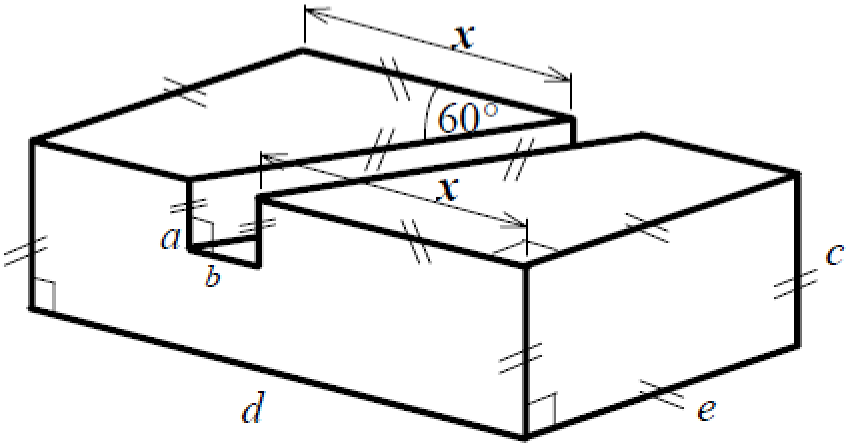


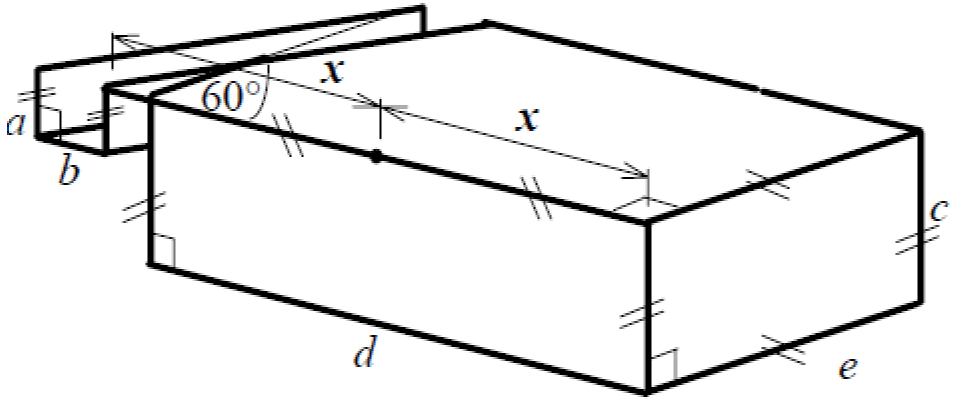


C0:P0.X=6.0 C4:D(P2,P3)=5.00
C1:P0.Y=7.0 C5:A(P2,P3,P0,P1)=0'
C2:P1.Y=7.0 C6:D(P1,P2)=8.00
C3:D(P0,P1)=10.00 C7:D(P0,P3)=7.00
Solving:
Using ground-x constraint C0: Locus P0L0: Point P0 is on line 1.00x+0.00y+-6.00=0
Using ground-y constraint C1: Locus P0L1: Point P0 is on line 0.00x+1.00y+-7.00=0
Combining locus P0L0 and P0L1, point P0 is at 6.00;7.00
Using ground-y constraint C2: Locus P1L0: Point P1 is on line 0.00x+1.00y+-7.00=0
Using point-point distance constraint C3 and point P0: Locus P1L1: Point P1 is on circle at 6.00;7.00 with R=10.00
Combining locus P1L0 and P1L1, point P1 is at 16.00;7.00 or -4.00;7.00 (Selected 1st)
Using point-point distance constraint C6 and point P1: Locus P2L0: Point P2 is on circle at 16.00;7.00 with R=8.00
Using point-point distance constraint C7 and point P0: Locus P3L0: Point P3 is on circle at 6.00;7.00 with R=7.00
***Translocating constraints using the parallelogram principle** Added point P4 at 0.90;7.30*
Translocating C7 into C8:D(P2,P4)=7.00
Translocating C4 into C9:D(P0,P4)=5.00
From parallelogram principle: C11:A(P2,P3,P0,P4)=0' C10:A(P0,P3,P2,P4)=0'
Using point-point distance constraint C9 and point P0:
Locus P4L0: Point P4 is on circle at 6.00;7.00 with R=5.00
Deriving from C5, C11: C12:A(P1,P0,P4)=0`
Using angle constraint C12 and points P0, P1, P0: Locus P4L1: Point P4 is on line 0.00x+1.00y+-7.00=0
Combining locus P4L0 and P4L1, point P4 is at 11.00;7.00 or 1.00;7.00 (Selected 1st)
Using point-point distance constraint C8 and point P4: Locus P2L1: Point P2 is on circle at 11.00;7.00 with R=7.00
Combining locus P2L0 and P2L1, point P2 is at 12.00;0.07 or 12.00;13.93 (Selected 2nd)
Using angle constraint C10 and points P3, P2, P4: Locus P3L1: Point P3 is on line 0.99x+-0.14y+ -4.94=0
Combining locus P3L0 and P3L1, point P3 is at 7.00;13.93 or 5.00;0.07 (Selected 1st)


P1
3
P2
60°
4
P0


P1
G
4
60°
P2
3
G
P0
G
Legend
P1
A Point
60°
Angular constraint
4
Distance constraint
G
Ground constraint
A link between a point and a constraint


x
60°
x
a
b
c
d
e


x
60°
x
a
b
c
d
e

Prolog (Programming in Logic)

- What is Prolog?
  - Full-featured programming language
  - Programs consist of logical formulas
  - Running a program means proving a theorem
- Syntax of Prolog
  - Predicates, objects, and functions:
    - cat(tuna), append(a,pair(b))
  - Variables: X, Y, List (capitalized)
  - Facts:
    - university(cornell).
    - prepend(a,pair(a,X)).
  - Rules:
    - animal(X) :- cat(X). **Read: “Animal is true if Cat is true”**
    - student(X) :- person(X), enrolled(X,Y), university(Y).

    → implication “:-” with single predicate on left and only non-negated predicates on the right. All variables implicitly “forall” quantified.
  - Queries:
    - student(X).

    → All variables implicitly “exists” quantified.

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

?- sibling(sally, erica).
Yes
```

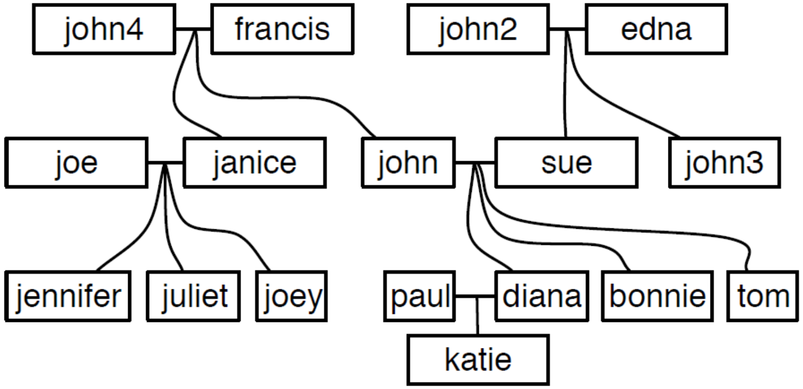
W-Prolog

```
parent(sue, diana).       parent(john, diana).
parent(sue, bonnie).      parent(john, bonnie).
parent(sue, tom).         parent(john, tom).
parent(diana, katie).     parent(paul, katie).
parent(edna, sue).        parent(john2, sue).
parent(edna, john3).      parent(john2, john3).
parent(francis, john).    parent(john4, john).
parent(francis, janice).  parent(john4, janice).
parent(janice, jennifer). parent(joe, jennifer).
parent(janice, juliet).   parent(joe, juliet).
parent(janice, joey).     parent(joe, joey).

% Each one of these statements, "parent(x,y)" indicates that "x" is a
% parent of "y".  These are called "facts" in Prolog parlance.

% Convenience rules to test (in)equality of terms.  Note the use of upper
% case letters.  The upper case letters are variables.  When used in a query,
% Prolog will attempt to match
eq(A, A).
neq(A, B) :- not(eq(A, B)).

% X is Y's ancestor if X is Y's parent.
ancestor(X, Y) :- parent(X, Y).
% X is Y's ancestor if X is parent of some Z who is Y's ancestor.  Note
% the use of the variable Z, and of (in some sense) recursion.
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```



http://waitaki.otago.ac.nz/~michael/wp/

Programming in Prolog

- Path Finding

```
path(Node1,Node2) :-
  edge(Node1,Node2).
path(Node1,Node2) :-
     edge(Node1,SomeNode),
     path(SomeNode,Node2).
edge(ith,lga).
edge(ith,phl).
edge(phl,sfo).
edge(lga,ord).
```

- Query
  - `path(ith,ord).`
  - `path(ith,X).`

Programming in Prolog

- Data structures: Lists

```
length([],0).
length([H|T],N) :- length(T,M), N is M+1.

member(X,[X|List]).
member(X,[Element|List]) :- member(X,List).

append([],List,List).
append([Element|L1],L2,[Element|L1L2]) :-
   append(L1,L2,L1L2).
```

- Query:
  - `length([a,b,c],3).`
  - `length([a,b,c],X).`
  - `member(b,[a,b,c]).`
  - `member(X,[a,b,c]).`

Programming in Prolog

Example: Symbolic derivatives (http://cs.wwc.edu/~cs_dept/KU/PR/Prolog.html)

```
% deriv(Polynomial, variable, derivative)
% dc/dx = 0
deriv(C,X,0) :- number(C).
% dx/dx} = 1
deriv(X,X,1).
% d(cv)/dx = c(dv/dx)
deriv(C*U,X,C*DU) :- number(C), deriv(U,X,DU).
% d(u v)/dx = u(dv/dx) + v(du/dx)
deriv(U*V,X,U*DV + V*DU) :- deriv(U,X,DU), deriv(V,X,DV).
% d(u ± v)/dx = du/dx ± dv/dx
deriv(U+V,X,DU+DV) :- deriv(U,X,DU), deriv(V,X,DV).
deriv(U-V,X,DU-DV) :- deriv(U,X,DU), deriv(V,X,DV).
% du^n/dx = nu^{n-1}(du/dx)
deriv(U^+N,X,N*U^+N1*DU) :- N1 is N-1, deriv(U,X,DU).
```

Programming in Prolog

- Towers of Hanoi: move N disks from pin a to pin b using pin c.

```
hanoi(N):-hanoi(N, a, b, c).

hanoi(0,A,B,C).

hanoi(N,FromPin,ToPin,UsingPin):-
      M is N-1,
      hanoi(M,FromPin,UsingPin,ToPin),
      move(FromPin,ToPin),
      hanoi(M,UsingPin,ToPin,FromPin).

move(From,To):-
      write([move, disk from, pin, From, to, pin,
      ToPin]),nl.
```

Programming in Prolog

- 8-Queens:

```
solve(P) :-
      perm([1,2,3,4,5,6,7,8],P),
      combine([1,2,3,4,5,6,7,8],P,S,D),
      all_diff(S),
      all_diff(D).
combine([X1|X],[Y1|Y],[S1|S],[D1|D]) :-
      S1 is X1 +Y1,
      D1 is X1 - Y1,
      combine(X,Y,S,D).
combine([],[],[],[]).
all_diff([X|Y]) :-  \+member(X,Y), all_diff(Y).
all_diff([X]).
```