Knowledge Representation (Pt 2)

- First-Order Predicate Calculus
  - Universal and Existential Quantification
  - Universal and Existential Instantiation
- Proving with AI
  - Unification
  - Eliza
  - FOPC Conversion to CNF and Resolution
  - Otter
  - Expert Systems
- The next step - Planning

Limitations of Propositional Logic

- Propositional logic cannot express general-purpose knowledge succinctly
  - We need 32 sentences to describe the relationship between wumpi and stenches
  - We would need another 32 sentences for pits and breezes
  - We would need at least 64 sentences to describe the effects of actions
  - How would we express the fact that there is only one Wumpus?
  - Also difficult to identify individuals (Mary, 3)
  - Generalizations, patterns, regularities difficult to represent (all triangles have 3 sides)

First-Order Predicate Calculus

- Propositional Logic uses only propositions (symbols representing facts), only possible values are True and False
- First-Order Logic includes:
  - Objects: peoples, numbers, places, ideas (atoms)
  - Predicate/Relations: relationships between objects or property of an object
    Examples: father(x, y), father(Fred, Mary)
    - Properties: properties of atoms (predicates with single variables)
      Example: red(ball)
    - Atomic Sentence/Formula: statements that can be combined (instantiated predicates)
      Examples: father(Fred, Mary), red(Ball)
  - Functions: father-of next, (any value in domain)

Example

- Express “Socrates is a man” in
  - Propositional logic: MANSOCRATES - single proposition representing entire idea
  - First-Order Predicate Calculus: Man(Socrates) - predicate representing property of constant Socrates

FOPC Definitions

- Term - Anything that identifies an object
  Function(args)
  Constant - function with 0 args
- Atomic sentence - Predicate with term arguments
  Enemies(WilyCoyote, RoadRunner)
  Married(FatherOf(Alex), MotherOf(Alex))
- Literals - atomic sentences and negated atomic sentences
- Connectives ($\Leftrightarrow, \Rightarrow, \wedge, \vee, \neg$)
- Quantifiers
  Universal Quantifier $\forall$
  Existential Quantifier $\exists$

Universal Quantifiers

- How do we express "All unicorns speak English" in Propositional Logic?
- We would need to specify a proposition for each unicorn
- $\forall$ is used to express facts and relationships that we know to be true for all members of a group (objects in the world)
- A variable is used in the place of an object
  $\forall x$, Unicorn(x) $\Rightarrow$ SpeakEnglish(x)
  The "domain" of x is the world
  The "scope" of x is the statement following $\forall$ (sometimes in [])
- Same as specifying
  Unicorn(Uni1) $\Rightarrow$ SpeakEnglish(Uni1) $\wedge$
  Unicorn(Uni2) $\Rightarrow$ SpeakEnglish(Uni2) $\wedge$
  Unicorn(Uni3) $\Rightarrow$ SpeakEnglish(Uni3) $\wedge$ …
  Unicorn(Table1) $\Rightarrow$ Table(Table1) $\wedge$ ...
  One statement for each object in the world
- We will leave variables lower case (sometimes ?x)
  Notice that x ranges over all objects, not just unicorns.

Existential Quantifiers

- This makes a statement about some object (not named)
  $\exists x$ [Bunny(x) $\wedge$ EatsCarrots(x)]
- And means there exists some object in the world (at least one) for which the statement is true. Same as disjunction over all objects in the world.
  (Bunny(Bun1) $\wedge$ EatsCarrots(Bun1)) $\vee$
  (Bunny(Bun2) $\wedge$ EatsCarrots(Bun2)) $\vee$
  (Bunny(Bun3) $\wedge$ EatsCarrots(Bun3)) $\vee$ …
  (Bunny(Table1) $\wedge$ EatsCarrots(Table1)) $\vee$ ...
- What about $\exists$ x, Unicorn(x) $\Rightarrow$ SpeakEnglish(x) ?

DeMorgan Rules

- $\forall x \neg P \Leftrightarrow \neg \exists x\ P$
- $\forall x\ P \Leftrightarrow \neg \exists\ x \neg P$
- $\neg \forall x \neg P \Leftrightarrow \exists x\ P$
- $\neg \forall x\ P \Leftrightarrow \exists x \neg P$
- Example: $\forall x$ LovesWatermelon(x) $\Leftrightarrow \neg \exists x$ $\neg$LovesWatermelon(x)

Other Properties

Predicate Calculus Equivalences		
Quantifier Negation	$\neg\forall x\ A(x) \equiv \exists x\ \neg A(x)$	$\neg\exists x\ A(x) \equiv \forall x\ \neg A(x)$
Universal Instantiation	$\forall x\ P(x) \Rightarrow S_t^x P$	$\exists y\ \forall x\ P(x) \Rightarrow \exists\ y\ S_t^x P$
Existential Instantiation	$\exists x\ P \Rightarrow S_t^x\ P$	$\forall y\ \exists x\ P \Rightarrow S_{A(y)}^x\ P$
Universal Generalization	$P \Rightarrow \forall x\ P$	
Existential Generalization	$S_t^x P \Rightarrow \exists x\ P(x)$	
Equivalences	$\forall x\ A \equiv A$	$\exists x\ A \equiv A$
Variable Renaming	$\forall x\ A \equiv \forall y\ S_y^x A$ if y free in A	$\exists x\ A \equiv \exists y\ S_y^x A$ if y free in A
Unification	$\forall x\ A \equiv S_t^x\ A \wedge \forall x$ A for any term t	$\exists x\ A \equiv S_t^x\ A \vee \exists x$ A for any term t
Quantifier Associativity	$\forall x\ (A \wedge B) \equiv \forall x\ A \wedge \forall x\ B$ $\exists x\ (A \wedge B) \Rightarrow \exists x\ A \wedge \exists x\ B$	$\exists x\ (A \vee B) \equiv \exists x\ A \vee \exists x\ B$ $\forall x\ (A \vee B) \Rightarrow \forall x\ A \vee \forall x\ B$
Quantifier Commutativity	$\forall x\ \forall y\ A \equiv \forall y\ \forall x\ A$	$\exists x\ \exists y\ A \equiv \exists y\ \exists x\ A$
Generalized Instantiation	$\forall x\ (A \vee B) \equiv A \vee \forall x\ B$ if x free in A	$\exists x\ (A \wedge B) \equiv A \wedge \exists x\ B$ if x free in A

Universal Instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\},\alpha)}$$

  for any variable v and ground term g

- $\forall x$ 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)*)

Existential Instantiation (EI)

- For any sentence α, variable v, and constant symbol k (that does not appear in the KB):

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/k\},\alpha)}$$

- $\exists x$ Crown(x) $\wedge$ OnHead(x, John)
  Crown(C_1) $\wedge$ OnHead(C_1,John)
  provided C_1 is a new constant symbol, called a *Skolem constant*

Instantiation Continued

- UI can be applied several times to add new sentences; the new KB is logically equivalent to the old
- EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable.

Examples

- All men are mortal
  $\forall x\ [Man(x) \Rightarrow Mortal(x)]$
- Socrates is a man.
  Man(Socrates)
- Socrates is mortal
  Mortal(Socrates)
- All purple mushrooms are poisonous
  $\forall x\ [(Purple(x) \wedge Mushroom(x)) \Rightarrow Poisonous(x)]$
- Every boy owns a dog
  $\forall x \exists y\ [Boy(x) \Rightarrow Owns(x, y)]$
  $\exists y \forall x\ [Boy(x) \Rightarrow Owns(x, y)]$
- Do these mean the same thing?

Examples

- A mushroom is poisonous only if it is purple
  $\forall x\ [(Mushroom(x) \wedge Poisonous(x)) \Rightarrow Purple(x)]$
- No purple mushroom is poisonous
  $\neg(\exists x\ [Purple(x) \wedge Mushroom(x) \wedge Poisonous(x)])$
- There is exactly one mushroom
  $\exists x\ [Mushroom(x) \wedge (\forall y\ (NEQ(x,y) \Rightarrow \neg Mushroom(y)))]$
- Every city has a dog catcher who has been bitten by every dog in town.
  $\forall a, b\ [City(a) \Rightarrow \exists c\ DogCatcher(c) \wedge (Dog(b) \wedge Lives(b,a) \Rightarrow Bit(b,c))]$
- No human enjoys golf
  $\forall x\ [Human(x) \Rightarrow \neg Enjoys(x, Golf)]$
- Some professor that is not a historian writes programs
  $\exists x\ [Professor(x) \wedge \neg Historian(x) \wedge Writes(x, Programs)]$

Example Proof

- Known:

1. If x is a parent of y, then x is older than y
   $\forall x,y\ [Parent(x,y) \Rightarrow Older(x,y)]$
2. If x is the mother of y, then x is a parent of y
   $\forall x, y\ [Mother(x,y) \Rightarrow Parent(x,y)]$
3. Lulu is the mother of Fifi
   Mother(Lulu, Fifi)
4. Prove: Lulu is older than Fifi
   Older(Lulu, Fifi)
   Parent(Lulu, Fifi) 2, 3, Modus Ponens
   Older(Lulu, Fifi) 1, 4, Modus Ponens

- Note that we "bind" the variable to a constant as we apply the rule. In generating a proof, we have to decide in what order to apply rules. The order shown here is called "forward chaining".

FOPC and Wumpus World

- Perception rules
  $\forall b,g,t\ Percept([Smell, b, g], t) \Rightarrow Smelled(t)$
- Here we are indicating a Percept at time t
  $\forall s,b,t\ Percept([s,b,Glitter], t) \Rightarrow AtGold(t)$
- We can use FOPC to write rules for selecting actions:
  - Reflex Agent:
    $\forall t\ AtGold(t) \Rightarrow Action(Grab, t)$
  - Reflex Agent With Internal State:
    $\forall t\ AtGold(t) \wedge \neg Holding(Gold, t) \Rightarrow Action(Grab, t)$
    - Holding(Gold, t) cannot be observed, so keeping track of change is essential

Deducing Hidden Properties

- Properties of locations:
  $\forall 1,t\ At(Agent,1,t) \wedge Smelt(t) \Rightarrow Smelly(1)$
  $\forall 1,t\ At(Agent,1,t) \wedge Breeze(t) \Rightarrow Breezy(1)$
- Squares are breezy near a pit:
  - **Diagnostic** rule: infer cause from effect
    $\forall y\ Breezy(y) \Rightarrow \exists x\ Pit(x) \wedge Adjacent(x,y)$
  - **Causal** rule: infer effect from cause
    $\forall x,y\ Pit(x) \wedge Adjacent(x,y) \Rightarrow Breezy(y)$
- Neither of these is complete For example, the causal rule doesn't say whether squares far away from pits can be breezy
- **Definition** for the *Breezy* predicate:
  $\forall y\ Breezy(y) \Leftrightarrow [\exists x\ Pit(x) \wedge Adjacent(x,y)]$

Proving with FOPC:
Reduction to Propositional Inference

- Suppose the KB contains:
  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  King(John)
  Greedy(John)
  Brother(Richard,John)
- Instantiating the universal sentence in all possible ways:
  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  King(John)
  Greedy(John)
  Brother(Richard, John)
- The new KB is propositionalized: proposition symbols are:
  - King(John), King(Richard), Evil(John), Greedy(John), etc.

Reduction Continued

- Claim: a ground sentence is entailed by new KB iff entailed by original KB
  Every FOL KB can be propositionalized so as to preserve entailment
- Idea: Propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms
  - Father(Father(Father(John)))
- Theorem: Herbrand (1930): If a sentence α is entailed by a FOL KB, it is entailed by a finite subset of the propositional KB
- Idea: For n = 0 to ∞ do
  create a propositional KB by instantiating with depth-n terms
  see if α is entailed by this KB
- Problem: works if α is entailed, loops if α is not entailed

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  $\forall y$ Greedy(y)
  Brother(Richard, John)
- It seems obvious that Evil(John) but propositionalization produces lots of facts such as Greedy(Richard) that are irrelevant
- With p k-ary predicate and n constants, there are $p \bullet n^k$ instantiations!

Proof Methods

- Proof methods divide into (roughly) two kinds:
  - Model checking
    - Truth table enumeration (sound and complete for propositional logic sentence) (exponential in n)
    - Improved backtracking
    - Heuristic search in model space
      - Show that all interpretations in which the left hand side of the rule is true, the right hand side is also true (sound but incomplete)
  - Application of inference rules
    - Sound generation of new sentences
    - Proof = a sequence of inference rule applications
      - Can use inference rules as operators in a standard search algorithm.
      - Typically requires translation of sentences into a normal form

For search to work with this, we need an additional tool.

Unification

- During proofs we sometimes want to match statements with variables or other statements.

  $\neg dog(x) \vee feathers(x)$
  feathers(Tweety)
  dog(Rufus)
  $\theta = \{x/\text{Tweety}\}$ x cannot then also match Rufus
- Given expressions p and q, a unifier of p and q is any binding list θ such that $p\theta = q\theta$ ($p\theta$ means binding list θ is applied to expression p)
- Match two expressions and build a **binding list** (θ)

  $\neg hold(P1, card) \wedge \neg hold(P2, card) \wedge \neg hold(P3, card) \Rightarrow solution(card)$
  $\neg hold(P1, Rope) \wedge \neg hold(P2, Rope) \wedge \neg hold(P3, Rope)$
- If we substitute *card* with Rope everywhere (θ = {*card*/Rope}), then the second statement is equivalent to the left-hand side of the first
- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that $\omega_1 s = \omega_2 s$

Substitution

- Three valid types of substitutions:
  1. variable $\rightarrow$ constant
  2. variable1 $\rightarrow$ variable2
  3. variable $\rightarrow$ function, if function doesn't contain variable.
- Things to look out for:
  1. What if variable is already bound?
  2. What if function contains a second variable, which is already bound to the first variable?

Substitution Example

- Statements:

  programmer(*x*) programmer(Bill)

  θ = {*x*/Bill} programmer(Bill)

- Statements:
  - father(*x*, father(John))
    father(grandfather(*y*),father(*y*))

  θ = {*x*/grandfather(*y*), *y*/John}

  father (grandfather (John), father (John))

Unifiers

- Note that there can be more than one binding list that unifies two expressions.

  f(*x*) f(*y*) θ = {*x*/*y* or *x* = foo, *y* = foo}
- In general, we prefer to not overly constrain the substitutions.
  - If kept general, result can apply to greater number of future situations.

Most General Unifier

- Unifier θ_1 is more general than unifier θ_2 if for every expression *p*, $p\theta_2$ is an instance of $p\theta_1$ (or $p\theta_1 = p\theta_2$)
- θ_1 = {*x*/*y*} is more general than θ_2 = {*x*/foo,*y*/foo}, because f(*x*) applies more often (*x*/*y*,*y*/foo) than f(foo) (*x*/foo,*y*/foo)
- If two expressions are unifiable, then there exists a mgu (most general unifier)
- The algorithm in the book returns a mgu.

  Q(F(*x*), *z*, A) and Q(*a*, *z*, *y*)
  Unifiable?
  Yes! θ = {*a*/F(*x*),*y*/A}

  P(*x*) and P(A)?
  P(F(*x*, G(A,*y*)), G(A,*y*)) and P(F(*x*,*z*), *z*)?
  Q(*x*, *y*, A) and Q(B, *y*, *z*)?
  R(*x*) and R(F(*x*))?

Example Proof with Unification

Bob is a buffalo	1.	Buffalo(Bob)
Pat is a pig	2.	Pig(Pat)
Buffalo outrun pigs	3.	$\forall xy$ Buffalo(x) $\wedge$ Pig(y) $\Rightarrow$ Faster(x,y)
Prove: Bob outruns Pat		
UE3,{x/Bob,y/Pat}		

Example Proof with Unification

Bob is a buffalo	1.	Buffalo(Bob)
Pat is a pig	2.	Pig(Pat)
Buffalo outrun pigs	3.	$\forall xy$ Buffalo(x) $\wedge$ Pig(y) $\Rightarrow$ Faster(x,y)
Prove: Bob outruns Pat		
UE3,{x/Bob,y/Pat}		
AI 1,2	4.	Buffalo(Bob) $\wedge$ Pig(Pat)

Example Proof with Unification

Bob is a buffalo	1.	Buffalo(Bob)
Pat is a pig	2.	Pig(Pat)
Buffalo outrun pigs	3.	$\forall xy$ Buffalo(x) $\wedge$ Pig(y) $\Rightarrow$ Faster(x,y)
Prove: Bob outruns Pat		
UE3,{x/Bob,y/Pat}		
AI 1,2	4.	Buffalo(Bob) $\wedge$ Pig(Pat)
UE3,{x/Bob,y/Pat}	5.	Buffalo(Bob) $\wedge$ Pig(Pat) $\Rightarrow$ Faster(Bob,Pat)

Unification and Eliza

- Very popular AI program developed in the 60s.
- Eliza simulates a Rogerian psychiatrist.
- Takes your statements, permutes them, rephrases them as a question.
- Can it pass the Turing test?
- Parry - simulates paranoiac
- Simulator of autistic child

hello
i would like to talk about artificial intelligence
i don't want to talk about you right now
what?
i want to know about ai
i like afit but i have to read this book
help
Goodbye

http://www.manifestation.com/neurotoys/eliza.php3

Eliza

- Eliza uses unification to match input sentences against a set of rules.
- Here variables can match a series of words.
- Rule 2:
  ?*(?x) “I want” ?* (?y)
  “What would it mean if you got” + ?y + “?”

- Looks for the words “I want”. Match ?x to everything before the phrase, and match ?y to everything after the phrase.
- Now respond “What would it mean if you got ?y”
  - Every now and then, I want a chocolate malt.
  - What would it mean if you got a chocolate malt?

Eliza Sample Run

Hello there eliza

HOW DO YOU DO. PLEASE STATE YOUR PROBLEM.

Computers are like students

WHAT RESEMBLANCE DO YOU SEE BETWEEN
COMPUTERS AND STUDENTS

They both run if they see programs

REALLY-- IF THEY SEE PROGRAMS

Yes that is correct

YOU ARE SURE

Yes

I UNDERSTAND

Eliza and Parry Sample Run

I don't understand your motives.
WHY DON'T YOU UNDERSTAND MY MOTIVES?

Shouldn't I be?
WHY DID YOU SAY THAT JUST NOW?

That's just the way it is.
SUPPOSE THAT'S JUST THE WAY IT WERE NOT.

So?
I'M NOT SURE I UNDERSTAND YOU FULLY.

You harp on the same questions too much.
WHAT DOES THAT SIMILARITY SUGGEST TO YOU?

I don't get you.
DO YOU WISH THAT YOU DID GET I?

Example Proof with Unification

Bob is a buffalo	1.	Buffalo(Bob)
Pat is a pig	2.	Pig(Pat)
Buffalo outrun pigs	3.	$\forall xy$ Buffalo(x) $\wedge$ Pig(y) $\Rightarrow$ Faster(x,y)
Prove: Bob outruns Pat		
UE3,{x/Bob,y/Pat}		
AI 1,2	4.	Buffalo(Bob) $\wedge$ Pig(Pat)
UE3,{x/Bob,y/Pat}	5.	Buffalo(Bob) $\wedge$ Pig(Pat) $\Rightarrow$ Faster(Bob,Pat)
M.P. 4,5	6.	Faster(Bob,Pat)

Generalized Modus Ponens (GMP)

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

p_1' is King(John) p_1 is King(x)
p_2' is Greedy(y) p_2 is Greedy(x)
θ is {x/John, y/John} q is Evil(x)

- GMP used with KB of definite clauses (exactly one positive literal) All variables assumed universally quantified

Proof Methods

- Proof methods divide into (roughly) two kinds:
  - Model checking
    - Truth table enumeration (sound and complete for propositional logic sentence) (exponential in n)
    - Improved backtracking
    - Heuristic search in model space
      - Show that all interpretations in which the left hand side of the rule is true, the right hand side is also true (sound but incomplete)
  - Application of inference rules through unification
    - Sound generation of new sentences from old
    - Proof = a sequence of inference rule applications
      - Can use inference rules as operators in a standard search algorithm.
      - Typically requires translation of sentences into a normal form

Forward Chaining

- When a new fact *p* is added to the KB
  - For each rule such that *p* "matches" (unifies with) the premise
- If the other premises are **known**
  - Then add the conclusion to the KB and continue chaining
- Forward chaining is **data-driven**
  - inferring properties and categories from percepts

Forward Chaining Example

- Add facts 1, 2, 3, 4, 5, 7 in turn.
  Number in [] = unification literal; √ indicates rule firing

1. Buffalo(x) ∧ Pig(y) ⇒ Faster(x,y)
2. Pig(y) ∧ Slug(z) ⇒ Faster(y,z)
3. Faster(x,y) ∧ Faster(y,z) ⇒ Faster(x,z)
4. Buffalo(Bob) **[1a,×]**
5. Pig(Pat) **[1b,√]**
⇒ **6.** Faster(Bob,Pat) **[3a,×]**, **[3b,×]**, **[2a,×]**
7. Slug(Steve) **[2b,√]**
⇒ **8.** Faster(Pat,Steve) **[3a,×]**, **[3b,√]**
⇒ **9.** Faster(Bob,Steve) **[3a,×]**, **[3b,×]**

Properties of Forward Chaining

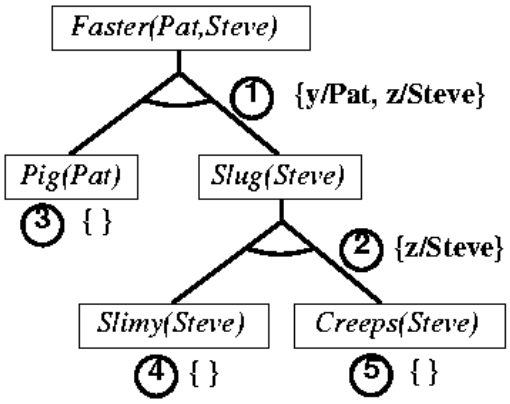
- Sound and complete for first-order definite clauses (proof similar to propositional proof)
- Datalog = first-order definite clauses + no functions, FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecideable

Backward Chaining

- When a query *q* is asked
  - If a matching fact *q'* is known, return the unifier or each rule whose consequent *q'* matches *q* attempt to prove each premise of the rule by backward chaining
    - (Some added complications in keeping track of the unifiers)
    - (More complications help to avoid infinite loops)
- Two versions: find **any** solution, find **all** solutions
- Backward chaining is the basis for **logic programming**, e.g., Prolog

Backward Chaining Example

1. Pig(y) ∧ Slug(z) ⇒ Faster(y,z)
2. Slimy(z) ∧ Creeps(z) ⇒ Slug(z)
3. Pig(Pat)
4. Slimy(Steve)
5. Creeps(Steve)



Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - Fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - Fix using caching of previous results (extra space!)

FOPC Conversion to CNF

- First, a brick is on something that is not a pyramid; second, there is nothing that a brick is on and that is on the brick as well; and third, there is nothing that is not a brick and also is the same thing as a brick.

$\forall x\ [\text{Brick}(x) \Rightarrow (\exists y\ [\text{On}(x,y) \wedge \neg\text{Pyramid}(y)] \wedge \neg\exists y\ [\text{On}(x,y) \wedge \text{On}(y,x)] \wedge \forall y\ [\neg\text{Brick}(y) \Rightarrow \neg\text{Equal}(x,y)])]$

1) Eliminate biconditionals and implications

Note: Implication $A \Rightarrow B \equiv \neg A \vee B$

$\forall x\ [\neg\text{Brick}(x) \vee (\exists y\ [\text{On}(x,y) \wedge \neg\text{Pyramid}(y)] \wedge \neg\exists y\ [\text{On}(x,y) \wedge \text{On}(y,x)] \wedge \forall y\ [\text{Brick}(y) \vee \neg\text{Equal}(x,y)])]$

Step 2

2) Move $\neg$ to literals

$\forall x\ [\neg\text{Brick}(x) \vee (\exists y\ [\text{On}(x,y) \wedge \neg\text{Pyramid}(y)] \wedge \forall y\ \neg[\text{On}(x,y) \wedge \text{On}(y,x)] \wedge \forall y\ [\text{Brick}(y) \vee \neg\text{Equal}(x,y)])]$

Note: Quantifier Negation

$\neg\forall x\ A(x) \equiv \exists x\ \neg A(x)$

$\neg\exists x\ A(x) \equiv \forall x\ \neg A(x)$

$\forall x\ [\neg\text{Brick}(x) \vee (\exists y\ [\text{On}(x,y) \wedge \neg\text{Pyramid}(y)] \wedge \forall y\ [\neg\text{On}(x,y) \vee \neg\text{On}(y,x)] \wedge \forall y\ [\text{Brick}(y) \vee \neg\text{Equal}(x,y)])]$

Note: De Morgan's laws

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

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

Step 3

3) Standardize variables (rename duplicates)

$\forall x\ [\neg \text{Brick}(x) \vee (\exists y\ [\text{On}(x,y) \wedge \neg \text{Pyramid}(y)] \wedge \forall y\ [\neg \text{On}(x,y) \vee \neg \text{On}(y,x)] \wedge \forall y\ [\text{Brick}(y) \vee \neg \text{Equal}(x,y)])]$

$\forall x\ [\neg \text{Brick}(x) \vee (\exists y\ [\text{On}(x,y) \wedge \neg \text{Pyramid}(y)] \wedge \forall z\ [\neg \text{On}(x,z) \vee \neg \text{On}(y,z)] \wedge \forall s\ [\text{Brick}(s) \vee \neg \text{Equal}(x,s)])]$

Step 4:Skolemization

4) Skolemize - Remove existential quantifiers using Existential Elimination.

- Make sure new constant is not used anywhere else.
- Danger because could be inside universal quantifiers.

  $\forall x\ \text{Person}(x) \Rightarrow \exists y\ \text{Heart}(y) \wedge \text{Has}(y,x)$

  If we just replace y with H, then

  $\forall x\ \text{Person}(x) \Rightarrow \exists y\ \text{Heart}(\text{H}) \wedge \text{Has}(\text{H},x)$

  which says that everyone has the same heart.

  Remember that because *y* is in the scope of *x*, *y* is dependent on the choice of *x*. In fact, we can represent *y* as a Skolem function of *x*.

  $\forall x\ \text{Person}(x) \Rightarrow \text{Heart}(\text{F}(x)) \wedge \text{Has}(\text{F}(x),x)$

Step 5 and 6

$\forall x\ [\neg \text{Brick}(x) \vee (\ [\text{On}(x,\text{F}(x)) \wedge \neg \text{Pyramid}(\text{F}(x))] \wedge \forall z\ [\neg \text{On}(x,z) \vee \neg \text{On}(y,z)] \wedge \forall s\ [\text{Brick}(s) \vee \neg \text{Equal}(x,s)])]$

5) Remove universal quantifiers

$\neg \text{Brick}(x) \vee (\ [\text{On}(x,\text{F}(x)) \wedge \neg \text{Pyramid}(\text{F}(x))] \wedge [\neg \text{On}(x,z) \vee \neg \text{On}(y,z)] \wedge [\text{Brick}(s) \vee \neg \text{Equal}(x,s)])$

6) Distribute $\wedge$ over $\vee$ (used distribution)

$(\neg \text{Brick}(x) \vee \text{On}(x,\text{F}(x))) \wedge$

$(\neg \text{Brick}(x) \vee \neg \text{Pyramid}(\text{F}(x))) \wedge$

$(\neg \text{Brick}(x) \vee \neg \text{On}(x,z) \vee \neg \text{On}(y,z)) \wedge$

$(\neg \text{Brick}(x) \vee \text{Brick}(s) \vee \neg \text{Equal}(x,s))$

Steps 7 and 8

7) Separate conjunctions into unique sentences

1. $\neg \text{Brick}(x) \vee \text{On}(x,\text{F}(x))$
2. $\neg \text{Brick}(x) \vee \neg \text{Pyramid}(\text{F}(x))$
3. $\neg \text{Brick}(x) \vee \neg \text{On}(x,z) \vee \neg \text{On}(y,z)$
4. $\neg \text{Brick}(x) \vee \text{Brick}(s) \vee \neg \text{Equal}(x,s)$

8) Give each sentence unique variables

1. $\neg \text{Brick}(x_1) \vee \text{On}(x_1,\text{F}(x_1))$
2. $\neg \text{Brick}(x_2) \vee \neg \text{Pyramid}(\text{F}(x_2))$
3. $\neg \text{Brick}(x_3) \vee \neg \text{On}(x_3,z_1) \vee \neg \text{On}(y_1,z_1)$
4. $\neg \text{Brick}(x_4) \vee \text{Brick}(s_1) \vee \neg \text{Equal}(x_4,s_1)$

Resolution

- A complete inference rule
- $(A \vee B, \neg B) \Rightarrow A$

$$\frac{(p_1 \vee p_2 \vee \ldots \vee p_m), (\neg p_1 \vee p_n \vee \ldots \vee p_q)}{p_2 \vee \ldots \vee p_m \vee p_n \vee \ldots \vee p_q}$$

- Unification with resolution
  $P(x) \vee Q(x), \neg Q(\text{Mary})$
  UNIFY the two clauses
  $P(\text{Mary}) \vee Q(\text{Mary})$
  $\neg Q(\text{Mary})$
  $\theta = \{x/\text{Mary}\}$

Proof by Refutation Using Resolution

- To prove a statement using resolution:
  - Convert each statement in database to CNF
  - Negate the goal statement and convert to CNF
  - Repeatedly apply resolution to generate new statements
  - If generate empty statement [], proof is complete
- This is Proof by refutation
  - We know database statements are all true
  - Resolution is sound
    All statements generated are true if database is true
  - How can we come up with empty (false) clause?
    - Only possible false statement is the negated goal
    - If negated goal is false, goal must be true ($A \vee \neg A$ is a tautology)

Resolution Properties

- Resolution is complete in the following sense: if a formula *g* follows from a set of formulas ***S***, then there is a sequence of resolutions of clauses in $\boldsymbol{S} \cup \neg g$ that terminates in the empty clause.

Example

- If Linus is sitting in the pumpkin patch, it must be Halloween. The Great Pumpkin appears on Halloween. Linus is sitting in the pumpkin patch. Prove the Great Pumpkin will appear today.

1. SitPatch(Linus) $\Rightarrow$ Halloween(Today)
2. Halloween(Today) $\Rightarrow$ GrPumpkin(Today)
3. SitPatch(Linus)

Rewrite:

1. $\neg$SitPatch(Linus) $\vee$ Halloween(Today)
2. $\neg$Halloween(Today) $\vee$ GrPumpkin(Today)
3. SitPatch(Linus)
4. $\neg$GrPumpkin(Today)

Negated goal Proof:

Example

- If Linus is sitting in the pumpkin patch, it must be Halloween. The Great Pumpkin appears on Halloween. Linus is sitting in the pumpkin patch. Prove the Great Pumpkin will appear today.

1. SitPatch(Linus) $\Rightarrow$ Halloween(Today)
2. Halloween(Today) $\Rightarrow$ GrPumpkin(Today)
3. SitPatch(Linus)

Rewrite:

1. $\neg$SitPatch(Linus) $\vee$ Halloween(Today)
2. $\neg$Halloween(Today) $\vee$ GrPumpkin(Today)
3. SitPatch(Linus)
4. $\neg$GrPumpkin(Today)

Negated goal Proof:

5. [2, 4] $\neg$Halloween(Today)
6. [1, 5] $\neg$SitPatch(Linus)
7. [3, 6] NULL

Example

- If the maid stole the jewelry, then the butler wasn't guilty. Either the maid stole the jewelry or she milked the cows. If the maid milked the cows, then the butler got the cream. Goal: Therefore, if the butler was guilty, then he got the cream.

G(M) $\Rightarrow$ $\neg$G(B)
G(M) $\vee$ Cows(M)
$\neg$Cows(M) $\vee$ Cream(B)
$\neg$(G(B) $\Rightarrow$ Cream(B))

Rewrite:

1. $\neg$G(M) $\vee$ $\neg$G(B)
2. G(M) $\vee$ Cows(M)
3. $\neg$Cows(M) $\vee$ Cream(B)
4. G(B)
5. $\neg$Cream(B)

Proof:

Example

- If the maid stole the jewelry, then the butler wasn't guilty. Either the maid stole the jewelry or she milked the cows. If the maid milked the cows, then the butler got the cream. Goal: Therefore, if the butler was guilty, then he got the cream.

  $G(M) \Rightarrow \neg G(B)$
  $G(M) \vee Cows(M)$
  $\neg Cows(M) \vee Cream(B)$
  $\neg(G(B) \Rightarrow Cream(B))$

  Rewrite:
  1. $\neg G(M) \vee \neg G(B)$
  2. $G(M) \vee Cows(M)$
  3. $\neg Cows(M) \vee Cream(B)$
  4. $G(B)$
  5. $\neg Cream(B)$

  Proof:
  6. [3, 5] $\neg Cows(M)$
  7. [2, 6] $G(M)$
  8. [1, 7] $\neg G(B)$
  9. [4, 8] NULL

Example

- Given:

  Mother(Lulu, Fifi)

  Alive(Lulu)

  $\forall\, x\, y\, [Mother(x,y) \Rightarrow Parent(x,y)$
  $\neg Mother(x,y) \vee Parent(x,y)$

  $\forall\, x\, y\, [Parent(x,y) \wedge Alive(x)) \Rightarrow Older(x,y)$
  $\neg Parent(x,y) \vee \neg Alive(x) \vee Older(x,y)$

  (Negated Goal) $\neg Older(Lulu, Fifi)$

  [1,3] Parent(Lulu, Fifi)

  [4,6] $\neg Alive(Lulu) \vee Older(Lulu, Fifi)$

  [2,7] Older(Lulu, Fifi)

  [5,8] NULL

Example

- What if the desired conclusion was “Something is older than Fifi”?
  - $\exists x$ Older(x, Fifi)
  - (Negated Goal) $\neg\exists x$ Older(x, Fifi)
  - $\neg$Older(x, Fifi)
- The last step of the proof would be:
  - [5,8] Older(Lulu, Fifi) resolved with $\neg$Older(x, Fifi) NULL with θ = {x/Lulu}
- Do not make the mistake of first forming clause from conclusion and then denying it
  - Goal: $\exists x$ Older(x, Fifi)
    Clausal Form: Older(C, Fifi)
    (Negated Goal): $\neg$Older(C, Fifi)
  - Cannot unify this statement with Older(Lulu, Fifi)

Example

1. $\neg$Read(x) $\vee$ Literate(x)
2. $\neg$Dolphin(y) $\vee$ $\neg$Literate(y)
3. Dolphin(A)
4. Intelligent(A)
5. (Negated Goal) $\neg$Intelligent(z) $\vee$ Read(z)

   Proof:
6. [4, 5] Read(A)
7. [1, 6] Literate(A)
8. [2, 7] $\neg$Dolphin(A)
9. [3, 8] NULL

Example

- Jack owns a dog
  Dog(D)
  Owns(J, D)
- Tuna is a cat
  Cat(T)
- Every dog owner is an animal lover
  $\forall x\, y\, [\text{Dog}(x) \wedge \text{Owns}(y,x) \Rightarrow \text{AnimalLover}(y)]$
- No animal lover kills an animal
  $\forall x\, z\, [\neg[\text{AnimalLover}(z) \wedge \text{Animal}(w) \wedge \text{Kill}(z,w)]]$
- Either Jack or Curiosity killed the cat who is called Tuna
  $\text{Kill}(J, T) \vee \text{Kill}(C, T)$
- Cats are animals
  $\forall u\, [\text{Cat}(u) \Rightarrow \text{Animal}(u)]$

- Prove: Curiosity killed the cat
  Kill(C, T)

Example

1. Dog(D)
2. Owns(J, D)
3. Cat(T)
4. $\neg\text{Dog}(x) \vee \neg\text{Owns}(y,x) \vee \text{AnimalLover}(y)$
5. $\neg\text{AnimalLover}(z) \vee \neg\text{Animal}(w) \vee \neg\text{Kill}(z,w)$
6. $\text{Kill}(J, T) \vee \text{Kill}(C, T)$
7. $\neg\text{Cat}(u) \vee \text{Animal}(u)$
8. (Negated Goal) $\neg\text{Kill}(C, T)$
9. [6, 8] Kill(J, T)
10. [5, 9] $\neg\text{AnimalLover}(J) \vee \neg\text{Animal}(T)$
11. [3, 7] Animal(T)
12. [10, 11] $\neg\text{AnimalLover}(J)$
13. [1, 2, 4] AnimalLover(J)
14. [12, 13] NULL

Resolution Strategies

- Marcus was a man.
  Man(Marcus)
- Marcus was a Pompeian.
  Pompeian(Marcus)
- All Pompeians were Romans.
  $\forall x$ [Pompeian(x) $\Rightarrow$ Roman(x)]
- Caesar was a ruler.
  Ruler(Caesar)
- All Romans were either loyal to Caesar or hated him.
  $\forall x$ [Roman(x) $\Rightarrow$ Loyalto(x, Caesar) $\vee$ Hated(x, Caesar)
- Everyone is loyal to someone.
  $\forall x \exists y$ [Loyalto(x, y)]
- Men only try to assassinate rulers they are not loyal to.
  $\forall x\, y$ [Man(x) $\wedge$ Ruler(y) $\wedge$ Tryassassinate(x,y) $\Rightarrow$ $\neg$Loyalto(x,y)]
- Marcus tried to assassinate Caesar.
  Tryassassinate(Marcus, Caesar)
- Goal: Marcus hated Caesar.

Resolution Strategies

1. Man(Marcus)
2. Pompeian(Marcus)
3. $\neg$Pompeian(x_1) $\vee$ Roman(x_1)
4. Ruler(Caesar)
5. $\neg$Roman(x_2) $\vee$ Loyalto(x_2, Caesar) $\vee$ Hate(x_2, Caesar)
6. Loyalto(x_3, $F_1(x_3)$)
7. $\neg$Man(x_4) $\vee$ $\neg$Ruler(y_1) $\vee$ $\neg$Tryassassinate(x_4, y_1) $\vee$ $\neg$Loyalto(x_4, y_1)
8. Tryassassinate(Marcus, Caesar) Negated Goal:
9. $\neg$Hate(Marcus, Caesar)
   Proof:
10. [5, 9] $\neg$Roman(Marcus) $\vee$ Loyalto(Marcus, Caesar) (θ = {x_2/Marcus})
11. [3, 10] $\neg$Pompeian(Marcus) $\vee$ Loyalto(Marcus, Caesar) (θ = {x_1/Marcus})
12. [2, 11] Loyalto(Marcus, Caesar)
13. [7, 12] $\neg$Man(Marcus) $\vee$ $\neg$Ruler(Caesar) $\vee$ $\neg$Tryassassinate(Marcus, Caesar)
    (θ = {x_4/Marcus, y_1/Caesar})
14. [8, 13] $\neg$Man(Marcus) $\vee$ $\neg$Ruler(Caesar)
15. [1, 14] $\neg$Ruler(Caesar)
16. [4, 15] [] Q.E.D.

Set of Support

- **Rationale:**
  - Conclusion should always play major role in proof
- **Method:**
  - Give priority to resolvents derived from set of support (clauses which are part of -G or are resolvents with a parent in the set of support)
- **Example:**
  - Resolve clauses [5, 9], then [3, 10], etc., the same way we did last time

Linear Format

- **Rationale:**
  - Gives some direction to the search [Anderson and Bledsoe] prove that any provable theorem in predicate calculus can be proven with this strategy
- **Method:**
  - Use most recent resolvent as a parent (the question of which two are resolved first is open)
- **Example:**
  - Resolve clauses [1, 7], then [4, NEW], then [8, NEW], then [5, NEW], etc.

Unit Resolution

- **Rationale:**
  - Want to derive [] (0 literals), therefore, we want to make smaller and smaller resolvents. Suppose c_1 has 4 literals, c_2 has 7 literals, R will have 9 literals!
- **Method:**
  - Use unit clause as a parent R = #literals(c_1) + #literals(c_2) - 2
  - If c_1 is a unit, R = #literals(c_2) - 1 (getting smaller)
- **Variation:**
  - Unit Preference Use unit if available, otherwise look for next smaller clause size
- **Example:**
  - Resolve clauses [1, 7], then [4,10], then [8,11], then [5,9], then [2,3], etc.

Unit-Resulting Resolution

- **Rationale:**
  - SUBSTANTIALLY decrease size of clauses
- **Method:**
  - Resolve SET of clauses One is a nonunit clause
  - Rest are unit clauses
  - Generates a new unit clause
- **Example:**
  - Resolve [2,3] yielding 10. Roman(Marcus), Resolve [9,10,5] yielding 11. Loyalto(Marcus, Caesar)
  - Resolve [1,4,8,7] yielding 12. ¬Loyalto(Marcus, Caesar)
  - Resolve [11,12] yielding []

Hyper-Resolution

- **Rationale:**
  - Not restricted to only unit clauses or unit results, but has the restriction of producing only positive results
- **Method:**
  - Resolve a clause N that contains at least one negative literal with a SET of clauses Ai which contain only positive literals Generates a new clause which contains only positive literals
- **Example:**
  - Resolve [2,3] yielding 10. Roman(Marcus)
  - Resolve [4,5] yielding 11. Loyalto(Marcus, Caesar) $\vee$ hate(Marcus, Caesar)
  - Resolve [7,1,4,8,11] yielding 12. Hate(Marcus, Caesar)
  - Resolve [9,12] yielding []

Binary Resolution

- **Rationale:**
  - Easy
- **Method:**
  - Select any two clauses which can be resolved

Breadth-First Resolution

- **Rationale:**
  - Complete
- **Method:**
  - Resolve all possible pairs of initial clauses, then all new clauses with initial set, continuing level by level

Comparison of Resolution Strategies

- Sound?
  - All of these strategies are sound
- Refutation complete?
  - Given an unsatisfiable set of clauses (KB + negated goal), the unsatisfiability can be derived using just the inference rule (resolution).
- Complete?
  - Set of support is complete
  - Hyper-resolution is complete unless constrained by set of support
  - Unit resolution is complete when each clause contains at most one positive literal (Horn clauses)
  - Breadth-first resolution is complete
  - Binary resolution is not refutation complete

Unification and Resolution

- Using the database below, prove that Colonel Mustard is a suspect.
  1. crime(Kitchen)
  2. in(ProfPlum, Library)
  3. talking(MissScarlet, ProfPlum)
  4. $\neg$talking(x_2, y_2) $\vee$ with(x_2, y_2)
  5. $\neg$with(x_4, y_4) $\vee$ $\neg$in(y_4, z_4) $\vee$ in(x_4, z_4)
  6. $\neg$in(x_5, y_5) $\vee$ $\neg$crime(y_5) $\vee$ suspect(x_5)
  7. talking(ColonelMustard, MisterGreen)
  8. $\neg$hungry(x_6) $\vee$ in(x_6, Kitchen)
  9. hungry(MisterGreen)]

Example

10. $\neg$suspect(CM)
11. [6, 10] $\neg$in(CM, y_6) $\vee$ $\neg$crime(y_6)
12. [1, 11] $\neg$in(CM, Kitchen)
13. [8, 9] in(MG, Kitchen)
14. [4, 7] with(CM, MG)
15. [5, 14] $\neg$in(MG, z_7) $\vee$ in(CM, z_7)
16. [12, 15] in(CM, Kitchen)
17. [12, 16] [] Q.E.D.

- What clause would you add to the database instead of the negated goal if you were trying to find out who is a suspect?

Green's Trick

- We have shown how resolution can prove a specific theorem.
- We can also use resolution to extract answers (perform problem solving).

- Procedure:
  - The question to be answered, or the goal, must be an existentially quantified statement. It is this variable that we will derive.
  - Form a disjunction between the positive goal clause and the negated goal clause, and add this disjunction to the database. Do not eliminate the existentially quantified variable.
  - Apply resolution as usual. When the positive goal clause is resolved, the variable will be instantiated with the answer.

Example

- Sally is studying with Morton. Morton is in the student union information office. If any person is studying with another person who is at a particular place, the first person is also at that place. If someone is at a particular place, then he or she can be reached on the telephone at the number for that place.

- What is the number where Sally can be reached?
  sw(x, y): x is studying with y
  a(x, y): x is at place y
  r(x, y): x can be reached (by telephone) at number y
  ph(x): the telephone number for place x

- We want to find a sequence of substitutions that will provide an answer to the problem. We therefore represent the question as a statement that the solution EXISTS.

- The question to be answered, or the goal, must be an existentially quantified statement. It is this variable that we will derive.

- Form a disjunction between the positive goal clause and the negated goal clause, and add this disjunction to the database. Do not eliminate the existentially quantified variable. The negated form will be eliminated (theorem proving), and the positive form will contain the answer.

Example

- Apply resolution as usual. When the positive goal clause is resolved, the variable will be instantiated with the answer.

FOPC statements

* SW(Sally, Morton)
* A(Morton, Union)
* $\forall x,y,z\ [sw(x, y) \wedge a(y, z) \Rightarrow a(x, z)]$
* $\forall x,y\ [a(x, y) \Rightarrow r(x, ph(y))]$

* GOAL: $\exists x\ r(Sally, x) \vee \neg r(Sally,x)$

Example

- Who is Lulu older than?
- Prove that there is an x such that Lulu is older than x
- FOPC: $\exists x$ Older(Lulu, x)
  (Negated Goal): $\neg$Older(Lulu, x)
- After disjoining the two together, a successful proof will yield x/Fifi

Example

- What is older than what?
- FOPC: $\exists x,y$ Older(x, y)
  (Negated Goal) $\neg$Older(x,y)
- A successful proof will yield x/Lulu, y/Fifi

OTTER

- "Organized Techniques for Theorem proving and Effective Research"
- OTTER was developed at Argonne, written in C, described in book "Automated Reasoning"
- Otter Home Page
  - http://www-unix.mcs.anl.gov/AR/otter/
- Employs binary resolution, hyper-resolution, binary paramodulation (use equality substitutions), set of support resolution, and unit-resulting resolution
- Takes FOPC statements, converts to clausal form, applies resolution

Otter Features

- Can use autonomous mode, where OTTER selects resolution strategies
- Answer literals
  - Can extract answer to query like we will show
  - Our example has an answer literal.
  - If no answer literal, end of proof yields “.” clause.
- Forward and backward subsumption
  - If clause P(x) exists in database and clause P(x) v Q(x) exists in database, we know P(x) SUBSUMES P(x) v Q(x), so delete P(x) v Q(x) from database.
- Factoring
  - Unifying 2 literals of same sign within a clause
- Weighting
  - Weight terms to affect ordering in the proof

Otter Format

- Set parameters of the system
- Give general formulas
- Give set of support (facts and formulas used in every step of proof)
- “%” is used at the beginning of a comment line.
- OPTIONS

```
set(ur_res)               % or other inference mechanisms, if not auto
assign(max_mem, 1500)     % use at most 1.5 megabytes
assign(max_seconds, 1800) % time limit of 30 CPU minutes
```

- FORMULA LIST

```
formula_list(?).          % "?" is either "sos" or "usable".
...  - facts and rules \\
end_of_list.
```

- SYNTAX
  - All clauses and other statements end with"."
  - & = and
  - | = or
  - (all x all y ())
  - (exists x exists y ())
  - - = not

Otter Format

- OTTER is not case sensitive. If a symbol is quantified, it is a variable. Otherwise, it is a predicate, function or constant.
- Put the negated goal in the sos list.
  - OTTER will keep going until sos is empty
  - Use clause in sos to make inferences (with clauses in usable list)
  - Move used clause to usable
- For answer extraction use $answer(literal).
- OTTER output contains
  - List of clauses from original database
  - List of new clauses with parent and resolution strategy
  - [ur, 4, 1] above(B, A).
  - If contradiction found
    - ---------- PROOF ----------
    - list of options used
    - stats
  - If no contradiction found and no clauses left to resolve “sos empty”

Otter Example

- Input:

```
set(ur_res).
set(hyper_res).

formula_list(usable).

all x all y all z (sw(x,y) & a(y,z) -> a(x,z)).

end_of_list.

formula_list(sos).

all x all y (a(x,y) -> r(x, ph(y))).
sw(Sally, Morton).
a(Morton, StudentUnion).
-(exists x (r(Sally, x))).

end_of_list.
```

Otter Example

- Show run on sallymorton1.txt

Otter Input Using $Answer

- Similar to using Green's Trick
- Input:

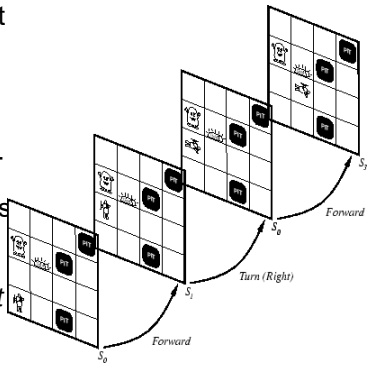
```
set(ur_res).
set(hyper_res).

formula_list(usable).

all x all y all z (sw(x,y) & a(y,z) -> a(x,z)).

end_of_list.

formula_list(sos).

all x all y (a(x,y) -> r(x, ph(y))).
sw(Sally, Morton).
a(Morton, StudentUnion).
-(exists x (-$answer(r(Sally,x)) & r(Sally, x))).

end_of_list.
```

Otter Output Using $Answer

- Show run on sallymorton2.txt

Keeping Track of Change

- Facts hold in <u>situations</u>, rather than eternally
  E.g., *Holding(Gold,Now)* rather than just *Holding(Gold)*
- *Situation calculus* is one way to represent change in FOPC:
  Adds a situation argument to each time-dependent predicate
  E.g., *Now* in *Holding(Gold,Now)* denotes a situation
- Situations are connected by the *Result* function
  Result(a,s) is the situation that results from doing *a* in *s*



Describing Actions

- Effect axiom: describe changes due to action
  $\forall s\ \text{AtGold}(s) \Rightarrow \text{Holding}(\text{Gold}, \text{Result}(\text{Grab}, s))$
- Frame axiom: describe **non-changes** due to action
  $\forall s\ \text{HaveArrow}(s) \Rightarrow \text{HaveArrow}(\text{Result}(\text{Grab}, s))$
- Frame problem: find an elegant way to handle non-change
  (a) Representation: avoid frame axioms
  (b) Inference: avoid repeated "copy-overs" to keep track of state
- Qualification problem: true descriptions of real actions require endless caveats - what if gold is slippery or nailed down or…
- Ramification problem: real actions have many secondary consequences - what about the dust on the gold, wear and tear on gloves,…

Making Plans

- Initial condition in KB:
  At(Agent,[1,1], S_0)
  At(Gold,[1,2], S_0)
- Query: *ASK(KB, Holding(Gold,s))* i.e., in what situation will I be holding the gold?
- Answer: {*s/Result*(*Grab,Result*(*Forward,*S_0))}
  i.e., go forward and then grab the gold
- This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Generating Plans: A Better Way

- Represent **plans** as action sequences $[a_1,a_2,\ldots,a_n]$
- *PlanResult*(*p*, *s*) is the result of executing plan *p* in state *s*
- Then the query *ASK(KB,* $\exists p$ *Holding(Gold, PlanResult(p,*S_0*)))* generates a solution similar to {*p*/[*Forward,Grab*]}
- Definition of *PlanResult* in terms of *Result*:
  $\forall$s *PlanResult*([], s) = s
  $\forall$a,p,s PlanResult([a|p], s) = PlanResult(p, Result(a, s))
- **Planners** are special-purpose reasoners designed to do this type of inference more efficiently than general-purpose reasoners.

Other Common Knowledge Representations

- Many problem solving tasks can be characterized as
  Knowledge Representation + Search
- The choice of KR can dramatically affect the ease of solving the problem.
- Propositional logic and FOPC are two common KRs, and there are several others discussed in Ch10 of AIMA 2ed.

Review

- First-Order Predicate Calculus
  - Universal and Existential Quantification
  - Universal and Existential Instantiation
- Proving with AI
  - Unification
  - Eliza
  - FOPC Conversion to CNF and Resolution
  - Otter
  - Proving with Actions (planning)
- Coming up Expert Systems