Resolution in FOL

Logic in Computer Science

1

Inference Rule of Resolution

- Reminder: Resolution rule for propositional logic:
  - Parent 1: $(P_1 \vee P_2 \vee ... \vee P_n)$
  - Parent 2: $(\neg P_1 \vee Q_2 \vee ... \vee Q_m)$
  - Resolvent: $(P_2 \vee ... \vee P_n \vee Q_2 \vee ... \vee Q_m)$
- Resolution in FOL:
  - Parent 1: $(P_1 \vee P_2 \vee ... \vee P_n)$
  - Parent 2: $(\neg Q_1 \vee Q_2 \vee ... \vee Q_m)$
  - Resolvent: $(P_2 \vee ... \vee P_n \vee Q_2 \vee ... \vee Q_m)\theta$
    - » if θ is a mgu of P_1 and Q_1
    - Parent 1: $P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$
    - Parent 2: $\neg P(z, f(u)) \vee \neg Q(z)$
    - Resolvent: $P(z, f(y)) \vee Q(y) \vee \neg Q(z)$ using θ = [x/z, u/a]

2

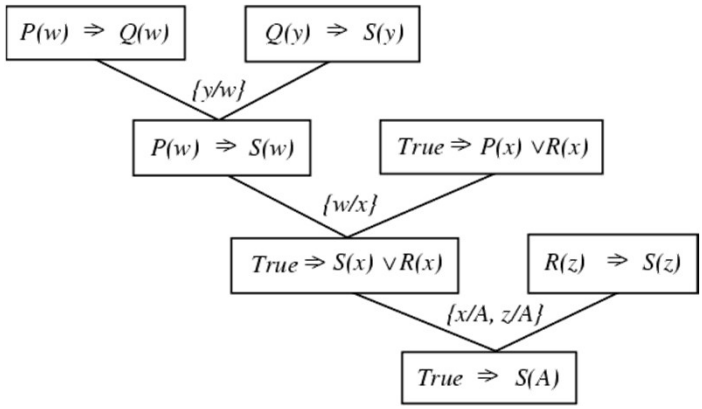
2

A resolution proof tree

Let C = { (-$P(w)$ | $Q(w)$), (-$Q(y)$ | $S(y)$), ($P(x)$ | $R(x)$), (-$R(z)$ | $S(z)$) }

C { $P(w) \equiv \rightarrow Q(w)$, $Q(y) \rightarrow S(y)$, $1 \rightarrow P(x) \vee R(x)$, $R(z) \rightarrow S(z)$ }

Prove that C |= $S(A)$ for any A.



3

3

An Example

- Assumptions:
  - Fido is a dog
  - All dogs are animals
  - All animals will die
- Goal:
  - Fido will die

Translate in FOL:

- Assumptions:
  - dog(fido)
  - ∀X dog(X) → animal(X)
  - ∀X animal(X) → die(X)
- Goal:
  - die(fido)

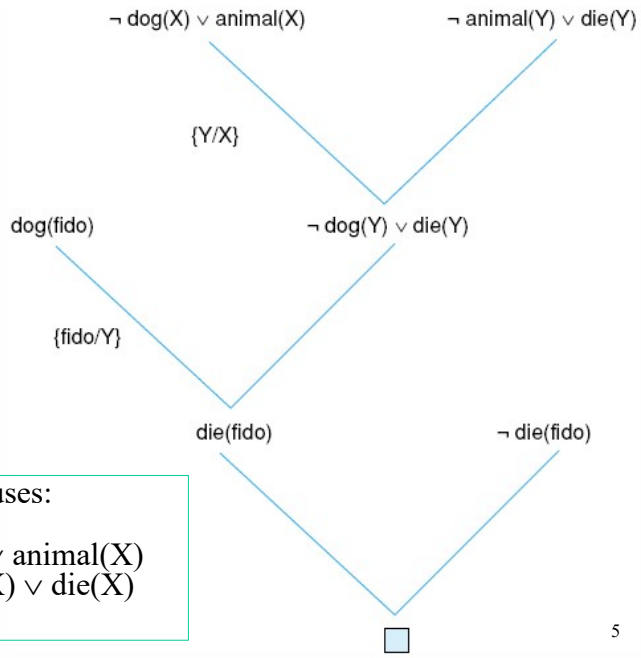
Convert to clauses:

1. dog(fido)
2. ¬dog(X) ∨ animal(X)
3. ¬animal(X) ∨ die(X)
4. ¬die(fido)
   // negation of Goal

4

4

Resolution proof for the "dog Fido" problem.



Convert to clauses:
1. dog(fido)
2. ¬dog(X) ∨ animal(X)
3. ¬animal(X) ∨ die(X)
4. ¬die(fido)

5

5

Resolution as a Refutation Prover

- **Proof by contradiction:**
  - **validity:** To show a formula X is valid, we show that ¬X is unsatisfiable.
  - **entailment:** Given a consistent set A of axioms and goal B, to show A |= B, we show that (A ∧ ¬ B) is unsatisfiable.

- Using **Resolution**:
  1. Convert either ¬X or (A ∧ ¬ B) into a set S of clauses;
  2. Derive the empty clause from S.

6

6

Resolution Example

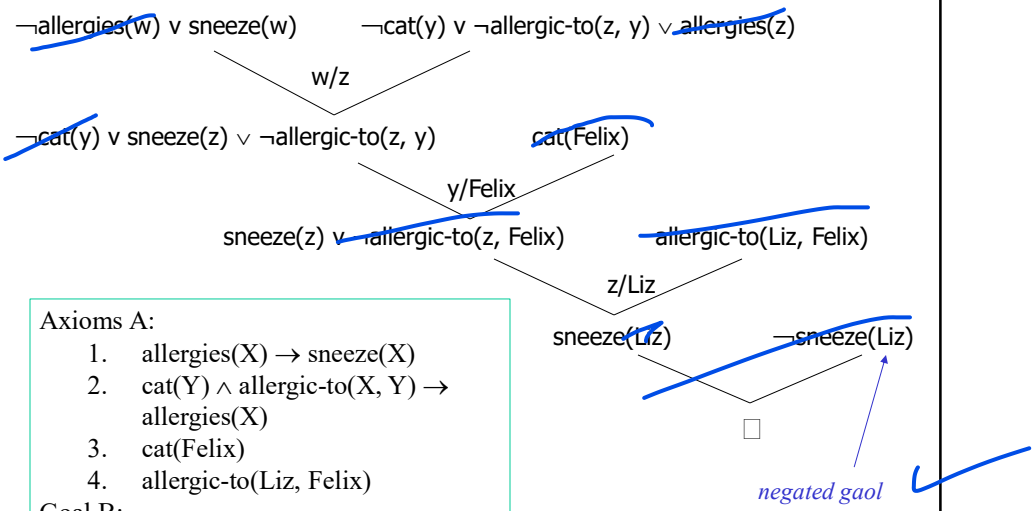
- Axioms A:
  1. allergies(X) → sneeze(X)
  2. cat(Y) ∧ allergic-to(X, Y) → allergies(X)
  3. cat(Felix)
  4. allergic-to(Liz, Felix)
- Goal B:

  sneeze(Liz)

7

7

Resolution Proof Tree



Axioms A:
1. allergies(X) → sneeze(X)
2. cat(Y) ∧ allergic-to(X, Y) → allergies(X)
3. cat(Felix)
4. allergic-to(Liz, Felix)

Goal B:
sneeze(Liz)

8

8

Inference Rule of Factoring

- **Resolution is not complete in FOL.**
  - E.g., S = { (p(x) | p(y)), (¬p(u) | ¬p(v)) } is unsatisfiable:
  - Sθ = { (p(a) | p(a)), (¬p(a) | ¬p(a)) } = { (p(a)), (¬p(a)) } with θ = [x/a, y/a, u/a, v/a].
  - Resolvent from S: (p(z) | ¬p(w)) —— no other resolvents
- **Factoring Rule**:
  - Premise: $(A_1 \vee A_2 \vee ... \vee A_n)$
  - Result: $(A_1 \vee A_2 \vee ... \vee A_n)\theta$

    if θ is a mgu of A_i and A_j, $i \neq j$

- Ex 1: (p(x) | p(y)) generates (p(x)) with θ = [y/x]
- Ex 2: (¬p(u) | ¬p(v)) generates (¬p(u)) with θ = [v/u]

9

9

Russell's Paradox

A barber shaves men if and only if they do not shave themselves. Should the barber shave himself or not?

Translate in FOL:

(A1) $\forall x \neg Shaves(x, x) \rightarrow Shaves(barber, x)$

(A2) $\forall y Shaves(barber, y) \rightarrow \neg Shaves(y, y)$

Convert to Clauses:

(C1) $Shaves(x, x) \vee Shaves(barber, x)$

(C2) $\neg Shaves(barber, y) \vee \neg Shaves(y, y)$

Four possible resolutions between C1 and C2:

(C1.1, C2.1) $Shaves(barber, barber) \vee \neg Shaves(barber, barber)$

(C1.1, C2.2) $Shaves(barber, x) \vee \neg Shaves(barber, x)$

(C1.2, C2.1) $Shaves(barber, x) \vee \neg Shaves(barber, x)$

(C1.2, C2.2) $Shaves(barber, barber) \vee \neg Shaves(barber, barber)$

32

10

Russell's Paradox

A barber shaves men if and only if they do not shave themselves. Should the barber shave himself or not?

Translate in FOL:

(A1) $\forall x \neg \text{Shaves}(x, x) \rightarrow \text{Shaves}(\text{barber}, x)$

(A2) $\forall y \text{ Shaves}(\text{barber}, y) \rightarrow \neg \text{ Shaves}(y, y)$

Convert to Clauses:

(C1) $\text{Shaves}(x, x) \vee \text{Shaves}(\text{barber}, x)$

(C2) $\neg\text{Shaves}(\text{barber}, y) \vee \neg\text{Shaves}(y, y)$

Factoring:

(From C1): Shaves(barber, barber)

(From C2): ¬Shaves (barber, barber)

Now it's easy to get the empty clause.

32

11

Soundness and Completeness of Resolution and Factoring

- Both **Resolution** and **Factoring** are sound.
  - If a clause C can be derived from S by either resolution or factoring, then $S \models C$ and $S \equiv S \cup \{C\}$.
- **Resolution and Factoring are refutation complete:** It can establish that S is unsatisfiable but cannot show that S is satisfiable. Thus, it cannot show that B is **not entailed** by A.
- **Theorem**: Resolution and Factoring define a sound and refutation complete inference system.
- **Factoring** is needed for completeness proof but rarely needed in practice.

12

12

Resolution and Factoring as a Search Procedure

- Resolution and factoring can be thought of as a search procedure with the empty clause as the search goal.
- The resolution proof tree is a demonstration of how the empty clause is found, starting from the clauses generated from the axioms and the negation of the theorem.
- When a pair of clauses generates a new resolvent clause, a new node is added to the tree with arcs directed from the two parent clauses to the resolvent.
- When a clause C' is generated from C by factoring, a new node is added to the tree with arc from C to C'.

13

13

Resolution Strategies

- A **strategy** defines the policy for picking two clauses for resolution.
- A strategy is **complete** if its use guarantees that the empty clause can be derived whenever it is entailed.
- There carried over from Propositional Logic:
  - Breadth-first
  - Depth-first/Linear Resolution
  - Length heuristics/Unit Resolution
  - Input resolution
  - Positive/Negative Resolution
  - Set of support
  - Ordered resolution

14

14

Breadth-First Search

- Definition of Level k:
  - Level 0 clauses are the original input clauses
  - Level k clauses are the resolvents computed from two clauses, one of which must be from level k-1 and the other from any level < k.
- Breadth-first: Compute all possible level 1 clauses, then all possible level 2 clauses, etc.
- Complete (with factoring), but very inefficient

15

15

Depth-First Search/Linear Resolution

- One parent clause of the resolution is the latest clause, like linear resolution
- Resolve literals in this clause in order (left to right)
- The other parent clause is in order (first to last); linear resolution uses any clause
- Linear resolution is complete with factoring
- Depth-first search is incomplete even with factoring
- This is how Prolog operates
- This forces the user to place what is important in a clause; the way the clauses are written controls the resolution

16

16

Length Heuristics/Unit Resolution

- **Shortest-clause heuristic**:
  Generate a clause with the fewest literals first
- **Unit resolution**:
  Prefer resolution steps in which at least one parent clause is a "unit clause," i.e., a clause containing a single literal
  - Not complete in general, but complete (without factoring) for Horn clauses

17

17

Horn clauses

- A Horn clause is a sentence of the form:

  $\neg P_1(x) \vee \neg P_2(x) \ldots \vee \neg P_n(x) \vee Q(x)$

  or $P_1(x) \wedge P_2(x) \wedge \ldots \wedge P_n(x) \rightarrow Q(x)$

  where
  - there are 0 or more P_is and 0 or 1 Q
  - the P_is and Q are atoms (positive literals)
- Special cases:
  - $P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow$ false
  - true $\rightarrow$ Q
- Non-Horn clauses: $p(a) \vee q(a)$, $p(x) \wedge q(x) \rightarrow r(x) \vee s(x)$
- Prolog is based on Horn clauses

18

18

Input Resolution

- **Input resolution**
  - At least one parent must be one of the input clauses.
  - Not complete in general, but complete (without factoring) for Horn clauses
  - Prolog uses this strategy
- **Theorem**: Unit resolution and input resolution are equivalent on completeness: The existence of one proof implies the existence of the other.

19

19

Positive/Negative Resolution

- **Positive Resolution**:
  One parent of the resolution must be positive (all literals are positive)
- **Negative Resolution**:
  One parent of the resolution must be negative (all literals are negative)
  - Prolog uses this strategy
- Both are complete (with factoring)

20

20

Set of Support

- A subset of input clauses is designed as the set of support. Usually, the clauses derived from the negation of the theorem are in the set-of-support.
- At least one parent clause must be in the set-of-support (i.e., derived from a goal clause).
- Resolvents are always put into the set of support.
- Prolog uses this strategy
- Complete with factoring (assuming all possible set-of-support clauses are derived)
- Gives a goal-directed character to the search

21

21

Ordered Resolution

- The user provides a partial order $>$ over atomic formulas.
  - $>$ is stable under substitution: $A > B$ implies $A\theta > B\theta$ for any θ.
  - $>$ is total on ground formulas (Herbrand base).
  - $>$ is well-founded: no infinite sequences of $A_1 > A_2 > \ldots > A_n > \ldots$
- Ordered Resolution:
  - Parent 1: $(P_1 \vee P_2 \vee \ldots \vee P_n)$
  - Parent 2: $(\neg Q_1 \vee Q_2 \vee \ldots \vee Q_m)$
  - Resolvent: $(P_2 \vee \ldots \vee P_n \vee Q_2 \vee \ldots \vee Q_m)\theta$
    - if θ is a mgu of P_1 and Q_1, and both P_1 and Q_1 are maximal in the parent clauses.

22

22

"Happy Student" Problem

- Anyone passing his logic exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. John did not study but he is lucky. Anyone who is lucky wins the lottery.
- Is John happy?

23

23

Anyone passing his history exams and winning the lottery is happy.

$\forall$ **X (pass (X,history)** $\wedge$ **win (X,lottery)** $\rightarrow$ **happy (X))**

Anyone who studies or is lucky can pass all his exams.

$\forall$ **X** $\forall$ **Y (study (X)** $\vee$ **lucky (X)** $\rightarrow$ **pass (X,Y))**

John did not study but he is lucky.

$\neg$ **study (john)** $\wedge$ **lucky (john)**

Anyone who is lucky wins the lottery.

$\forall$ **X (lucky (X)** $\rightarrow$ **win (X,lottery))**

These four predicate statements are now changed to clause form (Section 12.2.2):

1. $\neg$ **pass (X, history)** $\vee$ $\neg$ **win (X, lottery)** $\vee$ **happy (X)**
2. $\neg$ **study (Y)** $\vee$ **pass (Y, Z)**
3. $\neg$ **lucky (W)** $\vee$ **pass (W, V)**
4. $\neg$ **study (john)**
5. **lucky (john)**
6. $\neg$ **lucky (U)** $\vee$ **win (U, lottery)**

Into these clauses is entered, in clause form, the negation of the conclusion:

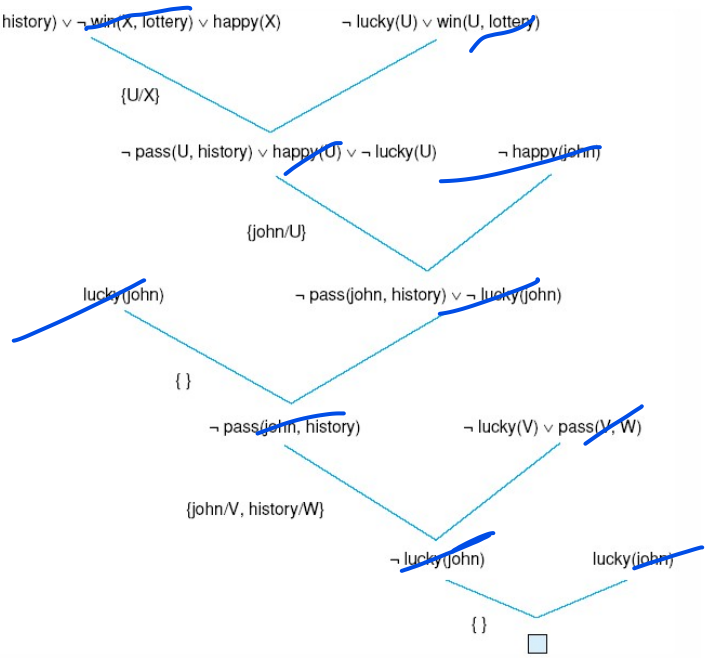
7. $\neg$ **happy (john)**

24

CSC411

24

Refutation proof tree for the "happy student" problem.



25

Exciting life problem

- Those people who read are not stupid. John can read and is wealthy. All people who are not poor and are smart are happy. Happy people have exciting lives. Can anyone be found with an exciting life?
- Translate the statements in FOL:
  - Assume $\forall x$ wealthy(x) $\equiv \neg$poor(x),
    $\forall x$ stupid(x) $\equiv \neg$smart(x)
  - $\forall y$ read(y) $\rightarrow$ smart(y)
  - read(John) $\wedge \neg$poor(John)
  - $\forall x \neg$poor(x) $\wedge$ smart(x) $\rightarrow$ happy(x)
  - $\forall z$ happy(z) $\rightarrow$ exciting(z)
  - $\exists w$ exciting(w) // conclusion.

26

26

Exciting life problem

- $\forall y\ read(y) \rightarrow smart(y)$
- $read(John) \wedge \neg poor(John)$
- $\forall x\ \neg poor(x) \wedge smart(x) \rightarrow happy(x)$
- $\forall z\ happy(z) \rightarrow exciting(z)$
- $\exists w\ exciting(w)$ // conclusion.

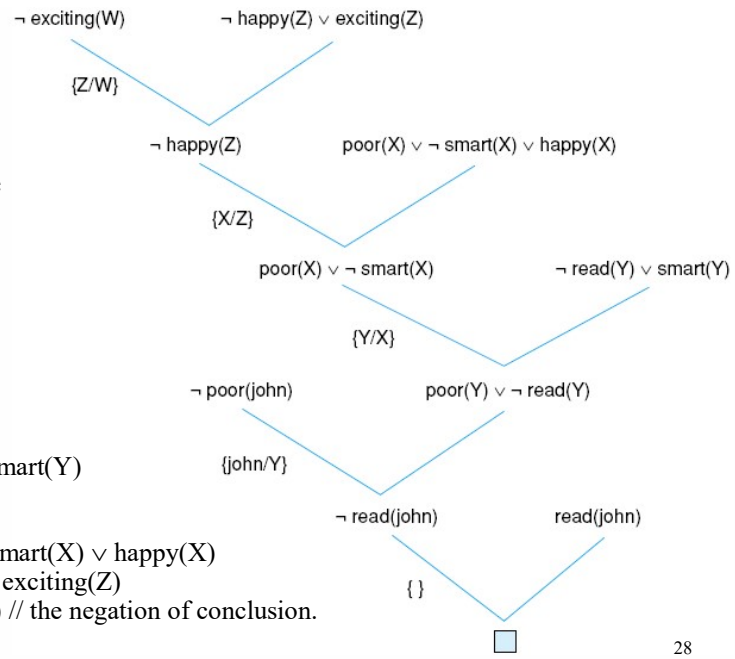
Convert into clauses:

- $\neg read(y) \vee smart(y)$
- $read(John)$
- $\neg poor(John)$
- $poor(x) \vee \neg smart(x) \vee happy(x)$
- $\neg happy(z) \vee exciting(z)$
- $\neg exciting(w)$ // the negation of conclusion.

27

27

Resolution proof for the "exciting life" problem.

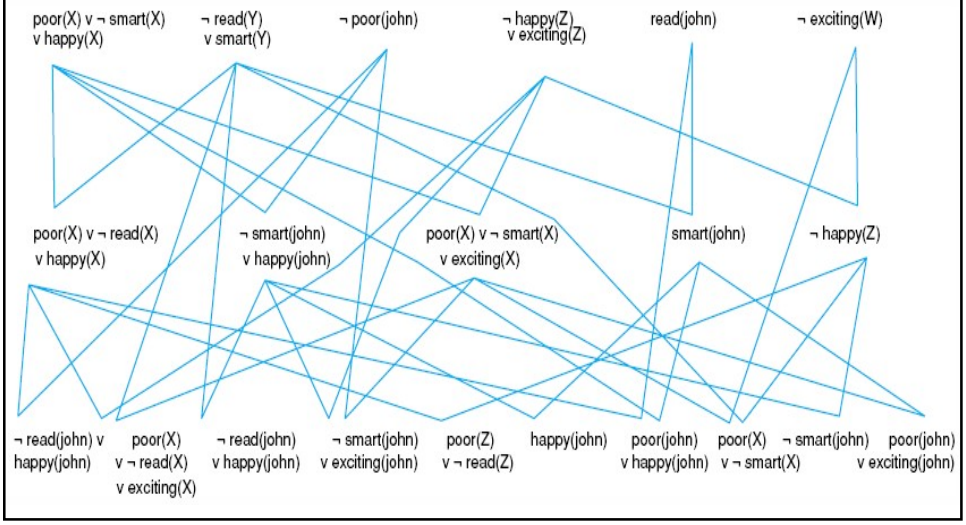


¬read(Y) ∨ smart(Y)
read(John)
¬poor(John)
poor(X) ∨ ¬smart(X) ∨ happy(X)
¬happy(Z) ∨ exciting(Z)
¬exciting(W) // the negation of conclusion.

28

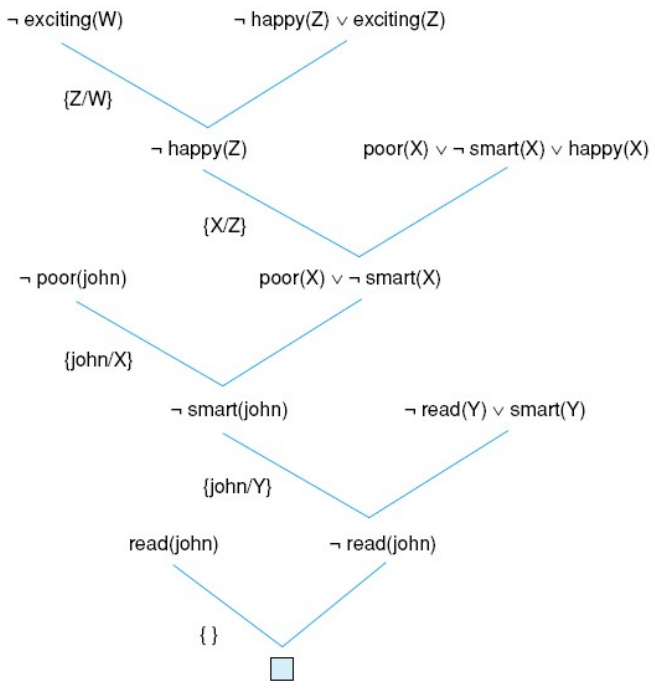
28

Clauses generated up to two levels for the "exciting life" problem by breadth-first search



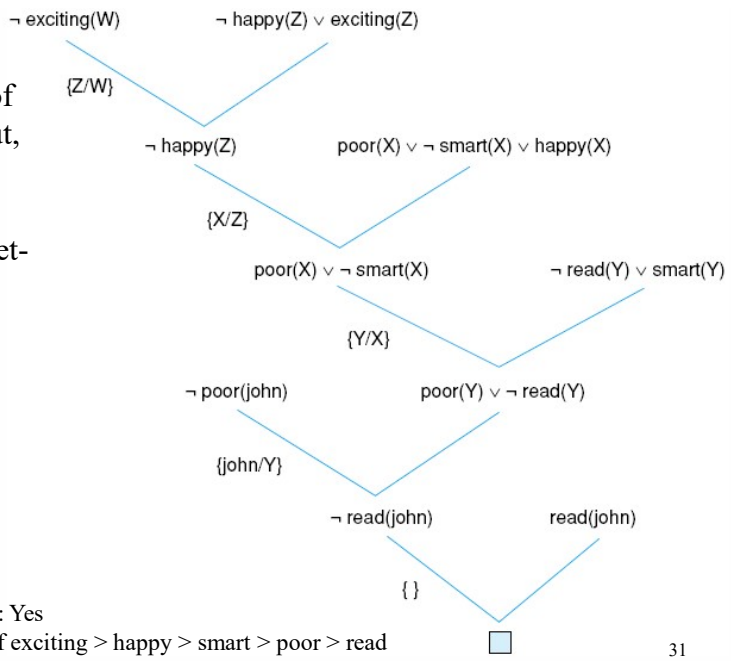
29

The unit, input, linear, negative, set-of-support resolution proofs on the "exciting life" problem.



30

Is this proof a unit, input, linear, positive, negative, set-of-support, or ordered resolution proof?

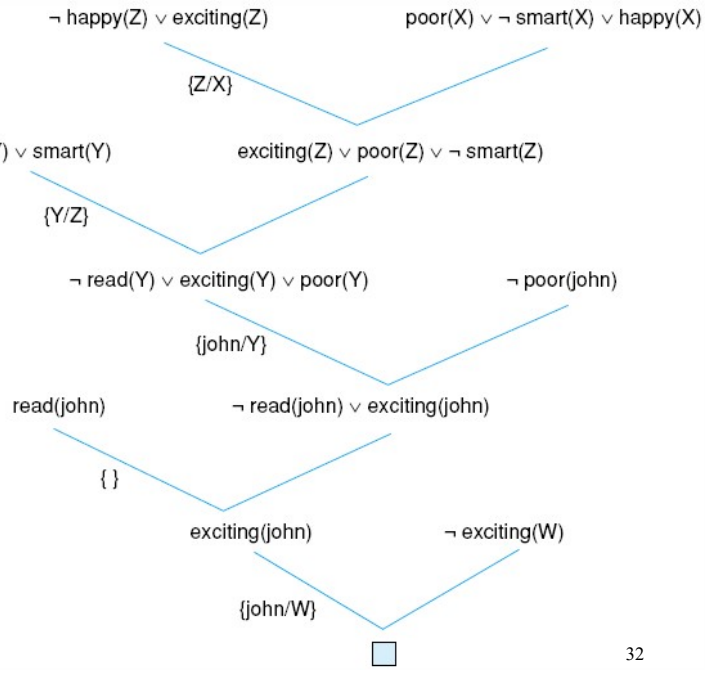


Unit: No
Input: Yes
Linear: Yes
Positive: No
Negative: No
Set-of-support: Yes
Ordered: Yes if exciting > happy > smart > poor > read

31

31

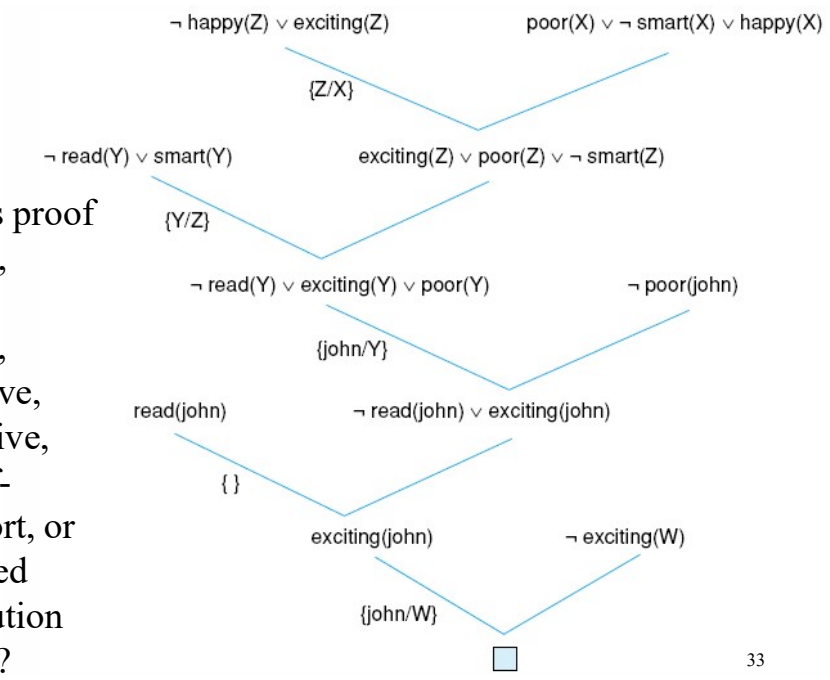
Another resolution proof for the "exciting life" problem



32

32

Is this proof a unit, input, linear, positive, negative, set-of-support, or ordered resolution proof?



33

33

Answer Extraction

- Extract correct answer to a problem from a resolution refutation by retaining information on the unification substitutions made in the resolution refutation
- "Exciting life problem"
  - Those people who read are not stupid. John can read and is wealthy. All people who are not poor and are smart are happy. Happy people have exciting lives.
  - Who can be found with an exciting life?
  - **Answer**: John.

34

34

Simplification Strategies

- **Tautology deletion:**
  Remove any clause containing two complementary literals (tautology)
- **Pure literal deletion:**
  If a predicate symbol always appears with the same "sign," remove all the clauses that contain it
- **Subsumption:**
  Eliminate all clauses that are subsumed by another clause (details in the next slide)

35

35

Subsumption

- Propositional Logic: clause B is subsumed by another clause A if every literal of A appears in B.
  - E.g., (p | q | r) is subsumed by (p | q).
- FOL: clause B is subsumed by another clause A if there exists a substitution such that every literal of Aθ appears in B.
  - (p(y) | q(y) | r(y)) is subsumed by (p(x) | q(x)) with θ = [x/y]
  - (p(A) | q(A)) is subsumed by (p(x) | q(x)) with θ = [x/A]
  - (p(f(a), f(a)) is subsumed by (p(f(x), y) | p(y, f(x))) with θ = [x/a, y/f(a)].
  - Factoring rule will generate (p(f(x), f(x))) from (p(f(x), y) | p(y, f(x))) with θ = [y/f(x)].

36

36

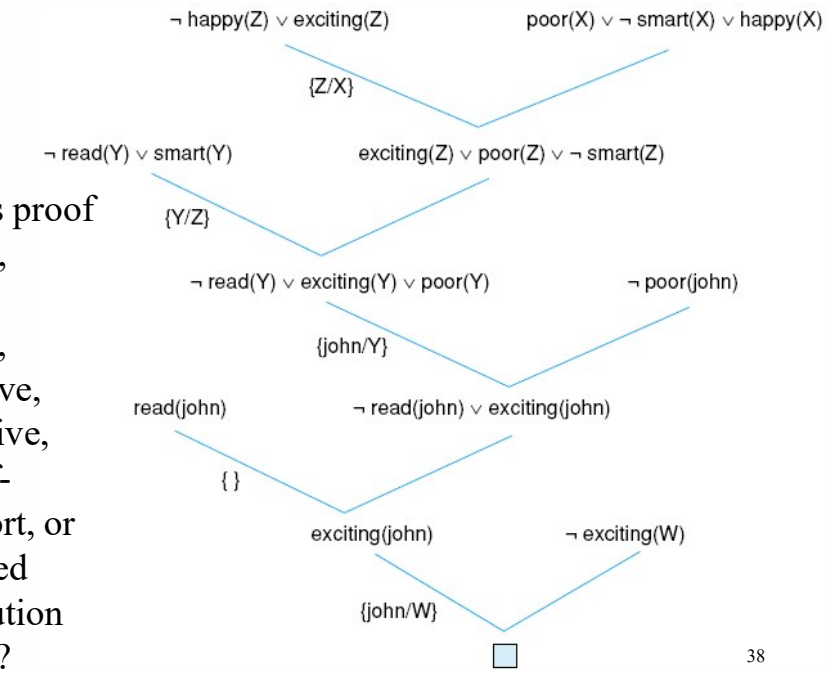
Unit Deletion

- Propositional Logic: literal A can be deleted from (A | α) if (¬A) is a unit clause, since unit resolution can generate (α) which subsumes (A | α).
- FOL: literal A can be deleted from (A | α) if (¬B) is a unit clause, and there exists substitution θ such that A = Bθ since unit resolution can generate (α) which subsumes (A | α).
- Example: p(a) can be removed from (p(a) | q(y) | r(y)) if there exists a clause (-p(x)).

37

37

Is this proof a unit, input, linear, positive, negative, set-of-support, or ordered resolution proof?



38

38

Simplification Strategies

- **Tautology deletion:**
  Remove any clause containing two complementary literals (tautology)
- **Pure literal deletion:**
  If a predicate symbol always appears with the same "sign," remove all the clauses that contain it
- **Subsumption:**
  Eliminate all clauses that are subsumed by another clause
- **Unit deletion:**
  Eliminate all literals whose complements are instances of unit clauses (details in the next slide)

39

39

Resolution Algorithm

```
proc resolution(C)
 1 G := preprocessing(C); // G: given clauses
 2 K := ∅; // K: kept clauses
 3 while G ≠ ∅ do
 4    A := pickClause(G); // heuristic for picking a clause
 5    G := G – {A};
 6    N := ∅; // new clauses from A and K by resolution
 7    for B∈ K if resolvable(A,B) do
 8       D := resolve(A,B);
 9       if D = ( ) return false; // the empty clause is found
10       if subsumedBy(D, G ∪ K) continue;
11       N := N ∪ factoring(D);
12    K := K ∪ {A};
13    for A ∈ G if subsumedBy(A, N) do G := G – {A};
14    for B ∈ K if subsumedBy(B, N) do K := K – {B};
15    G := G ∪ N;
16 return true; // K is saturated by resolution
```

40

Life Cycle of a Clause in *Resolution*

```
proc resolution(C)
 1 G := preprocessing(C); // G: given clauses
 2 K := ∅; // K: kept clauses
 3 while (G ≠ ∅) do
 4    A := pickClause(G);
 5    G := G – {A};
 6    N := ∅; // new clauses from A and K
 7    for B∈ K if resolvable(A,B) do
 8       D := resolve(A,B);
 9       if D = ( ) return false;
10       if subsumedBy(D, G ∪ K) continue;
11       N := N ∪ factoring(D);
12    K := K ∪ {A};
13    for A ∈ G if subsumedBy(A, N) do G := G – {A};
14    for B ∈ K if subsumedBy(B, N) do K := K – {B};
15    G := G ∪ N;
16 return true; // K is saturated by resolution
```

C: input clauses
N: New clauses
  Line 13-14: work
 D: member of N, line 11
  Line 8: birth
  Line 10: Pass test
G: Given clauses
  Line 1: birth from C
  Line 15: birth from N
  Line 10: work
 A: member of G, line 4
  Line 8: reproduce
  Line 13: death
  Line 12: promotion
K: Kept clauses
  Line 10: work
 B: member of G, line 7
  Line 8: reproduce
  Line 14: death

41

Example: Hoofers Club

- **Problem Statement:** Tony, Tom and Liz belong to the Hoofers Club. Every member of the Hoofers Club is either a skier or a mountain climber or both. No mountain climber likes rain, and all skiers like snow. Liz dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow.

- **Query:** Is there a member of the Hoofers Club who is a mountain climber but not a skier?

42

Example: Hoofers Club…

Translation into FOL Sentences (closed formulas)

Let **S(x)** mean **x** is a skier, **M(x)** mean x is a mountain climber, and **L(x,y)** mean **x** likes **y**, where the domain of **x** is Hoofers Club members, and the domain of **y** is snow and rain. We can now translate the English sentences into the following FOL :

1. $\forall x\ S(x) \vee M(x)$
2. $\neg(\exists x\ M(x) \wedge L(x, Rain))$
3. $\forall x\ S(x) \rightarrow L(x, Snow)$
4. $\forall y\ L(Liz, y) \leftrightarrow \neg\ L(Tony, y)$
5. L(Tony, Rain)
6. L(Tony, Snow)
7. Goal: $\exists x\ M(x) \wedge \neg S(x)$
8. Negation of the Goal: $\neg(\exists x\ M(x) \wedge \neg S(x))$

43

Example: Hoofers Club…

Conversion to Clause Form and Resolution…

1. (S(x1) | M(x1))
2. (~M(x2) | ~L(x2, Rain))
3. (~S(x3) | L(x3, Snow))
4. (~L(Tony, x4) | ~L(Liz, x4))
5. (L(Tony, x5) | L(Liz, x5))
6. (L(Tony, Rain))
7. (L(Tony, Snow))
8. (~M(x8) | S(x8)) // Negation of the Goal
9. (S(x1)) // 1 and 8 with $\theta_1 = [x8/x1]$
10. (L(x1, Snow)) // 3 and 9 with $\theta_2 = [x3/x1]$
11. (~L(Tony, Snow)) // 4 and 10 with $\theta_3 = [x1/Liz, x4/Snow]$
12. □ // 7 and 11

Answer Extraction:
(x8)θ = ?
$\theta = \theta_1\theta_2\theta_3$
= [x8/Liz, x3/Liz, x1/Liz, x4/Snow]

44

Unsatisfiable Core: unsatisfiable subset of clauses

Conversion to Clause Form and Resolution…

1. (S(x1) | M(x1))
2. (~M(x2) | ~L(x2, Rain))
3. (~S(x3) | L(x3, Snow))
4. (~L(Tony, x4) | ~L(Liz, x4))
5. (L(Tony, x5) | L(Liz, x5))
6. (L(Tony, Rain))
7. (L(Tony, Snow))
8. (~M(x8) | S(x8)) // Negation of the Goal
9. (S(x1)) // 1 and 8 with θ_1 = [x8/x1]
10. (L(x1, Snow)) // 3 and 9 with θ_2 = [x3/x1]
11. (~L(Tony, Snow)) // 4 and 10 with θ_3 = [x1/Liz, x4/Snow]
12. □ // 7 and 11

From resolution proof, we can identify a core of 5 clauses: 1, 3, 4, 7, 8

45

Prover9

- Prover9 is a state-of-the-art automated theorem prover for first-order logic based on resolution.
- The author of Prover9 is Bill McCune.
- Prover9 is the successor of the Otter prover, also written by McCune:
- http://www.cs.unm.edu/~mccune/prover9/

33

46

Russell's Paradox in Prover9

Input:

-Shaves(x,x) -> Shaves(barber,x).

Shaves(barber,y) -> - Shaves(y,y).

Output:

1 -Shaves(x,x) -> Shaves(barber,x) # label(non_clause). [assumption].

2 Shaves(barber,x) -> -Shaves(x,x) # label(non_clause). [assumption].

3 Shaves(x,x) | Shaves(barber,x). [clausify(1)].

4 -Shaves(barber,x) | -Shaves(x,x). [clausify(2)].

5 Shaves(barber,barber). [factor(3,a,b)].

6 $F. [factor(4,a,b),unit_del(a,5)].

34

47

Cat Tuna Problem

Did Curiosity kill the cat?

- 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?
- These can be represented as follows:

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

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

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

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

E. $Cat(Tuna)$

F. $\forall x\ Cat(x) \rightarrow Animal(x)$

G. $Kills(Curiosity, Tuna)$ ⟵ GOAL

48

48

Cat Tuna Problem

A. $\exists x\ Dog(x) \wedge Owns(Jack, x)$
B. $\forall x\ (\exists y\ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
C. $\forall x\ AnimalLover(x) \rightarrow (\forall y\ Animal(y) \rightarrow \neg Kills(x, y))$
D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x\ Cat(x) \rightarrow Animal(x)$
G. $Kills(Curiosity, Tuna)$

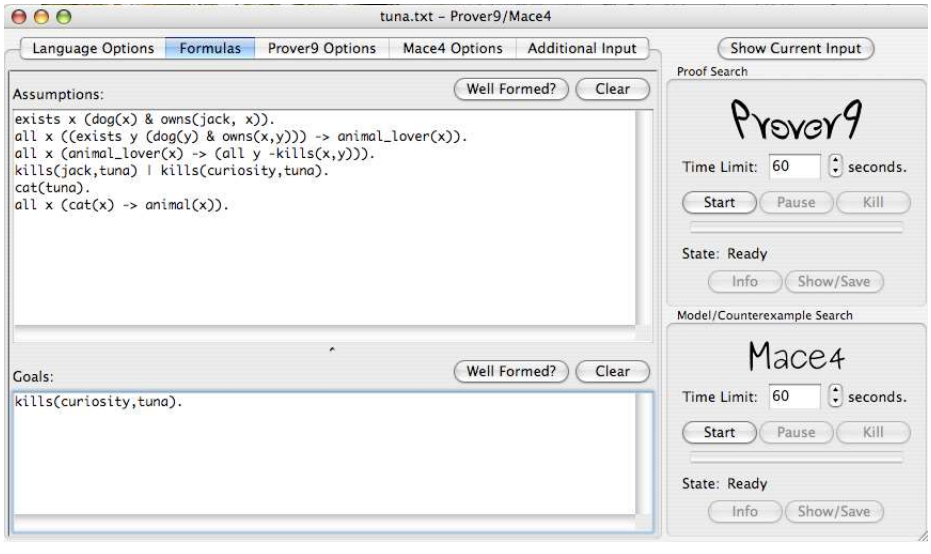
Prover9's input:

```
exists x (dog(x) & owns(Jack, x)).
all x (exists y (dog(y) & owns(x, y)) -> animalLover(x)).
all x all y (animalLover(x) -> (animal(y) -> -kills(x, y))).
kills(Jack, Tuna) | kills(Curiosity, Tuna).
cat(Tuna).
all x (cat(x) -> animal(x)).
-kills(Curiosity, Tuna).
```

49

49

Cat Tuna Example: Prover9 Input



50

Prover9's Proof

```
% ------------------- Proof -------------------
1 (exists x (dog(x) & owns(Jack,x))) # label(non_clause).
[assumption].
2 (all x ((exists y (dog(y) & owns(x,y))) -> animalLover(x))) #
label(non_clause).  [assumption].
3 (all x (animalLover(x) -> (all y (animal(y) -> -kills(x,y))))) #
label(non_clause).  [assumption].
4 (all x (cat(x) -> animal(x))) # label(non_clause).  [assumption].
5 -dog(x) | -owns(y,x) | animalLover(y).  [clausify(2)].
6 dog(c1).  [clausify(1)].
7 -owns(x,c1) | animalLover(x).  [resolve(5,a,6,a)].
8 owns(Jack,c1).  [clausify(1)].
9 animalLover(Jack).  [resolve(7,a,8,a)].
10 -animalLover(x) | -animal(y) | -kills(x,y).  [clausify(3)].
11 -cat(x) | animal(x).  [clausify(4)].
12 cat(Tuna).  [assumption].
13 animal(Tuna).  [resolve(11,a,12,a)].
14 -animal(x) | -kills(Jack,x).  [resolve(9,a,10,a)].
15 kills(Jack,Tuna) | kills(Curiosity,Tuna).  [assumption].
16 -kills(Curiosity,Tuna).  [assumption].
17 -kills(Jack,Tuna).  [resolve(13,a,14,a)].
18 $F.  [back_unit_del(15),unit_del(a,17),unit_del(b,16)].
```

51

51

Cat Tuna Problem

Did Curiosity kill the cat?

- **Convert to clause form**
  - A1. (Dog(D)) ← D is a skolem constant
  - A2. (Owns(Jack,D))
  - B. (¬Dog(y) | ¬Owns(x, y) | AnimalLover(x))
  - C. (¬AnimalLover(u) | ¬Animal(v) | ¬Kills(u, v))
  - D. (Kills(Jack, Tuna) | Kills(Curiosity, Tuna))
  - E. Cat(Tuna)
  - F. (¬Cat(z) | Animal(z))
- **Add the negation of query:**
  - ¬G: (¬Kills(Curiosity, Tuna))

52

A resolution refutation proof

A1. (Dog(D))

A2. (Owns(Jack, D))

B. (¬Dog(y) | ¬Owns(x, y) | AnimalLover(x))

C. (¬AnimalLover(u) | ¬Animal(v) | ¬Kills(u, v))

D. (Kills(Jack, Tuna) | Kills(Curiosity, Tuna))

E. Cat(Tuna)

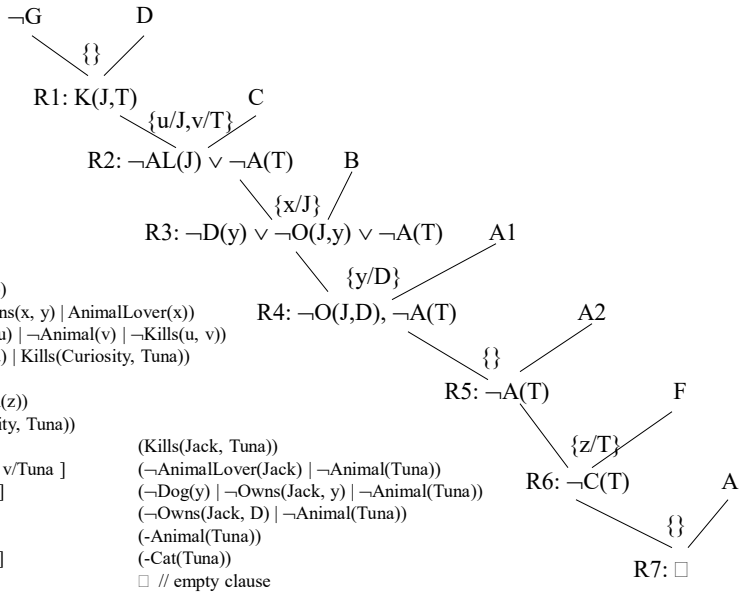
F. (¬Cat(z) | Animal(z))

¬G: (¬Kills(Curiosity, Tuna))

R1: ¬G, D, [ ]	(Kills(Jack, Tuna))
R2: R1, C, [u/Jack, v/Tuna]	(¬AnimalLover(Jack) | ¬Animal(Tuna))
R3: R2, B, [x/Jack]	(¬Dog(y) | ¬Owns(Jack, y) | ¬Animal(Tuna))
R4: R3, A1, [y/D]	(¬Owns(Jack, D) | ¬Animal(Tuna))
R5: R4, A2, [ ]	(-Animal(Tuna))
R6: R5, F, [z/Tuna]	(-Cat(Tuna))
R7: R6, E, [ ]	□ // empty clause

53

The resolution proof tree



A1. (Dog(D))
A2. (Owns(Jack, D))
B. (¬Dog(y) | ¬Owns(x, y) | AnimalLover(x))
C. (¬AnimalLover(u) | ¬Animal(v) | ¬Kills(u, v))
D. (Kills(Jack, Tuna) | Kills(Curiosity, Tuna))
E. Cat(Tuna)
F. (¬Cat(z) | Animal(z))
¬G: (¬Kills(Curiosity, Tuna))

R1: ¬G, D, [ ]	(Kills(Jack, Tuna))
R2: R1, C, [u/Jack, v/Tuna]	(¬AnimalLover(Jack) | ¬Animal(Tuna))
R3: R2, B, [x/Jack]	(¬Dog(y) | ¬Owns(Jack, y) | ¬Animal(Tuna))
R4: R3, A1, [y/D]	(¬Owns(Jack, D) | ¬Animal(Tuna))
R5: R4, A2, [ ]	(-Animal(Tuna))
R6: R5, F, [z/Tuna]	(-Cat(Tuna))
R7: R6, E, [ ]	□ // empty clause

54

Prover9's Resolution Strategies

- Binary resolution, Set-of-Support, Ordered resolution
- Hyper-resolution
  - **Hyper-resolution** *consists of a sequence of positive resolutions between one non-positive clause (called* nucleus*) and a set of positive clauses (called* satellites*), until a positive clause or the empty clause is produced.*
- Ex: ($r(a) \mid r(b) \mid r(c)$) is the result of hyper-resolution from

  {c1: ($-p(x) \mid -q(x) \mid r(x)$), c2: ($q(a) \mid r(b)$), c3: ($p(a) \mid r(c)$)}
  - c1 is the nucleus; c2 and c3 are satellites.
  - The resolvent from c1 and c2 is ($p(a) \mid r(a) \mid r(b)$), which produces the second resolvent ($r(a) \mid r(b) \mid r(c)$) with c3.

55

55

Prover9's Resolution Strategies

- Binary resolution, Set-of-Support, Ordered resolution
- Hyper-resolution
- Unit-resulting (ur) resolution
  - **Unit-resulting resolution** *consists of a sequence of unit resolutions between one non-unit clause (called* nucleus*) and a set of unit clauses (called* electrons*), until a unit clause or the empty clause is produced.*
- Ex: ($-p(\text{i}(a, b))$) is the result of ur-resolution from

  {c1: ($-p(x) \mid -p(\underline{\text{i}}(x, y)) \mid p(y)$), c2: ($\text{p}(a)$), c3: ($-p(b)$)}
  - c1 is the nucleus; c2 and c3 are electrons.
  - The resolvent from c1 and c2 is ($-p(i(a, y)) \mid p(y)$), which produces the second resolvent ($-p(\text{i}(a, b))$ with c3.

56

56

Applications of Prover9

- Bill McCune and Larry Wos
  - Argonne National Laboratories
  - FO resolution provers: EQP, Otter, Prover9
- **Robbins Problem** (axioms of Boolean algebras)
  - Stated 60+ years ago, mathematicians failed
  - 1996: EQP solved in 8 days in 1996 (+human work)
- General application to algebraic axiomatisations
  - Generate possible axioms for algebras
  - Prove new axioms equivalent to old

57

Umbrellas Puzzle

- You know the following things:
  - Someone living in your house is wet
  - If a person is wet, it is because of the rain, the sprinklers, or both
  - If a person is wet because of the sprinklers, the sprinklers must be on
  - If a person is wet because of rain, that person must not be carrying any umbrella
  - There is an umbrella in your house, which is not in the closet
  - An umbrella that is not in the closet must be carried by some person who lives in that house
  - Nobody are carrying any umbrella
- Can you conclude that the sprinklers are on?

58

(Extremely informal statement of) Gödel's Incompleteness Theorem

- First-order logic is not rich enough to model basic arithmetic.
- For any consistent system of axioms that is rich enough to capture basic arithmetic (in particular, mathematical induction), there exist true sentences that cannot be proved from those axioms.

59

A more challenging exercise

- Suppose:
  - There are exactly 3 objects in the world,
  - spouse(x) is a function, i.e., it give the spouse of x.
  - If x is the spouse of y, then y is the spouse of x, i.e., spouse defines a commutative relation.
- Prove:
  - Something is its own spouse

60

More challenging exercise

- There are exactly 3 objects in the world,
  - There exist x, y, z: (NOT(x=y) AND NOT(x=z) AND NOT (y=z))
  - for all w, x, y, z: (w=x OR w=y OR w=z OR x=y OR x=z OR y=z)
- If x is the spouse of y, then y is the spouse of x, i.e., spouse defines a commutative relation.
  - for all x, y: ((Spouse(x)=y) => (Spouse(y)=x))
- The negation of "Something is its own spouse"
  - for all x, y: ((Spouse(x)=y) => NOT(x=y))

61

Mace4: SAT Solver in FOL

- Mace4 accepts the first-order formulas as Prover9
- Mace4 converts the first-order formulas into propositional clauses assuming the domain is finite.
- Mace4 searches for a propositional model and converts the model as a finite model of FOL.
- Mace4 releases the burden of the user of writing encoder/decoder for using a SAT solver.
- Mace4 is a valuable complement to Prover9, looking for counterexamples of a FOL theorem
- Mace4 can be used to help debugging input clauses and formulas for Prover9.

62

62

FOL for Latin Square Problems

- A Latin square of size N is an N by N square such that each row and each column is a permutation of S ={ 0, 1, …, N-1}.
- A Latin square can be defined by function *: S, S→ S, x*y = z iff the entry at row x and column y is z.
- To specify that each row has no duplicate elements:
  - (x * z = y * z) → x = y.
- To specify that each column has no duplicate elements:
  - (x * y = x * z) → y = z.
- A Latin square can be encoded in propositional logic by variables $p_{x,y,z}$ such that $p_{x,y,z}$ = 1 iff x*y = z.
- In general, f(a, b, …, c) = y, we encode it by $p_{a,b,\ldots,c,y}$. if the domain is finite.

63

63

Mace4 Input for Latin Square Problems

```
% Mace4 produces 10944 Latin squares of size 5.
assign(domain_size, 5).
assign(max_models, -1).   % This means "no limit"
clear(print_models).

formulas(assumptions).
% Latin square axioms
x * z != y * z | x = y.     % from (x * z = y * z) → x = y
x * y != x * z | y = z.     % from (x * y = x * z) → y = z
end_of_list.
```

64

64

Mace4: Input for Latin squares with constraint qg5.

```
assign(domain_size, 8).

formulas(assumptions).
x * z != y * z | x = y.
x * y != x * z | y = z.
x * x = x.
(((y * x) * y) * y) = x.
end_of_list.
```

Mace4 output:

```
 [ function(*(_,_),
    [ 0, 2, 6, 5, 1, 4, 7, 3,
      7, 1, 5, 6, 2, 3, 0, 4,
      3, 6, 2, 1, 5, 7, 4, 0,
      1, 7, 4, 3, 0, 6, 2, 5,
      6, 3, 0, 7, 4, 1, 5, 2,
      2, 0, 3, 4, 7, 5, 1, 6,
      5, 4, 7, 0, 3, 2, 6, 1,
      4, 5, 1, 2, 6, 0, 3, 7 ])
]).
```

65

Mace4: Alternative Input for qg5

```
assign(domain_size, 8).
formulas(assumptions).
% Latin square axioms in equational form
x * (x \ y) = y.
x \ (x * y) = y.
(x / y) * y = x.
(x * y) / y = x.
x * x = x.
(((y * x) * y) * y) = x.
end_of_list.
```

66

How to Translate FOL Clauses into PL

1. The domain must be finite, i.e., { 0, 1, …, n-1 }
   assign(domain_size, 8).
2. Each clause must be flattened: each atom must be either $p(a_1, a_2, \ldots, a_n)$ or $f(a_1, \ldots, a_n) = b$, where a_i's are either variables or constants.
   x * (x \ y) = y becomes (x * z = y) | ¬(x \ y = z)
   (((y * x) * y) * y) = x becomes
   (u*y = x) | ¬(y*x = z) | ¬(z*y = u)
3. The functional axiom must be added for each function.
   ¬(x*y = u) | ¬(x*y = v) | (u = v) // each cell has one value
4. Replace each variable by all the values in the domain.

67

Axioms for Equality

- **reflexivity**: $x = x$.
- **symmetry**: $(x = y) \mid \neg(y = x)$.
- **transitivity**: $\neg(x = y) \mid \neg(y = z) \mid (x = z)$.
- **function monotonicity**:
  $\neg(x_i = y_i) \mid f(\ldots, x_i, \ldots) = f(\ldots, y_i, \ldots)$
- **predicate monotonicity**:
  $\neg(x_i = y_i) \mid \neg p(\ldots, x_i, \ldots) \mid p(\ldots, y_i, \ldots)$

68

68

Importance of Equality "="

- We can express "there are at least two elements x such that $A(x)$ holds" as $\exists x,y \neg(x = y) \wedge A(x) \wedge A(y)$.
- We can express "there are at most two elements x such that $A(x)$ holds" as

  $\forall x,y,z\ (A(x) \wedge A(y) \wedge A(z)) \rightarrow (x = y) \vee (y = z) \vee (x = z)$
- We can express "there are exactly two elements x such that $A(x)$ holds" as the conjunction of the above two formulas.

69

69

Rewrite Rules for Equality

- Without equality axions, we cannot prove f(f(a)) = a from f(a) = b and f(b) = a.
- Treating equations as rewrite rules, we can have an efficient reasoning tool.
- From f(a) $\rightarrow$ b and f(b) $\rightarrow$ a, we can rewrite f(f(a)) to a, thus showing f(f(a)) = a.
- Rewrite systems: a set of rewrite rules
- Two important properties:
  - **Termination**: no infinite sequence of rewritings
  - **Confluence**: no matter how to rewrite, the result is the same.

70

70

Inductive Theorem Proving with Equations

- Let E = { add(x, 0) = x, add(x, s(y)) = s(add(x, y)) }
- How to prove add(x, y) = add(y, x)?
- How to prove add(add(x, y), z) = add(x, add(y, z))?
- These are inductive theorems and need an induction proof.
- Automated methods exist for proving such properties.
- Details are given in Chapter 7.

71

71