Advanced topics on LOGIC and SEMANTICS

Conjunctive Normal Form

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

- $(A \vee B \vee \neg C)$ is a clause

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

- $(A \vee B \vee \neg C)$ is a clause, which is a disjunction of literals
- A, B, and ¬ C are literals

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

- $(A \vee B \vee \neg C)$ is a clause, which is a disjunction of literals
- A, B, and ¬ C are literals, each of which is a variable or the negation of a variable.

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

- $(A \vee B \vee \neg C)$ is a clause, which is a disjunction of literals
- A, B, and ¬ C are literals, each of which is a variable or the negation of a variable.
- Each clause is a requirement that must be satisfied and can be satisfied in multiple ways

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas:

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

  - $(A \vee B \vee \neg C)$ is a clause, which is a disjunction of literals
  - A, B, and $\neg$ C are literals, each of which is a variable or the negation of a variable.
  - Each clause is a requirement that must be satisfied and can be satisfied in multiple ways
  - Every sentence in propositional logic can be written in CNF

Converting to CNF

Converting to CNF

1. Eliminate arrows using definitions

Converting to CNF

1. Eliminate arrows using definitions
2. Drive in negations using De Morgan's Laws

$$\neg(\phi \vee \varphi) \equiv \neg\phi \wedge \neg\varphi$$
$$\neg(\phi \wedge \varphi) \equiv \neg\phi \vee \neg\varphi$$

Converting to CNF

1. Eliminate arrows using definitions
2. Drive in negations using De Morgan's Laws

$$\neg(\phi \vee \varphi) \equiv \neg\phi \wedge \neg\varphi$$
$$\neg(\phi \wedge \varphi) \equiv \neg\phi \vee \neg\varphi$$

3. Distribute **or** over **and**

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

Converting to CNF

1. Eliminate arrows using definitions
2. Drive in negations using De Morgan's Laws

$$\neg(\phi \vee \varphi) \equiv \neg\phi \wedge \neg\varphi$$
$$\neg(\phi \wedge \varphi) \equiv \neg\phi \vee \neg\varphi$$

3. Distribute **or** over **and**

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

4. Every sentence can be converted to CNF, but it may grow exponentially in size

CNF Conversion Example

$$(A \vee B) \rightarrow (C \rightarrow D)$$

CNF Conversion Example

$$(A \vee B) \rightarrow (C \rightarrow D)$$

1. Eliminate arrows

$$\neg(A \vee B) \vee (\neg C \vee D)$$

CNF Conversion Example

$$(A \vee B) \rightarrow (C \rightarrow D)$$

1. Eliminate arrows

$$\neg(A \vee B) \vee (\neg C \vee D)$$

2. Drive in negations

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

CNF Conversion Example

$$(A \vee B) \rightarrow (C \rightarrow D)$$

1. Eliminate arrows

$$\neg(A \vee B) \vee (\neg C \vee D)$$

2. Drive in negations

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

3. Distribute

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

Propositional Resolution

Propositional Resolution

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation:

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation:
  - Convert all sentences to CNF

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation:
  - Convert all sentences to CNF
  - Negate the desired conclusion (converted to CNF)

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation:
  - Convert all sentences to CNF
  - Negate the desired conclusion (converted to CNF)
  - Apply resolution rule until either
    - Derive false (a contradiction)
    - Can't apply any more

Propositional Resolution

- Resolution rule:

$$\frac{\alpha \vee \beta \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation:
  - Convert all sentences to CNF
  - Negate the desired conclusion (converted to CNF)
  - Apply resolution rule until either
    - Derive false (a contradiction)
    - Can't apply any more
- Resolution refutation is sound and complete
  - If we derive a contradiction, then the conclusion follows from the axioms
  - If we can't apply any more, then the conclusion cannot be proved from the axioms.

Propositional Resolution Example

Prove R

1	$P \vee Q$
2	$P \rightarrow R$
3	$Q \rightarrow R$

Step	Formula	Derivation

Propositional Resolution Example

Prove R

1	P v Q
2	$P \rightarrow R$
3	$Q \rightarrow R$

Step	Formula	Derivation
1	P v Q	Given

Propositional Resolution Example

Prove R

1	P v Q
2	$P \rightarrow R$
3	$Q \rightarrow R$

Step	Formula	Derivation
1	P v Q	Given
2	$\neg$ P v R	Given

Propositional Resolution Example

Prove R

1	P v Q
2	$P \rightarrow R$
3	$Q \rightarrow R$

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given

Propositional Resolution Example

Prove R

1	P v Q
2	$P \rightarrow R$
3	$Q \rightarrow R$

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4
7	¬ Q	3,4

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4
7	¬ Q	3,4
8	R	5,7

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4
7	¬ Q	3,4
8	R	5,7
9	•	4,8

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

false v R

¬ R v false

false v false

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4
7	¬ Q	3,4
8	R	5,7
9	•	4,8

Propositional Resolution Example

Prove R

1	P v Q
2	P → R
3	Q → R

false v R

¬ R v false

---

false v false

Step	Formula	Derivation
1	P v Q	Given
2	¬ P v R	Given
3	¬ Q v R	Given
4	¬ R	Negated conclusion
5	Q v R	1,2
6	¬ P	2,4
7	¬ Q	3,4
8	R	5,7
9	•	4,8

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion
4	•	1,2

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion
4	•	1,2

Note that $(P \wedge \neg P) \rightarrow Z$ is valid

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion
4	•	1,2

Note that $(P \wedge \neg P) \rightarrow Z$ is valid

Any conclusion follows from a contradiction – and so strict logic systems are very brittle.

Example Problem

Convert to CNF

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Example Problem

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Convert to CNF

- $\neg(\neg P \vee Q) \vee Q$
- $(P \wedge \neg Q) \vee Q$
- $(P \vee Q) \wedge (\neg Q \vee Q)$
- $(P \vee Q)$

Example Problem

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Convert to CNF

- $\neg(\neg P \vee Q) \vee Q$
- $(P \wedge \neg Q) \vee Q$
- $(P \vee Q) \wedge (\neg Q \vee Q)$
- $(P \vee Q)$

- $\neg(\neg P \vee P) \vee R$
- $(P \wedge \neg P) \vee R$
- $(P \vee R) \wedge (\neg P \vee R)$

Example Problem

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Convert to CNF

- $\neg(\neg P \vee Q) \vee Q$
- $(P \wedge \neg Q) \vee Q$
- $(P \vee Q) \wedge (\neg Q \vee Q)$
- $(P \vee Q)$

- $\neg(\neg P \vee P) \vee R$
- $(P \wedge \neg P) \vee R$
- $(P \vee R) \wedge (\neg P \vee R)$

- $\neg(\neg R \vee S) \vee \neg(\neg S \vee Q)$
- $(R \wedge \neg S) \vee (S \wedge \neg Q)$
- $(R \vee S) \wedge (\neg S \vee S) \wedge (R \vee \neg Q) \wedge (\neg S \vee \neg Q)$
- $(R \vee S) \wedge (R \vee \neg Q) \wedge (\neg S \vee \neg Q)$

Resolution Proof Example

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg (S \rightarrow Q)$

1	$P \vee Q$	
2	$P \vee R$	
3	$\neg P \vee R$	
4	$R \vee S$	
5	$R \vee \neg Q$	
6	$\neg S \vee \neg Q$	
7	$\neg R$	Neg

Resolution Proof Example

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg (S \rightarrow Q)$

1	$P \vee Q$	
2	$P \vee R$	
3	$\neg P \vee R$	
4	$R \vee S$	
5	$R \vee \neg Q$	
6	$\neg S \vee \neg Q$	
7	$\neg R$	Neg
8	S	4,7
9	$\neg Q$	6,8
10	P	1,9
11	R	3,10
12	•	7,11

Proof Strategies

Proof Strategies

- Unit preference: prefer a resolution step involving a unit clause (clause with one literal).
  - Produces a shorter clause – which is good since we are trying to produce a zero-length clause, that is, a contradiction.

Proof Strategies

- Unit preference: prefer a resolution step involving a unit clause (clause with one literal).
  - Produces a shorter clause – which is good since we are trying to produce a zero-length clause, that is, a contradiction.
- Set of support: Choose a resolution involving the negated goal or any clause derived from the negated goal.
  - We're trying to produce a contradiction that follows from the negated goal, so these are "relevant" clauses.
  - If a contradiction exists, one can find one using the set-of-support strategy.

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

First-Order Resolution -- Conversion to Clausal Form

First-Order Resolution

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

uppercase letters: constants

lowercase letters: variables

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

Syllogism:

All men are mortal

Socrates is a man

Socrates is mortal

uppercase letters: constants

lowercase letters: variables

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

Syllogism:

All men are mortal

Socrates is a man

Socrates is mortal

uppercase letters: constants

lowercase letters: variables

$$\frac{\forall x.\ \neg P(x) \vee Q(x) \qquad P(A)}{Q(A)}$$

Equivalent by definition of implication

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

Syllogism:

All men are mortal
Socrates is a man
Socrates is mortal

uppercase letters: constants

lowercase letters: variables

$$\frac{\forall x.\ \neg P(x) \vee Q(x) \qquad P(A)}{Q(A)}$$

Equivalent by definition of implication

$$\frac{\neg P(A) \vee Q(A) \qquad P(A)}{Q(A)}$$

Substitute A for x, still true

then

Propositional resolution

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

Syllogism:

All men are mortal

Socrates is a man

Socrates is mortal

uppercase letters: constants

lowercase letters: variables

$$\frac{\forall x.\ \neg P(x) \vee Q(x) \qquad P(A)}{Q(A)}$$

Equivalent by definition of implication

Two new things:

- converting FOL to clausal form
- resolution with variable substitution

$$\frac{\neg P(A) \vee Q(A) \qquad P(A)}{Q(A)}$$

Substitute A for x, still true

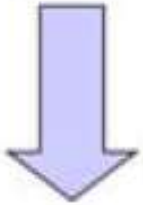
then

Propositional resolution

Clausal Form

- like CNF in outer structure
- no quantifiers

$$\forall x.\ \exists y.\ P(x) \rightarrow R(x, y)$$



$$\neg P(x) \vee R(x, F(x))$$

Converting to Clausal Form

Converting to Clausal Form

1. Eliminate arrows

$$\alpha \leftrightarrow \beta \Rightarrow (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

$$\alpha \rightarrow \beta \Rightarrow \neg\alpha \vee \beta$$

Converting to Clausal Form

1. Eliminate arrows

$$\alpha \leftrightarrow \beta \Rightarrow (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

$$\alpha \rightarrow \beta \Rightarrow \neg\alpha \vee \beta$$

2. Drive in negation

$$\neg(\alpha \vee \beta) \Rightarrow \neg\alpha \wedge \neg\beta$$

$$\neg(\alpha \wedge \beta) \Rightarrow \neg\alpha \vee \neg\beta$$

$$\neg\neg\alpha \Rightarrow \alpha$$

$$\neg\forall x.\ \alpha \Rightarrow \exists x.\ \neg\alpha$$

$$\neg\exists x.\ \alpha \Rightarrow \forall x.\ \neg\alpha$$

Converting to Clausal Form

1. Eliminate arrows

$$\alpha \leftrightarrow \beta \Rightarrow (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

$$\alpha \rightarrow \beta \Rightarrow \neg\alpha \vee \beta$$

2. Drive in negation

$$\neg(\alpha \vee \beta) \Rightarrow \neg\alpha \wedge \neg\beta$$

$$\neg(\alpha \wedge \beta) \Rightarrow \neg\alpha \vee \neg\beta$$

$$\neg\neg\alpha \Rightarrow \alpha$$

$$\neg\forall x.\ \alpha \Rightarrow \exists x.\ \neg\alpha$$

$$\neg\exists x.\ \alpha \Rightarrow \forall x.\ \neg\alpha$$

3. Rename variables apart

$$\forall x.\ \exists y.\ (\neg P(x) \vee \exists x.\ Q(x, y)) \Rightarrow$$

$$\forall x_1.\ \exists y_2.\ (\neg P(x_1) \vee \exists x_3.\ Q(x_3, y_2))$$

Skolemization

4. Skolemize

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\ P(x) \Rightarrow P(\text{Fred})$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\ P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\ R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\ P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\ R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

$$\exists x.\ P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\ P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\ R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

$$\exists x.\ P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$

$$\exists x.\ P(x) \wedge \exists x.\ Q(x) \Rightarrow P(\text{Frog}) \wedge Q(\text{Grog})$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\, P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\, R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

$$\exists x.\, P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$

$$\exists x.\, P(x) \wedge \exists x.\, Q(x) \Rightarrow P(\text{Frog}) \wedge Q(\text{Grog})$$

$$\exists y.\, \forall x.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Englebert})$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\, P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\, R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

$$\exists x.\, P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$

$$\exists x.\, P(x) \wedge \exists x.\, Q(x) \Rightarrow P(\text{Frog}) \wedge Q(\text{Grog})$$

$$\exists y.\, \forall x.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Englebert})$$

   - substitute new function of all universal vars in outer scopes

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\, P(x) \Rightarrow P(\text{Fred})$$

$$\exists x, y.\, R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$

$$\exists x.\, P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$

$$\exists x.\, P(x) \wedge \exists x.\, Q(x) \Rightarrow P(\text{Frog}) \wedge Q(\text{Grog})$$

$$\exists y.\, \forall x.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Englebert})$$

   - substitute new function of all universal vars in outer scopes

$$\forall x.\, \exists y.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Beloved}(x))$$

Skolemization

4. Skolemize
   - substitute new name for each existential var

$$\exists x.\, P(x) \Rightarrow P(\text{Fred})$$
$$\exists x, y.\, R(x, y) \Rightarrow R(\text{Thing1}, \text{Thing2})$$
$$\exists x.\, P(x) \wedge Q(x) \Rightarrow P(\text{Fleep}) \wedge Q(\text{Fleep})$$
$$\exists x.\, P(x) \wedge \exists x.\, Q(x) \Rightarrow P(\text{Frog}) \wedge Q(\text{Grog})$$
$$\exists y.\, \forall x.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Englebert})$$

   - substitute new function of all universal vars in outer scopes

$$\forall x.\, \exists y.\, \text{Loves}(x, y) \Rightarrow \forall x.\, \text{Loves}(x, \text{Beloved}(x))$$
$$\forall x.\, \exists y.\, \forall z.\, \exists w.\, P(x, y, z) \wedge R(y, z, w) \Rightarrow$$
$$P(x, F(x), z) \wedge R(F(x), z, G(x, z))$$

Convert to Clausal Form: Last Steps

5. Drop universal quantifiers

$$\forall x.\ \text{Loves}(x, \text{Beloved}(x)) \Rightarrow \text{Loves}(x, \text{Beloved}(x))$$

Convert to Clausal Form: Last Steps

5. Drop universal quantifiers

$$\forall x.\ \text{Loves}(x, \text{Beloved}(x)) \Rightarrow \text{Loves}(x, \text{Beloved}(x))$$

6. Distribute or over and; return clauses

$$P(z) \vee (Q(z,w) \wedge R(w,z)) \Rightarrow$$
$$\{\{P(z), Q(z,w)\}, \{P(z), R(w,z)\}\}$$

Convert to Clausal Form: Last Steps

5. Drop universal quantifiers

$$\forall x.\ \text{Loves}(x, \text{Beloved}(x)) \Rightarrow \text{Loves}(x, \text{Beloved}(x))$$

6. Distribute or over and; return clauses

$$P(z) \vee (Q(z,w) \wedge R(w,z)) \Rightarrow \{\{P(z), Q(z,w)\}, \{P(z), R(w,z)\}\}$$

7. Rename the variables in each clause

$$\{\{P(z), Q(z,w)\}, \{P(z), R(w,z)\}\} \Rightarrow \{\{P(z_1), Q(z_1,w_1)\}, \{P(z_2), R(w_2,z_2)\}\}$$

Example: Converting to clausal form

Example: Converting to clausal form

a. John owns a dog
$\exists x.\ D(x) \wedge O(J,x)$

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$(D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$ D(y) v $\neg$ O(x,y) v L(x)

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$(D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$ D(y) v $\neg$ O(x,y) v L(x)
$\neg$ D(y) v $\neg$ O(x,y) v L(x)

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$(D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$ D(y) v $\neg$ O(x,y) v L(x)
$\neg$ D(y) v $\neg$ O(x,y) v L(x)

c. Lovers-of-animals do not kill animals
$\forall$ x. L(x) $\rightarrow$ ($\forall$ y. A(y) $\rightarrow$ $\neg$ K(x,y))

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$(D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$ D(y) v $\neg$ O(x,y) v L(x)
$\neg$ D(y) v $\neg$ O(x,y) v L(x)

c. Lovers-of-animals do not kill animals
$\forall$ x. L(x) $\rightarrow$ ($\forall$ y. A(y) $\rightarrow$ $\neg$ K(x,y))
$\forall$ x. $\neg$ L(x) v ($\forall$ y. A(y) $\rightarrow$ $\neg$ K(x,y))
$\forall$ x. $\neg$ L(x) v ($\forall$ y. $\neg$ A(y) v $\neg$ K(x,y))

Example: Converting to clausal form

a. John owns a dog
$\exists$ x. D(x) $\wedge$ O(J,x)
D(Fido) $\wedge$ O(J, Fido)

b. Anyone who owns a dog is a lover-of-animals
$\forall$ x. ($\exists$ y. D(y) $\wedge$ O(x,y)) $\rightarrow$ L(x)
$\forall$ x. ($\neg\exists$ y. (D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$(D(y) $\wedge$ O(x,y)) v L(x)
$\forall$ x. $\forall$ y. $\neg$ D(y) v $\neg$ O(x,y) v L(x)
$\neg$ D(y) v $\neg$ O(x,y) v L(x)

c. Lovers-of-animals do not kill animals
$\forall$ x. L(x) $\rightarrow$ ($\forall$ y. A(y) $\rightarrow$ $\neg$ K(x,y))
$\forall$ x. $\neg$ L(x) v ($\forall$ y. A(y) $\rightarrow$ $\neg$ K(x,y))
$\forall$ x. $\neg$ L(x) v ($\forall$ y. $\neg$ A(y) v $\neg$ K(x,y))
$\neg$ L(x) v $\neg$ A(y) v $\neg$ K(x,y)

More converting to clausal form

d. Either Jack killed Tuna or curiosity killed Tuna
K(J,T) v K(C,T)

More converting to clausal form

d. Either Jack killed Tuna or curiosity killed Tuna
K(J,T) v K(C,T)

e. Tuna is a cat
C(T)

More converting to clausal form

d. Either Jack killed Tuna or curiosity killed Tuna
K(J,T) v K(C,T)

e. Tuna is a cat
C(T)

f. All cats are animals
¬ C(x) v A(x)

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Thank You

First-Order Resolution -- Substitution and Unification

First-Order Resolution

$$\frac{\forall x.\ P(x) \rightarrow Q(x) \qquad P(A)}{Q(A)}$$

Syllogism:

All men are mortal

Socrates is a man

Socrates is mortal

uppercase letters: constants

lowercase letters: variables

$$\frac{\forall x.\ \neg P(x) \vee Q(x) \qquad P(A)}{Q(A)}$$

Equivalent by definition of implication

$$\frac{\neg P(A) \vee Q(A) \qquad P(A)}{Q(A)}$$

Substitute A for x, still true

then

Propositional resolution

The key is finding the correct substitutions for the variables.

Substitutions

Substitutions

P(x, f(y), B) : an atomic sentence

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant
P(x, f(A), B)	{y/A}	

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant
P(x, f(A), B)	{y/A}	
P(g(z), f(A), B)	{x/g(z), y/A}	

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant
P(x, f(A), B)	{y/A}	
P(g(z), f(A), B)	{x/g(z), y/A}	
P(C, f(A), B)	{x/C, y/A}	Ground instance

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1, \ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant
P(x, f(A), B)	{y/A}	
P(g(z), f(A), B)	{x/g(z), y/A}	
P(C, f(A), B)	{x/C, y/A}	Ground instance

Substitutions

P(x, f(y), B) : an atomic sentence

Substitution instances	Substitution $\{v_1/t_1,\ldots, v_n/t_n\}$	Comment
P(z, f(w), B)	{x/z, y/w}	Alphabetic variant
P(x, f(A), B)	{y/A}	
P(g(z), f(A), B)	{x/g(z), y/A}	
P(C, f(A), B)	{x/C, y/A}	Ground instance

Applying a substitution:

P(x, f(y), B) {y/A} = P(x,f(A),B)

P(x, f(y), B) {y/A, x/y} = P(A, f(A), B)

Unification

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s
- Let ω_1 = x and ω_2 = y, the following are unifiers

s	ω_1 s	ω_2 s

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s
- Let ω_1 = x and ω_2 = y, the following are unifiers

s	ω_1 s	ω_2 s
{y/x}	x	x

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s
- Let ω_1 = x and ω_2 = y, the following are unifiers

s	ω_1 s	ω_2 s
{y/x}	x	x
{x/y}	y	y

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s
- Let ω_1 = x and ω_2 = y, the following are unifiers

s	ω_1 s	ω_2 s
{y/x}	x	x
{x/y}	y	y
{x/f(f(A)), y/f(f(A))}	f(f(A))	f(f(A))

Unification

- Expressions ω_1 and ω_2 are unifiable iff there exists a substitution s such that ω_1 s = ω_2 s
- Let ω_1 = x and ω_2 = y, the following are unifiers

s	ω_1 s	ω_2 s
{y/x}	x	x
{x/y}	y	y
{x/f(f(A)), y/f(f(A))}	f(f(A))	f(f(A))
{x/A, y/A}	A	A

Most General Unifier

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}
P(f(x), y, g(y))	P(f(x), z, g(x))	{y/x, z/x}

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}
P(f(x), y, g(y))	P(f(x), z, g(x))	{y/x, z/x}
P(x, B, B)	P(A, y, z)	{x/A, y/B, z/B}

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}
P(f(x), y, g(y))	P(f(x), z, g(x))	{y/x, z/x}
P(x, B, B)	P(A, y, z)	{x/A, y/B, z/B}
P(g(f(v)), g(u))	P(x, x)	{x/g(f(v)), u/f(v)}

Most General Unifier

g is a most general unifier of ω_1 and ω_2 iff for all unifiers s, there exists s′ such that ω_1 s = (ω_1 g) s′ and ω_2 s = (ω_2 g) s′

ω_1	ω_2	MGU
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}
P(f(x), y, g(y))	P(f(x), z, g(x))	{y/x, z/x}
P(x, B, B)	P(A, y, z)	{x/A, y/B, z/B}
P(g(f(v)), g(u))	P(x, x)	{x/g(f(v)), u/f(v)}
P(x, f(x))	P(x, x)	No MGU!

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
  else if x is a variable, return unify-var(x, y, s)
  else if y is a variable, return unify-var(y, x, s)
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
  else if x is a variable, return unify-var(x, y, s)
  else if y is a variable, return unify-var(y, x, s)
  else if x is a predicate or function application,
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
  else if x is a variable, return unify-var(x, y, s)
  else if y is a variable, return unify-var(y, x, s)
  else if x is a predicate or function application,
       if y has the same operator,
            return unify(args(x), args(y), s)
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
  else if x is a variable, return unify-var(x, y, s)
  else if y is a variable, return unify-var(y, x, s)
  else if x is a predicate or function application,
       if y has the same operator,
            return unify(args(x), args(y), s)
       else  return fail
```

Unification Algorithm

```
unify(Expr x, Expr y, Subst s){
  if s = fail, return fail
  else if x = y, return s
  else if x is a variable, return unify-var(x, y, s)
  else if y is a variable, return unify-var(y, x, s)
  else if x is a predicate or function application,
       if y has the same operator,
            return unify(args(x), args(y), s)
       else  return fail
  else                          ; x and y have to be lists
       return unify(rest(x), rest(y),
                    unify(first(x), first(y), s))
}
```

Unify-var subroutine

Substitute in for var and x as long as possible, then add new binding

```
unify-var(Variable var, Expr x, Subst s){
```

Unify-var subroutine

Substitute in for var and x as long as possible, then add new binding

```
unify-var(Variable var, Expr x, Subst s){
  if var is bound to val in s,
        return unify(val, x, s)
```

Unify-var subroutine

Substitute in for var and x as long as possible, then add new binding

```
unify-var(Variable var, Expr x, Subst s){
  if var is bound to val in s,
        return unify(val, x, s)
  else if x is bound to val in s,
        return unify-var(var, val, s)
```

Unify-var subroutine

Substitute in for var and x as long as possible, then add new binding

```
unify-var(Variable var, Expr x, Subst s){
  if var is bound to val in s,
       return unify(val, x, s)
  else if x is bound to val in s,
       return unify-var(var, val, s)
  else if var occurs anywhere in (x s), return fail
```

Unify-var subroutine

Substitute in for var and x as long as possible, then add new binding

```
unify-var(Variable var, Expr x, Subst s){
  if var is bound to val in s,
        return unify(val, x, s)
  else if x is bound to val in s,
        return unify-var(var, val, s)
  else if var occurs anywhere in (x s), return fail
  else return add({var/x}, s)
}
```

Some Examples

ω_1	ω_2	MGU
A(B, C)	A(x, y)	{x/B, y/C}
A(x, f(D,x))	A(E, f(D,y))	{x/E, y/E}
A(x, y)	A(f(C,y), z)	{x/f(C,y),y/z}
P(A, x, f(g(y)))	P(y, f(z), f(z))	{y/A,x/f(z),z/g(y)}
P(x, g(f(A)), f(x))	P(f(y), z, y)	none
P(x, f(y))	P(z, g(w))	none

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Thank You

First-Order Resolution -- Resolution with Variables

Resolution with Variables

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta} \quad \text{MGU}(\varphi, \psi) = \theta$$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta} \quad \text{MGU}(\varphi, \psi) = \theta$$

$$\frac{P(x) \vee Q(x, y) \quad \neg P(A) \vee R(B, z)}{}$$

$\theta = \{x/A\}$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta} \quad \text{MGU}(\varphi, \psi) = \theta$$

$$\frac{P(x) \vee Q(x, y) \quad \neg P(A) \vee R(B, z)}{(Q(x, y) \vee R(B, z))\theta}$$

$\theta = \{x/A\}$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta} \quad \text{MGU}(\varphi, \psi) = \theta$$

$$\frac{P(x) \vee Q(x, y) \quad \neg P(A) \vee R(B, z)}{(Q(x, y) \vee R(B, z))\theta}$$

$$Q(A, y) \vee R(B, z)$$

$\theta = \{x/A\}$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta} \quad \text{MGU}(\varphi, \psi) = \theta$$

$$\frac{P(x) \vee Q(x, y) \quad \neg P(A) \vee R(B, x)}{}$$

$$\frac{P(x) \vee Q(x, y) \quad \neg P(A) \vee R(B, z)}{(Q(x, y) \vee R(B, z))\theta}$$

$$Q(A, y) \vee R(B, z)$$

$\theta = \{x/A\}$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \text{MGU}(\varphi, \psi) = \theta \qquad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta}$$

$$\frac{\forall x, y.\ \ P(x) \vee Q(x, y) \qquad \forall x.\ \ \neg P(A) \vee R(B, x)}{}$$

$$\frac{\forall x, y.\ \ P(x) \vee Q(x, y) \qquad \forall z.\ \ \neg P(A) \vee R(B, z)}{(Q(x, y) \vee R(B, z))\theta}$$

$$Q(A, y) \vee R(B, z)$$

$\theta = \{x/A\}$

Resolution with Variables

$$\frac{\alpha \vee \varphi \quad \text{MGU}(\varphi, \psi) = \theta \qquad \neg\varphi \vee \beta}{(\alpha \vee \beta)\theta}$$

$$\frac{\forall x, y.\ \ P(x) \vee Q(x, y) \qquad \forall x.\ \ \neg P(A) \vee R(B, x)}{}$$

$$\frac{\forall x, y.\ \ P(x) \vee Q(x, y) \qquad \forall z.\ \ \neg P(A) \vee R(B, z)}{(Q(x, y) \vee R(B, z))\theta}$$

$$Q(A, y) \vee R(B, z)$$

$\theta = \{x/A\}$

$$\frac{P(x_1) \vee Q(x_1, y_1) \qquad \neg P(A) \vee R(B, x_2)}{(Q(x_1, y_1) \vee R(B, x_2))\theta}$$

$$Q(A, y_1) \vee R(B, x_2)$$

$\theta = \{x_1/A\}$

Curiosity Killed the Cat

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}
11	¬ L(J) v ¬ A(T)	4,9 {x/J, y/T}

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}
11	¬ L(J) v ¬ A(T)	4,9 {x/J, y/T}
12	¬ L(J)	10,11

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}
11	¬ L(J) v ¬ A(T)	4,9 {x/J, y/T}
12	¬ L(J)	10,11
13	¬ D(y) v ¬ O(J,y)	3,12 {x/J}

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}
11	¬ L(J) v ¬ A(T)	4,9 {x/J, y/T}
12	¬ L(J)	10,11
13	¬ D(y) v ¬ O(J,y)	3,12 {x/J}
14	¬ D(Fido)	13,2 {y/Fido}

Curiosity Killed the Cat

1	D(Fido)	a
2	O(J,Fido)	a
3	¬ D(y) v ¬ O(x,y) v L(x)	b
4	¬ L(x) v ¬ A(y) v ¬ K(x,y)	c
5	K(J,T) v K(C,T)	d
6	C(T)	e
7	¬ C(x) v A(x)	f
8	¬ K(C,T)	Neg
9	K(J,T)	5,8
10	A(T)	6,7 {x/T}
11	¬ L(J) v ¬ A(T)	4,9 {x/J, y/T}
12	¬ L(J)	10,11
13	¬ D(y) v ¬ O(J,y)	3,12 {x/J}
14	¬ D(Fido)	13,2 {y/Fido}
15	•	14,1

Proving validity

- How do we use resolution refutation to prove something is valid?

Proving validity

- How do we use resolution refutation to prove something is valid?
- Normally, we prove a sentence is entailed by the set of axioms

Proving validity

- How do we use resolution refutation to prove something is valid?
- Normally, we prove a sentence is entailed by the set of axioms
- Valid sentences are entailed by the empty set of sentences

Proving validity

- How do we use resolution refutation to prove something is valid?
- Normally, we prove a sentence is entailed by the set of axioms
- Valid sentences are entailed by the empty set of sentences
- To prove validity by refutation, negate the sentence and try to derive contradiction.

Proving validity: example

- Syllogism

$$\left(\forall x.\, P(x) \rightarrow Q(x)\right) \wedge P(A) \rightarrow Q(A)$$

Proving validity: example

- Syllogism

$$\left(\forall x.\, P(x) \rightarrow Q(x)\right) \wedge P(A) \rightarrow Q(A)$$

- Negate and convert to clausal form

$$\neg\left(\left(\forall x.\, P(x) \rightarrow Q(x)\right) \wedge P(A) \rightarrow Q(A)\right)$$

$$\neg\left(\neg\left(\forall x.\, \neg P(x) \vee Q(x)\right) \vee \neg P(A) \vee Q(A)\right]$$

$$\left(\forall x.\, \neg P(x) \vee Q(x)\right) \wedge P(A) \wedge \neg Q(A)$$

$$\left(\neg P(x) \vee Q(x)\right) \wedge P(A) \wedge \neg Q(A)$$

Proving validity: example

- Do proof

1.	$\neg P(x) \vee Q(x)$	
2.	$P(A)$	
3.	$\neg Q(A)$	
4.		
5.		

Proving validity: example

- Do proof

1.	$\neg P(x) \vee Q(x)$	
2.	$P(A)$	
3.	$\neg Q(A)$	
4.	$Q(A)$	1,2
5.	■	3,4

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Miscellaneous Logic Topics

Miscellaneous Logic Topics

- Factoring
- Green's trick
- Equality
- Completeness and decidability

Binary Resolution

- Binary resolution matches one literal from each clause
- Binary resolution isn't complete

Binary Resolution

- Binary resolution matches one literal from each clause
- Binary resolution isn't complete
- Can we get a contradiction from these clauses?

$$P(x) \vee P(y)$$

$$\neg P(v) \vee \neg P(w)$$

Binary Resolution

- Binary resolution matches one literal from each clause
- Binary resolution isn't complete
- Can we get a contradiction from these clauses?

$$P(x) \vee P(y)$$

$$\neg P(v) \vee \neg P(w)$$

- We should!

Binary Resolution

- Binary resolution matches one literal from each clause
- Binary resolution isn't complete
- Can we get a contradiction from these clauses?

$$P(x) \vee P(y)$$

$$\neg P(v) \vee \neg P(w)$$

- We should!
- But all we can get is $P(x) \vee \neg P(w)$

Binary Resolution

- Binary resolution matches one literal from each clause
- Binary resolution isn't complete
- Can we get a contradiction from these clauses?

$$P(x) \vee P(y)$$

$$\neg P(v) \vee \neg P(w)$$

- We should!
- But all we can get is $P(x) \vee \neg P(w)$
- And from there all we can do is get back to one of the original clauses

Factoring

- Generalized resolution lets you resolve away multiple literals at once

Factoring

- Generalized resolution lets you resolve away multiple literals at once
- It's simpler to introduce a new inference rule, called factoring

$$\frac{\alpha \vee \beta \vee \gamma}{(\alpha \vee \gamma)\theta} \quad \theta = \text{MGU}(\alpha, \beta)$$

Factoring

- Generalized resolution lets you resolve away multiple literals at once
- It's simpler to introduce a new inference rule, called factoring

$$\frac{\alpha \vee \beta \vee \gamma}{(\alpha \vee \gamma)\theta} \quad \theta = \text{MGU}(\alpha, \beta)$$

- Example

$$\underline{Q(y) \vee P(x, y) \vee P(v, A)}$$

Factoring

- Generalized resolution lets you resolve away multiple literals at once
- It's simpler to introduce a new inference rule, called factoring

$$\frac{\alpha \vee \beta \vee \gamma}{(\alpha \vee \gamma)\theta} \qquad \theta = \text{MGU}(\alpha, \beta)$$

- Example

$$\frac{Q(y) \vee P(x, y) \vee P(v, A)}{(Q(y) \vee P(x, y))\{x / v, y / A\}}$$

$$Q(A) \vee P(v, A)$$

Factoring

- Generalized resolution lets you resolve away multiple literals at once
- It's simpler to introduce a new inference rule, called factoring

$$\frac{\alpha \vee \beta \vee \gamma}{(\alpha \vee \gamma)\theta} \quad \theta = \text{MGU}(\alpha, \beta)$$

- Example

$$\frac{Q(y) \vee P(x, y) \vee P(v, A)}{(Q(y) \vee P(x, y))\{x / v, y / A\}}$$

$$Q(A) \vee P(v, A)$$

- Binary resolution plus factoring is complete

Green's Trick

- Use resolution to get answers to existential queries

Green's Trick

- Use resolution to get answers to existential queries

$\exists x.\ \text{Mortal}(x)$

1.	$\neg\text{Man}(x) \vee \text{Mortal}(x)$	
2.	$\text{Man}(\textit{Socrates})$	
3.	$\neg\text{Mortal}(x) \vee \text{Answer}(x)$	
4.		
5.		

Green's Trick

- Use resolution to get answers to existential queries

$\exists x.\ \text{Mortal}(x)$

1.	$\neg\text{Man}(x) \vee \text{Mortal}(x)$	
2.	$\text{Man}(\textit{Socrates})$	
3.	$\neg\text{Mortal}(x) \vee \text{Answer}(x)$	
4.	$\text{Mortal}(\textit{Socrates})$	1,2
5.	$\text{Answer}(\textit{Socrates})$	3,5

Equality

- Special predicate in syntax and semantics; need to add something to our proof system

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation
- Instead, we will axiomatize equality as an equivalence relation

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation
- Instead, we will axiomatize equality as an equivalence relation

$$\forall x.\mathrm{Eq}(x, x)$$

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation
- Instead, we will axiomatize equality as an equivalence relation

$$\forall x.\mathrm{Eq}(x,x)$$

$$\forall x,y.\mathrm{Eq}(x,y) \rightarrow \mathrm{Eq}(y,x)$$

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation
- Instead, we will axiomatize equality as an equivalence relation

$$\forall x.\mathrm{Eq}(x,x)$$

$$\forall x,y.\mathrm{Eq}(x,y) \rightarrow \mathrm{Eq}(y,x)$$

$$\forall x,y,z.\mathrm{Eq}(x,y) \wedge \mathrm{Eq}(y,z) \rightarrow \mathrm{Eq}(x,z)$$

Equality

- Special predicate in syntax and semantics; need to add something to our proof system
- Could add another special inference rule called paramodulation
- Instead, we will axiomatize equality as an equivalence relation

$$\forall x.\mathrm{Eq}(x,x)$$

$$\forall x,y.\mathrm{Eq}(x,y) \rightarrow \mathrm{Eq}(y,x)$$

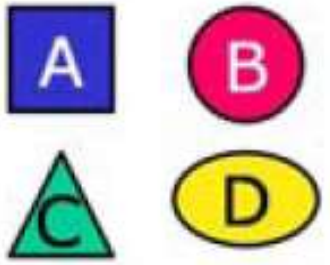
$$\forall x,y,z.\mathrm{Eq}(x,y) \wedge \mathrm{Eq}(y,z) \rightarrow \mathrm{Eq}(x,z)$$

- For every predicate, allow substitutions

$$\forall x,y.\mathrm{Eq}(x,y) \rightarrow (\mathrm{P}(x) \rightarrow \mathrm{P}(y))$$

Proof Example

- Let's go back to our old geometry domain and try to prove what the hat of A is



Proof Example

- Let's go back to our old geometry domain and try to prove what the hat of A is
- Axioms in FOL (plus equality axioms)

$\text{Above}(A, C)$

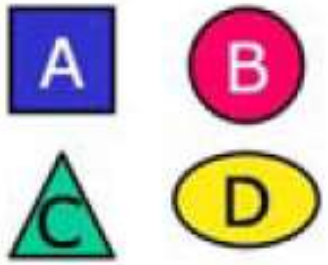
$\text{Above}(B, D)$

$\neg\exists x.\ \text{Above}(x, A)$

$\neg\exists x.\ \text{Above}(x, B)$

$\forall x, y.\ \text{Above}(x, y) \rightarrow \text{hat}(y) = x$

$\forall x.\ (\neg\exists y.\ \text{Above}(y, x)) \rightarrow \text{hat}(x) = x$



Proof Example

- Let's go back to our old geometry domain and try to prove what the hat of A is
- Axioms in FOL (plus equality axioms)

$\text{Above}(A, C)$

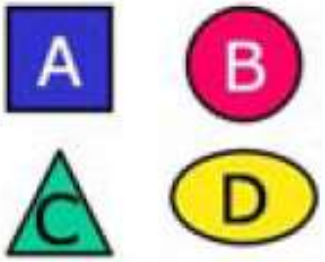
$\text{Above}(B, D)$

$\neg\exists x.\ \text{Above}(x, A)$

$\neg\exists x.\ \text{Above}(x, B)$

$\forall x, y.\ \text{Above}(x, y) \rightarrow \text{hat}(y) = x$

$\forall x.\ (\neg\exists y.\ \text{Above}(y, x)) \rightarrow \text{hat}(x) = x$



- Desired conclusion: $\exists x.\ \text{hat}(A) = x$
- Use Green's trick to get the binding of x

The Clauses

1.	Above(A, C)	
2.	Above(B, D)	
3.	~Above(x, A)	
4.	~Above(x, B)	
5.	~Above(x, y) v Eq(hat(y), x)	
6.	Above(sk(x), x) v Eq(hat(x), x)	
7.	Eq(x, x)	
8.	~Eq(x, y) v ~Eq(y, z) v Eq(x, z)	
9.	~Eq(x, y) v Eq(y, x)	
10.		
11.		
12.		

The Query

1.	Above(A, C)	
2.	Above(B, D)	
3.	~Above(x, A)	
4.	~Above(x, B)	
5.	~Above(x, y) v Eq(hat(y), x)	
6.	Above(sk(x), x) v Eq(hat(x), x)	
7.	Eq(x, x)	
8.	~Eq(x, y) v ~Eq(y, z) v Eq(x, z)	
9.	~Eq(x, y) v Eq(y, x)	
10.	~Eq(hat(A), x) v Answer(x)	

The Proof

1.	Above(A, C)	
2.	Above(B, D)	
3.	~Above(x, A)	
4.	~Above(x, B)	
5.	~Above(x, y) v Eq(hat(y), x)	
6.	Above(sk(x), x) v Eq(hat(x), x)	
7.	Eq(x, x)	
8.	~Eq(x, y) v ~Eq(y, z) v Eq(x, z)	
9.	~Eq(x, y) v Eq(y, x)	
10.	~Eq(hat(A), x) v Answer(x)	conclusion
11.	Above(sk(A), A) v Answer(A)	6, 10 {x/A}
12.	Answer(A)	11, 3 {x/sk(A)}

Hat of D

1.	Above(A, C)	
2.	Above(B, D)	
3.	~Above(x, A)	
4.	~Above(x, B)	
5.	~Above(x, y) v Eq(hat(y), x)	
6.	Above(sk(x), x) v Eq(hat(x), x)	
7.	Eq(x, x)	
8.	~Eq(x, y) v ~Eq(y, z) v Eq(x, z)	
9.	~Eq(x, y) v Eq(y, x)	
10.	~Eq(hat(D), x) v Answer(x)	conclusion
11.	~Above(x,D) v Answer(x)	5, 10 {x1/x}
12.	Answer(B)	11, 2 {x/B}

Who is Jane's Lover?

- Jane's lover drives a red car
- Fred is the only person who drives a red car
- Who is Jane's lover?

1.	Drives(lover(Jane))	
2.	~Drives(x) v Eq(x,Fred)	
3.	~Eq(lover(Jane),x) v Answer(x)	
4.	Eq(lover(Jane), Fred)	1,2 {x/lover(Jane)}
5.	Answer(Fred)	3,4 {x/Fred}

Completeness and Decidability

- Complete: If KB entails S, then we can prove S from KB

Completeness and Decidability

- Complete: If KB entails S, then we can prove S from KB
- Gödel's Completeness Theorem: *There exists a complete proof system for FOL*

Completeness and Decidability

- Complete: If KB entails S, then we can prove S from KB

- Gödel's Completeness Theorem: *There exists a complete proof system for FOL*
- Robinson's Completeness Theorem: *Resolution refutation is a complete proof system for FOL*

Completeness and Decidability

- Complete: If KB entails S, then we can prove S from KB

- Gödel's Completeness Theorem: *There exists a complete proof system for FOL*
- Robinson's Completeness Theorem: *Resolution refutation is a complete proof system for FOL*

- *FOL is semi-decidable:* if the desired conclusion follows from the premises then eventually resolution refutation will find a contradiction.
  - If there's a proof, we'll halt with it
  - If not, maybe we'll halt, maybe not

Adding Arithmetic

Adding Arithmetic

- Gödel's Incompleteness Theorem: *There is no consistent, complete proof system for FOL + Arithmetic.*
- Either there are sentences that are true, but not provable or there are sentences that are provable, but not true.

Adding Arithmetic

- Gödel's Incompleteness Theorem: *There is no consistent, complete proof system for FOL + Arithmetic.*
- Either there are sentences that are true, but not provable or there are sentences that are provable, but not true.

- Arithmetic gives you the ability to construct code-names for sentences within the logic.

  P = "P is not provable."

Adding Arithmetic

- Gödel's Incompleteness Theorem: *There is no consistent, complete proof system for FOL + Arithmetic.*
- Either there are sentences that are true, but not provable or there are sentences that are provable, but not true.

- Arithmetic gives you the ability to construct code-names for sentences within the logic.

  P = "P is not provable."

  - If P is true: it's not provable (incomplete)
  - If P is false: it's provable (inconsistent)

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

LOGIC
in the
Real World

Logic in the Real World

Logic in the Real World

- Encode information formally in web pages

Logic in the Real World

- Encode information formally in web pages
- Business rules

Logic in the Real World

- Encode information formally in web pages
- Business rules
- Airfare pricing

Airfare Pricing

- Ignore, for now, finding the best itinerary
- Given an itinerary, what's the least amount we can pay for it?
- Can't just add up prices for the flight legs; different prices for different flights in various combinations and circumstances

Fare Restrictions

- Passenger under 2 or over 65
- Passenger accompanying someone paying full fare
- Doesn't go through an expensive city
- No flights during rush hour
- Stay over Saturday night
- Layovers are legal
- Round-the-world itinerary that doesn't backtrack
- Regular two phase round-trip
- No flights on another airline
- This fare would not be cheaper than the standard price

Ontology

Ontology is the science of something and of nothing, of being and not-being, of the thing and the mode of the thing, of substance and accident.

Leibniz

Ontology

Ontology is the science of something and of nothing, of being and not-being, of the thing and the mode of the thing, of substance and accident.

Leibniz

The Role of Ontological Engineering in B2B Net Markets

ONTOLOGY.ORG
ENABLING VIRTUAL BUSINESS

Ontology

- What kinds of things are there in the world?
- What are their properties and relations?

Ontology is the science of something and of nothing, of being and not-being, of the thing and the mode of the thing, of substance and accident.

Leibniz

The Role of Ontological Engineering in B2B Net Markets

ONTOLOGY.ORG
ENABLING VIRTUAL BUSINESS

Airfare Domain Ontology

Airfare Domain Ontology

- passenger
- flight
- city
- airport
- terminal
- flight segment (list of flights, to be flown all in one "day")
- itinerary (a passenger and list of flight segments)

Airfare Domain Ontology

- passenger
- flight
- city
- airport
- terminal
- flight segment (list of flights, to be flown all in one "day")
- itinerary (a passenger and list of flight segments)
- list
- number

Representing Properties

Representing Properties

- Object P is red
  - Red(P)
  - Color(P, Red)
  - color(P) = Red
  - Property(P, Color, Red)

Representing Properties

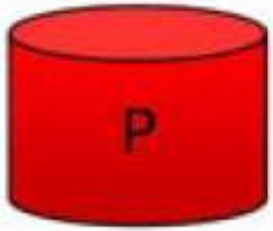
- Object P is red
  - Red(P)
  - Color(P, Red)
  - color(P) = Red
  - Property(P, Color, Red)

- All the blocks in stack S are the same color

$$\exists c.\ \forall b.\ \text{In}(b, S) \rightarrow \text{Color}(b, c)$$

Representing Properties

- Object P is red
  - Red(P)
  - Color(P, Red)
  - color(P) = Red
  - Property(P, Color, Red)



- All the blocks in stack S are the same color

$$\exists c.\ \forall b.\ \text{In}(b, S) \rightarrow \text{Color}(b, c)$$

- All the blocks in stack S have the same properties

$$\forall p.\ \exists v.\ \forall b.\ \text{In}(b, S) \rightarrow \text{Property}(b, p, v)$$

Basic Relations

Basic Relations

- Age(passenger, number)
- Nationality(passenger, country)
- Wheelchair(passenger)
- Origin(flight, airport)
- Destination(flight, airport)
- Departure_Time(flight, number)
- Arrival_Time(flight, number)
- Latitude(city, number)
- Longitude(city, number)
- In_Country(city, country)
- In_City(airport, city)
- Passenger(itinerary, passenger)
- Flight_Segments(itinerary, passenger, segments)
- Nil
- cons(object,list) => list

Basic Relations

- Age(passenger, number)
- Nationality(passenger, country)
- Wheelchair(passenger)
- Origin(flight, airport)
- Destination(flight, airport)
- Departure_Time(flight, number)
- Arrival_Time(flight, number)
- Latitude(city, number)
- Longitude(city, number)
- In_Country(city, country)
- In_City(airport, city)
- Passenger(itinerary, passenger)
- Flight_Segments(itinerary, passenger, segments)
- Nil
- cons(object,list) => list

Age(Fred, 47)
Nationality(Fred, US)
~Wheelchair(Fred)

Defined Relations

- Define complex relations in terms of basic ones
- Like using subroutines

$$\forall i.\ \mathrm{P}(i) \wedge \mathrm{Q}(i) \rightarrow \mathrm{Qualifies}\,37(i)$$

Defined Relations

- Define complex relations in terms of basic ones
- Like using subroutines

$$\forall i.\ \mathrm{P}(i) \wedge \mathrm{Q}(i) \rightarrow \mathrm{Qualifies\,37}(i)$$

- Implication rather than equivalence
  - easier to specify definitions in pieces

$$\forall i.\ \mathrm{R}(i) \wedge \mathrm{S}(i) \rightarrow \mathrm{Qualifies\,37}(i)$$

Defined Relations

- Define complex relations in terms of basic ones
- Like using subroutines

$$\forall i.\ P(i) \wedge Q(i) \rightarrow \text{Qualifies}\,37(i)$$

- Implication rather than equivalence
  - easier to specify definitions in pieces

$$\forall i.\ R(i) \wedge S(i) \rightarrow \text{Qualifies}\,37(i)$$

  - can't use the other direction

$$\text{Qualifies}\,37(i) \rightarrow ?$$

  - if you need it, write the equivalence

$$\forall i.\ (P(i) \wedge Q(i)) \vee (R(i) \wedge S(i)) \leftrightarrow \text{Qualifies}\,37(i)$$

Infant Fare

$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

Infant Fare

$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

$$\forall i\ (\ \exists p, a.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2) \rightarrow \text{InfantFare}(i)$$

Infant Fare

$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

$$\forall i\ (\ \exists p, a.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2) \rightarrow \text{InfantFare}(i)$$

- First form is typical of rule-based systems
- Second form (only!) can be made into an equivalence

Infant Fare

$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

$$\forall i\ (\ \exists p, a.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2) \rightarrow \text{InfantFare}(i)$$

- First form is typical of rule-based systems
- Second form (only!) can be made into an equivalence
- What about a < 2 ?

Infant Fare

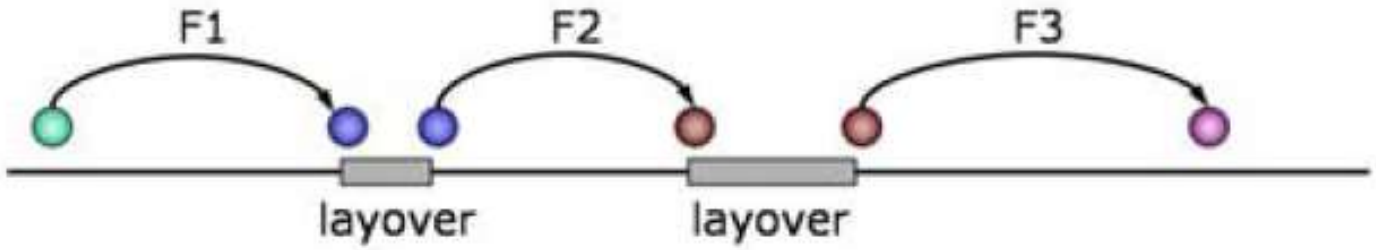
$$\forall i, a, p.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2 \rightarrow \text{InfantFare}(i)$$

$$\forall i\ (\ \exists p, a.\ \text{Passenger}(i, p) \wedge \text{Age}(p, a) \wedge a < 2) \rightarrow \text{InfantFare}(i)$$

- First form is typical of rule-based systems
- Second form (only!) can be made into an equivalence
- What about a < 2 ?
  - axiomatize arithmetic
  - build it in to theorem prover

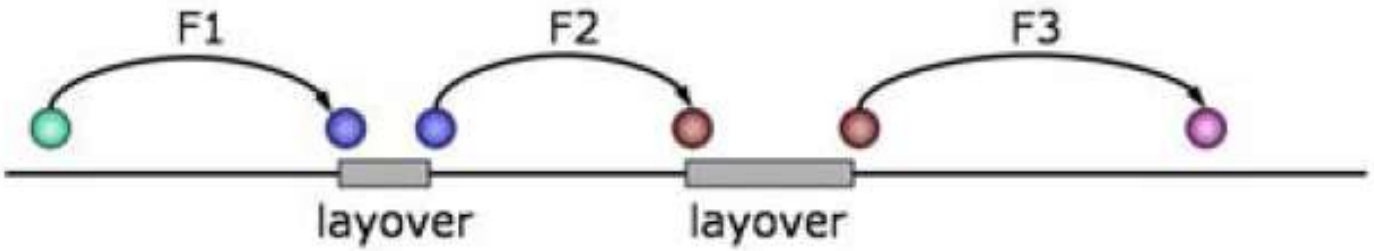
$$P(a) \vee (3 > 2) \vee (a > 1 + 2) \Rightarrow P(a) \vee (a > 3)$$

Well-Formed Segment



Well-Formed Segment

- Departure and arrival airports match up correctly
- Layovers aren't too short
- Layovers aren't too long



Lists in Logic

- Nil : constant
- cons : function

Lists in Logic

- Nil : constant
- cons : function
- cons(A, cons(B, Nil)) : list with two elements

Lists in Logic

- Nil : constant
- cons : function
- cons(A, cons(B, Nil)) : list with two elements
- $\forall x$. LengthOne(cons(x, Nil))

$$\forall l, x.\ l = \text{cons}(x, \text{Nil}) \rightarrow \text{LengthOne}(l)$$
$$\forall l.\ (\exists x.\ l = \text{cons}(x, \text{Nil})) \rightarrow \text{LengthOne}(l)$$

Well-Formed Segment: Base Case

Define recursively, going down the list of flights

Well-Formed Segment: Base Case

Define recursively, going down the list of flights

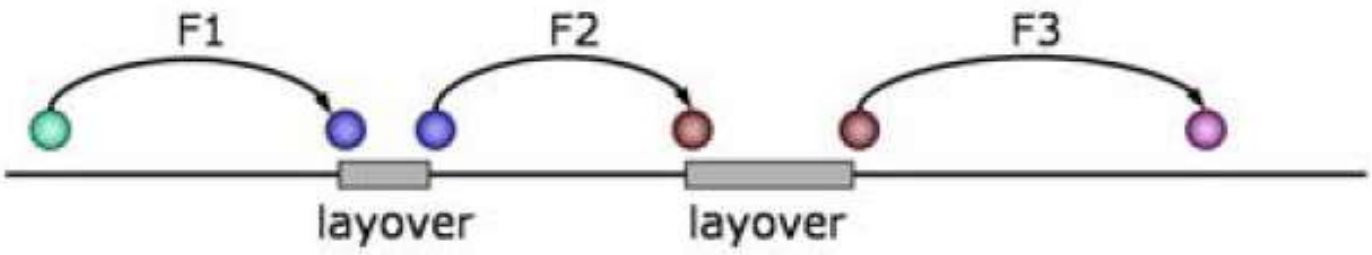
Any segment with 1 flight is well-formed

$\forall f$. WellFormed(cons(f, Nil))

Well-Formed Segment: Recursion

A flight segment with at least two flights is well-formed if

- first two flights are contiguous
- layover time between first two flights is legal
- rest of the flight segment is well-formed

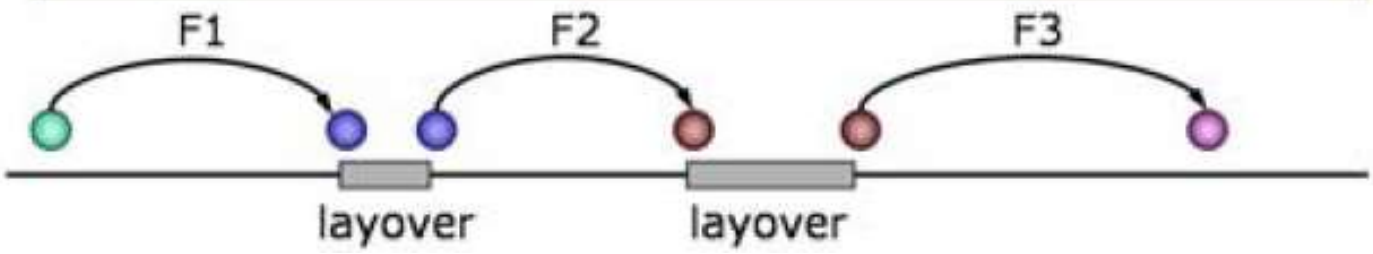


Well-Formed Segment: Recursion

A flight segment with at least two flights is well-formed if

- first two flights are contiguous
- layover time between first two flights is legal
- rest of the flight segment is well-formed

$$\forall f_1, f_2, r.\ \text{Contiguous}(f_1, f_2) \wedge \text{LegalLayover}(f_1, f_2) \wedge \text{WellFormed}(\text{cons}(f_2, r)) \rightarrow \text{WellFormed}(\text{cons}(f_1, \text{cons}(f_2, r)))$$



Helper Relations

Helper Relations

- Flights are contiguous if the arrival airport of the first is the same as the departure airport of the second

$$\forall f_1, f_2.\ (\exists c.\ \text{Destination}(f_1, c) \wedge \text{Origin}(f_2, c)) \rightarrow \text{Contiguous}(f_1, f_2)$$

Helper Relations

- Flights are contiguous if the arrival airport of the first is the same as the departure airport of the second

$$\forall f_1, f_2.\ (\exists c.\ \text{Destination}(f_1, c) \wedge \text{Origin}(f_2, c)) \rightarrow \text{Contiguous}(f_1, f_2)$$

- Layovers are legal if they're not too short and not too long

$$\forall f_1, f_2.\ \text{LayoverNotTooShort}(f_1, f_2) \wedge \text{LayoverNotTooLong}(f_1, f_2) \rightarrow \text{LayoverLegal}(f_1, f_2)$$

Not Too Short

- A layover is not too short if it's more than 30 minutes long

$$\forall f_1, f_2.\ (\exists t_1, t_2.\ \text{ArrivalTime}(f_1, t_1) \wedge \text{DepartureTime}(f_2, t_2) \wedge (t_2 - t_1 > 30)) \rightarrow \text{LayoverNotTooShort}(f_1, f_2)$$

Not Too Short

- A layover is not too short if it's more than 30 minutes long

$$\forall f_1, f_2.\ (\exists t_1, t_2.\ \text{ArrivalTime}(f_1, t_1) \wedge \text{DepartureTime}(f_2, t_2) \wedge (t_2 - t_1 > 30)) \rightarrow \text{LayoverNotTooShort}(f_1, f_2)$$

- These are like the rules the airlines use, but it could involve all of common sense to know how long to allow someone to change planes

Not Too Long

- A layover is not too long if it's less than three hours

$$\forall f_1, f_2.\ (\exists t_1, t_2.\ \text{ArrivalTime}(f_1, t_1) \wedge \text{DepartureTime}(f_2, t_2) \wedge (t_2 - t_1 < 180)) \rightarrow \text{LayoverNotTooLong}(f_1, f_2)$$

Not Too Long

- A layover is not too long if it's less than three hours

$$\forall f_1, f_2.\ (\exists t_1, t_2.\ \text{ArrivalTime}(f_1, t_1) \wedge \text{DepartureTime}(f_2, t_2) \wedge (t_2 - t_1 < 180)) \rightarrow \text{LayoverNotTooLong}(f_1, f_2)$$

- A layover is also not too long if there was no other way to make the next leg of your journey sooner

$$\forall f_1, f_2.\ (\exists o, d, t_2.\ \text{Origin}(f_2, o) \wedge \text{Destination}(f_2, d) \wedge \text{DepartureTime}(f_2, t_2) \wedge \neg\exists f_3, t_3.\ (\text{Origin}(f_3, o) \wedge \text{Destination}(f_3, d) \wedge \text{DepartureTime}(f_3, t_3) \wedge (t_3 < t_2) \wedge \text{LayoverNotTooShort}(f_1, f_3))) \rightarrow \text{LayoverNotTooLong}(f_1, f_2)$$

References

- Flach, P. (2012). Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press
- Russell S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach. 4th Edition: Pearson Education

  https://www.academia.edu/45126798/Artificial_Intelligence_A_Modern_Approach_4th_Edition_?auto=download
- Rich E., Knight K. & Nair, S. B. (2011). Artificial Intelligence 3rd Edition: Tata McGraw Hill
- Nilsson, N.J. (1998). Artificial Intelligence: a new synthesis. Morgan Kaufmann.
- Lucila Ohno-Machado, and Peter Szolovits. HST.947 Medical Artificial Intelligence. Spring 2005. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Thank You