Introduction to Artificial Intelligence
Lecture 9: Logical Inference

Nina Gierasimczuk

Outline

- Propositional Theorem Proving
- Resolution
- Horn Clauses and Definite Clauses
- Effective Propositional Model Checking

Outline

Truth-table Method for Inference

The most intuitive way to check validity of inference by brut-force truth-tables.

- *KB*: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well in the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

Truth-table Method for Inference

The most intuitive way to check validity of inference by brut-force truth-tables.

- *KB*: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well in the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

r	p	ℓ	$p \vee \ell$	$r \leftrightarrow p \vee \ell$	$\neg r$	$\neg p$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	1	1	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	0	0

Truth-table Method for Inference

The most intuitive way to check validity of inference by brut-force truth-tables.

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well in the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

r	p	ℓ	$p \vee \ell$	$r \leftrightarrow p \vee \ell$	$\neg r$	$\neg p$
0	0	0	0	**1**	**1**	**1**
0	0	1	1	0	**1**	1
0	1	0	1	0	**1**	0
0	1	1	1	0	**1**	0
1	0	0	0	0	0	1
1	0	1	1	**1**	0	1
1	1	0	1	**1**	0	0
1	1	1	1	**1**	0	0

Semantic vs Syntactic Approach

- (Semantic) model checking:

  enumerating models and showing it holds in all models.

- (Syntactic) theorem proving:

  applying rules of inference directly the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.

Semantic vs Syntactic Approach

- (Semantic) model checking:

  enumerating models and showing it holds in all models.
- (Syntactic) theorem proving:

  applying rules of inference directly the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.

Reason: If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

Some Crucial Concepts

- Logical equivalence

  two formulas φ and ψ are logically equivalent:

  if they are true in the same set of models, or

  if each of them entails the other: $\varphi \equiv \psi$ if and only if $\varphi \models \psi$ and $\psi \models \varphi$.

Some Crucial Concepts

- Logical equivalence

  two formulas φ and ψ are logically equivalent:

  if they are true in the same set of models, or

  if each of them entails the other: $\varphi \equiv \psi$ if and only if $\varphi \models \psi$ and $\psi \models \varphi$.

- Validity (tautology)

  φ is valid if it is true in all models.

  Every valid sentence is logically equivalent to $\top$.

Some Crucial Concepts

- Logical equivalence

  two formulas φ and ψ are logically equivalent:

  if they are true in the same set of models, or

  if each of them entails the other: $\varphi \equiv \psi$ if and only if $\varphi \models \psi$ and $\psi \models \varphi$.

- Validity (tautology)

  φ is valid if it is true in all models.

  Every valid sentence is logically equivalent to $\top$.

- Satisfiability

  φ is satisfiable if it is true in some model.

  The SAT problem, determining the satisfiability of sentences, was the first problem shown to be NP-complete.

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Validity and Satisfiability

$$\varphi \text{ is valid iff } \neg\varphi \text{ is unsatisfiable}$$

$$\varphi \text{ is satisfiable iff } \neg\varphi \text{ is not valid}$$

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$$\varphi \models \psi \textit{ if and only if } \varphi \to \psi \textit{ is valid.}$$

Validity and Satisfiability

$$\varphi \text{ is valid iff } \neg\varphi \text{ is unsatisfiable}$$

$$\varphi \text{ is satisfiable iff } \neg\varphi \text{ is not valid}$$

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$$\varphi \models \psi \textit{ if and only if } \varphi \to \psi \textit{ is valid.}$$

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$$\varphi \models \psi \textit{ if and only if } \varphi \to \psi \textit{ is valid.}$$

$\varphi \models \psi$ if and only if the sentence $(\varphi \wedge \neg\psi)$ is ?

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$$\varphi \models \psi \text{ if and only if } \varphi \to \psi \text{ is valid.}$$

$\varphi \models \psi$ if and only if the sentence $(\varphi \wedge \neg\psi)$ is ?

go to www.menti.com, enter digit code: 2376 8309

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$\varphi \models \psi$ *if and only if* $\varphi \to \psi$ *is valid.*

$\varphi \models \psi$ if and only if the sentence $(\varphi \wedge \neg\psi)$ is unsatisfiable

Validity and Satisfiability

φ is valid iff $\neg\varphi$ is unsatisfiable

φ is satisfiable iff $\neg\varphi$ is not valid

Theorem (Semantic Deduction Theorem)

For any sentences φ and ψ of propositional logic

$$\varphi \models \psi \text{ if and only if } \varphi \to \psi \text{ is valid.}$$

$\varphi \models \psi$ if and only if the sentence $(\varphi \wedge \neg\psi)$ is unsatisfiable

(reductio ad absurdum, proof by refutation, proof by contradiction)

Inference and Proofs

Inference rules are applied to derive a proof of a formula:

a chain of conclusions that leads to the formula.

Inference and Proofs

Inference rules are applied to derive a proof of a formula:

a chain of conclusions that leads to the formula.

Examples of inference rules

$$\frac{\varphi \rightarrow \psi, \ \varphi}{\psi} \qquad \text{(Modus Ponens)}$$

$$\frac{\varphi \wedge \psi}{\varphi} \qquad \text{(And-Elimination)}$$

Using Search Algorithms to Find Proofs

A friendly statement of the theorem proving problem as a search problem:

- Initial state: the initial knowledge base.
- Actions: the inference rules.
- Result: the result of an action is to add the conclusion to KB.
- Goal: the sentence we are trying to prove.

Monotonicity

In monotonic logics the set of entailed sentences can only increase as information is added to the knowledge base.

For any sentences φ and ψ, if $KB \models \varphi$ then $KB \cup \psi \models \varphi$.

Note: non-monotonic logics, which violate the monotonicity property, capture a common property of human reasoning: changing one's mind.

Outline

Resolution: one rule to rule them all

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \; m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k} \qquad \text{(Unit Resolution)}$$

where literals ℓ_i and m are complementary (i.e., one is negation of the other).

Resolution: one rule to rule them all

$$\frac{\ell_1 \vee \ldots \vee \ell_k,\ m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k} \qquad \text{(Unit Resolution)}$$

where literals ℓ_i and m are complementary (i.e., one is negation of the other).

$$\frac{\ell_1 \vee \ldots \vee \ell_k,\ m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

(Full Resolution)

where literals ℓ_i and m_j are complementary (i.e., one is negation of the other).

Resolution: one rule to rule them all

$$\frac{\ell_1 \vee \ldots \vee \ell_k,\ m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k} \qquad \text{(Unit Resolution)}$$

where literals ℓ_i and m are complementary (i.e., one is negation of the other).

$$\frac{\ell_1 \vee \ldots \vee \ell_k,\ m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

(Full Resolution)

where literals ℓ_i and m_j are complementary (i.e., one is negation of the other).

Practical comment: mind the Factoring (removing duplicates)!

E.g., if we resolve $(A \vee B)$ with $(A \vee \neg B)$, we obtain $(A \vee A)$.
Single A is enough!

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF?

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

1. Eliminate $\leftrightarrow$: $(r \rightarrow p \vee s) \wedge (p \vee s \rightarrow r)$.

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

1. Eliminate $\leftrightarrow$: $(r \rightarrow p \vee s) \wedge (p \vee s \rightarrow r)$.
2. Eliminate $\rightarrow$: $(\neg r \vee p \vee s) \wedge (\neg(p \vee s) \vee r)$.

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

1. Eliminate $\leftrightarrow$: $(r \rightarrow p \vee s) \wedge (p \vee s \rightarrow r)$.
2. Eliminate $\rightarrow$: $(\neg r \vee p \vee s) \wedge (\neg(p \vee s) \vee r)$.
3. Move $\neg$ inwards (De Morgan): $(\neg r \vee p \vee s) \wedge ((\neg p \wedge \neg s) \vee r)$.

Conjunctive normal form (CNF)

- Resolution applies to clauses (disjunctions of literals).
- So, KBs should then be coded as conjunctions of clauses (CNFs).
- Can any sentence of propositional logic be translated into CNF? YES!

Example

Robert will pass the exam if and only if he will be prepared or lucky.

$r \leftrightarrow p \vee s$

1. Eliminate $\leftrightarrow$: $(r \rightarrow p \vee s) \wedge (p \vee s \rightarrow r)$.
2. Eliminate $\rightarrow$: $(\neg r \vee p \vee s) \wedge (\neg(p \vee s) \vee r)$.
3. Move $\neg$ inwards (De Morgan): $(\neg r \vee p \vee s) \wedge ((\neg p \wedge \neg s) \vee r)$.
4. Distribute $\wedge$ over $\vee$: $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r)$.

A Resolution Algorithm

To show that $KB \models \varphi$, we show that $KB \wedge \neg\varphi$ is unsatisfiable.
We do this by proving a contradiction.

1. First, $KB \wedge \neg\varphi$ is converted into CNF.

A Resolution Algorithm

To show that $KB \models \varphi$, we show that $KB \wedge \neg\varphi$ is unsatisfiable.
We do this by proving a contradiction.

1. First, $KB \wedge \neg\varphi$ is converted into CNF.
2. Then, the resolution rule is applied to the resulting clauses.

A Resolution Algorithm

To show that $KB \models \varphi$, we show that $KB \wedge \neg\varphi$ is unsatisfiable.
We do this by proving a contradiction.

1. First, $KB \wedge \neg\varphi$ is converted into CNF.
2. Then, the resolution rule is applied to the resulting clauses.
3. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.

A Resolution Algorithm

To show that $KB \models \varphi$, we show that $KB \wedge \neg\varphi$ is unsatisfiable.
We do this by proving a contradiction.

1. First, $KB \wedge \neg\varphi$ is converted into CNF.
2. Then, the resolution rule is applied to the resulting clauses.
3. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
4. The process continues until one of two things happens:
   - A there are no new clauses that can be added, in which case KB does not entail φ; or,
   - B two clauses resolve to yield the empty clause, in which case KB entails φ.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

Example

- *KB*: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of *KB* as CNF.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $clauses := (\neg r \vee p \vee s), (\neg p \vee r), (\neg s \vee r), \neg r, p$.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $clauses{:=}\ (\neg r \vee p \vee s), (\neg p \vee r), (\neg s \vee r), \neg r, p$.
4. $clauses{:=}clauses \cup \{r\}$ from resolving $(\neg p \vee r)$ with p.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $\mathit{clauses} := (\neg r \vee p \vee s), (\neg p \vee r), (\neg s \vee r), \neg r, p$.
4. $\mathit{clauses} := \mathit{clauses} \cup \{r\}$ from resolving $(\neg p \vee r)$ with p.
5. $\mathit{clauses} := \mathit{clauses} \cup \{\perp\}$ from resolving r with $\neg r$.

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $clauses := (\neg r \vee p \vee s), (\neg p \vee r), (\neg s \vee r), \neg r, p$.
4. $clauses := clauses \cup \{r\}$ from resolving $(\neg p \vee r)$ with p.
5. $clauses := clauses \cup \{\perp\}$ from resolving r with $\neg r$.
6. Result: empty clause, so

Example

- KB: Robert does well in the exam if and only if he is prepared or lucky. Robert does not do well the exam.
- φ: Robert is not prepared.
- Question: $KB \models \varphi$?

1. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r$, formalisation of KB as CNF.
2. $(\neg r \vee p \vee s) \wedge (\neg p \vee r) \wedge (\neg s \vee r) \wedge \neg r \wedge p$, CNF for contradiction.
3. $clauses$:= $(\neg r \vee p \vee s)$, $(\neg p \vee r)$, $(\neg s \vee r)$, $\neg r$, p.
4. $clauses$:=$clauses \cup \{r\}$ from resolving $(\neg p \vee r)$ with p.
5. $clauses$:=$clauses \cup \{\bot\}$ from resolving r with $\neg r$.
6. Result: empty clause, so $KB \models \varphi$.

Resolution algorithm is good

1. Always terminates.
2. Is complete, by the ground resolution theorem: If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Outline

- Propositional Theorem Proving
- Resolution
- Horn Clauses and Definite Clauses
- Effective Propositional Model Checking

Special kinds of clauses

Sometimes it's enough if we restrict the language to special types of clauses:

Definition

A definite clause is a clause of literals of which exactly one is positive.
For example: $(\neg p \vee \neg s \vee r)$ is a definite clause, while $(p \vee s \vee \neg r)$ is not.

Special kinds of clauses

Sometimes it's enough if we restrict the language to special types of clauses:

Definition

A definite clause is a clause of literals of which exactly one is positive.
For example: $(\neg p \vee \neg s \vee r)$ is a definite clause, while $(p \vee s \vee \neg r)$ is not.

Definition

A Horn clause is a disjunction of literals of which at most one is positive.

Special kinds of clauses

Sometimes it's enough if we restrict the language to special types of clauses:

Definition

A definite clause is a clause of literals of which exactly one is positive.
For example: $(\neg p \lor \neg s \lor r)$ is a definite clause, while $(p \lor s \lor \neg r)$ is not.

Definition

A Horn clause is a disjunction of literals of which at most one is positive.

All definite clauses are Horn clauses, as are clauses with no positive literals (goal clauses).

Special kinds of clauses

Sometimes it's enough if we restrict the language to special types of clauses:

Definition

A definite clause is a clause of literals of which exactly one is positive.
For example: $(\neg p \lor \neg s \lor r)$ is a definite clause, while $(p \lor s \lor \neg r)$ is not.

Definition

A Horn clause is a disjunction of literals of which at most one is positive.

All definite clauses are Horn clauses, as are clauses with no positive literals (goal clauses).

Horn clauses are closed under resolution:
if you resolve two Horn clauses, you get back a Horn clause.

The Importance of Horn Clauses

Why are Horn clauses interesting?

The Importance of Horn Clauses

Why are Horn clauses interesting?

- Definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. E.g., $(\neg p \vee \neg s \vee r)$ is equivalent to $(p \wedge s) \rightarrow r$. The premise is called the body and the conclusion is called the head. A sentence consisting of a single positive literal, such as r, is called a fact $(\top \rightarrow r)$.

The Importance of Horn Clauses

Why are Horn clauses interesting?

- Definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. E.g., $(\neg p \vee \neg s \vee r)$ is equivalent to $(p \wedge s) \rightarrow r$. The premise is called the body and the conclusion is called the head. A sentence consisting of a single positive literal, such as r, is called a fact $(\top \rightarrow r)$.
- Inference with Horn clauses can be done by forward- and backward-chaining.

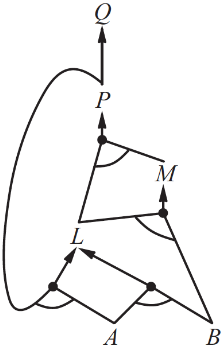
The Importance of Horn Clauses

Why are Horn clauses interesting?

- Definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. E.g., $(\neg p \vee \neg s \vee r)$ is equivalent to $(p \wedge s) \rightarrow r$. The premise is called the body and the conclusion is called the head. A sentence consisting of a single positive literal, such as r, is called a fact $(\top \rightarrow r)$.
- Inference with Horn clauses can be done by forward- and backward-chaining.
- Deciding entailment with Horn clauses is linear in the size of the knowledge base.

Forward- and Backward-chaining on AND-OR Graphs

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Outline

General Model-Checking Algorithms for Propositional Inference

- the algorithms checking satisfiability: the SAT problem
- testing entailment $\varphi \models \psi$, is done by testing unsatisfiability of $\varphi \land \neg\psi$
- two types: backtracking and local search

Davis-Putnam Algorithm (DPLL Algorithm)

DPLL is an improvement on the truth-table method that works on CNFs.

Davis-Putnam Algorithm (DPLL Algorithm)

DPLL is an improvement on the truth-table method that works on CNFs.

Improvements over TT:

1. Early termination, e.g., if $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless of B and C.

Davis-Putnam Algorithm (DPLL Algorithm)

DPLL is an improvement on the truth-table method that works on CNFs.

Improvements over TT:

1. Early termination, e.g., if $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless of B and C.
2. Pure symbol heuristic, e.g., in $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, the symbol A is pure. If a sentence has a model, then it has a model with the pure symbols assigned so as to make their literals true, because doing so can never make a clause false.

Davis-Putnam Algorithm (DPLL Algorithm)

DPLL is an improvement on the truth-table method that works on CNFs.

Improvements over TT:

1. Early termination, e.g., if $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless of B and C.
2. Pure symbol heuristic, e.g., in $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, the symbol A is pure. If a sentence has a model, then it has a model with the pure symbols assigned so as to make their literals true, because doing so can never make a clause false.
3. Unit clause heuristic, e.g., if the model contains $B = \top$, then $(\neg B \vee \neg C)$ simplifies to $\neg C$, which is a unit clause. Assigning one unit clause can create another unit clause, such 'cascade' of forced assignments is called unit propagation.

Davis-Putnam Algorithm (DPLL Algorithm)

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, { })

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
      DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))

WalkSat Algorithm

- Local search algorithms can be applied to SAT problem given the right evaluation function.

WalkSat Algorithm

- Local search algorithms can be applied to SAT problem given the right evaluation function.
- The goal is to find an assignment that satisfies every clause, so
- evaluation function counts the number of unsatisfied clauses.

WalkSat Algorithm

- Local search algorithms can be applied to SAT problem given the right evaluation function.
- The goal is to find an assignment that satisfies every clause, so
- evaluation function counts the number of unsatisfied clauses.
- We flip the truth value of one symbol at a time and escape local minima with randomness.

WalkSat Algorithm

- Local search algorithms can be applied to SAT problem given the right evaluation function.
- The goal is to find an assignment that satisfies every clause, so
- evaluation function counts the number of unsatisfied clauses.
- We flip the truth value of one symbol at a time and escape local minima with randomness.

WalkSat: On every iteration, the algorithm picks an unsatisfied clause and picks a symbol in the clause to flip. It chooses randomly between two ways to pick which symbol to flip:

1. a min-conflicts step that minimises the number of unsatisfied clauses in the new state, and
2. a random walk step that picks the symbol randomly.

WalkSat Algorithm

function WALKSAT(*clauses*, p, *max_flips*) **returns** a satisfying model or *failure*
inputs: *clauses*, a set of clauses in propositional logic
p, the probability of choosing to do a "random walk" move, typically around 0.5
max_flips, number of flips allowed before giving up

model ← a random assignment of *true*/*false* to the symbols in *clauses*
for $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability p flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
return *failure*

End of Lecture 9