Fundamentals of Artificial Intelligence
Chapter 07: **Logical Agents**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
http://disi.unitn.it/rseba/DIDATTICA/fai_2020/

Teaching assistant: Mauro Dragoni – dragoni@fbk.eu
http://www.maurodragoni.com/teaching/fai/

M.S. Course "Artificial Intelligence Systems", academic year 2020-2021

Last update: Tuesday 8th December, 2020, 13:07

Outline

1. Propositional Logic
2. Propositional Reasoning
   - Resolution
   - DPLL
   - Modern CDCL SAT Solvers
   - Reasoning with Horn Formulas
   - Local Search
3. Knowledge-Based Agents
4. Agents Based on Propositional Reasoning

Outline

Propositional Logic (aka Boolean Logic)

Basic Definitions and Notation

- Propositional formula (aka Boolean formula or sentence)
  - $\top, \bot$ are formulas
  - a propositional atom $A_1, A_2, A_3, ...$ is a formula;
  - if φ_1 and φ_2 are formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$, $\varphi_1 \oplus \varphi_2$ are formulas.
- *Atoms*(φ): the set $\{A_1, ..., A_N\}$ of atoms occurring in φ.
- Literal: a propositional atom A_i (positive literal) or its negation $\neg A_i$ (negative literal)
  - Notation: if $l := \neg A_i$, then $\neg l := A_i$
- Clause: a disjunction of literals $\bigvee_j l_j$ (e.g., $(A_1 \vee \neg A_2 \vee A_3 \vee ...)$)
- Cube: a conjunction of literals $\bigwedge_j l_j$ (e.g., $(A_1 \wedge \neg A_2 \wedge A_3 \wedge ...)$)

Semantics of Boolean operators

Truth Table

α	β	$\neg\alpha$	$\alpha\wedge\beta$	$\alpha\vee\beta$	$\alpha\rightarrow\beta$	$\alpha\leftarrow\beta$	$\alpha\leftrightarrow\beta$	$\alpha\oplus\beta$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$

Note

- $\wedge$, $\vee$, $\leftrightarrow$ and $\oplus$ are commutative:
  $$(\alpha\wedge\beta) \iff (\beta\wedge\alpha)$$
  $$(\alpha\vee\beta) \iff (\beta\vee\alpha)$$
  $$(\alpha\leftrightarrow\beta) \iff (\beta\leftrightarrow\alpha)$$
  $$(\alpha\oplus\beta) \iff (\beta\oplus\alpha)$$
- $\wedge$ and $\vee$ are associative:
  $$((\alpha\wedge\beta)\wedge\gamma) \iff (\alpha\wedge(\beta\wedge\gamma)) \iff (\alpha\wedge\beta\wedge\gamma)$$
  $$((\alpha\vee\beta)\vee\gamma) \iff (\alpha\vee(\beta\vee\gamma)) \iff (\alpha\vee\beta\vee\gamma)$$

Semantics of Boolean operators

Truth Table

α	β	$\neg\alpha$	$\alpha\wedge\beta$	$\alpha\vee\beta$	$\alpha\rightarrow\beta$	$\alpha\leftarrow\beta$	$\alpha\leftrightarrow\beta$	$\alpha\oplus\beta$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$

Note

- $\wedge$, $\vee$, $\leftrightarrow$ and $\oplus$ are commutative:
  $$(\alpha \wedge \beta) \iff (\beta \wedge \alpha)$$
  $$(\alpha \vee \beta) \iff (\beta \vee \alpha)$$
  $$(\alpha \leftrightarrow \beta) \iff (\beta \leftrightarrow \alpha)$$
  $$(\alpha \oplus \beta) \iff (\beta \oplus \alpha)$$
- $\wedge$ and $\vee$ are associative:
  $$((\alpha \wedge \beta) \wedge \gamma) \iff (\alpha \wedge (\beta \wedge \gamma)) \iff (\alpha \wedge \beta \wedge \gamma)$$
  $$((\alpha \vee \beta) \vee \gamma) \iff (\alpha \vee (\beta \vee \gamma)) \iff (\alpha \vee \beta \vee \gamma)$$

Remark: Semantics of Implication "$\rightarrow$" (aka "$\Rightarrow$", "$\supset$")

The semantics of Implication "$\alpha \rightarrow \beta$" may be counter-intuitive

$\alpha \rightarrow \beta$: "the antecedent (aka premise) α implies the consequent (aka conclusion) β" (aka "if α holds, then β holds" (but not vice versa))

- does not require causation or relevance between α and β
  - ex: "5 is odd implies Tokyo is the capital of Japan" is true in p.l. (under standard interpretation of "5", "odd", "Tokyo", "Japan")
  - relation between antecedent & consequent: they are both true
- is true whenever its antecedent is false
  - ex: "5 is even implies Sam is smart" is true (regardless the smartness of Sam)
  - ex: "5 is even implies Tokyo is in Italy" is true (!)
  - relation between antecedent & consequent: the former is false
- does not require temporal precedence of α wrt. β
  - ex: "the grass is wet implies it must have rained" is true (the consequent precedes temporally the antecedent)

Remark: Semantics of Implication "$\rightarrow$" (aka "$\Rightarrow$", "$\supset$")

The semantics of Implication "$\alpha \rightarrow \beta$" may be counter-intuitive

$\alpha \rightarrow \beta$: "the antecedent (aka premise) α implies the consequent (aka conclusion) β" (aka "if α holds, then β holds" (but not vice versa))

- does not require causation or relevance between α and β
  - ex: "5 is odd implies Tokyo is the capital of Japan" is true in p.l. (under standard interpretation of "5", "odd", "Tokyo", "Japan")
  - relation between antecedent & consequent: they are both true
- is true whenever its antecedent is false
  - ex: "5 is even implies Sam is smart" is true (regardless the smartness of Sam)
  - ex: "5 is even implies Tokyo is in Italy" is true (!)
  - relation between antecedent & consequent: the former is false
- does not require temporal precedence of α wrt. β
  - ex: "the grass is wet implies it must have rained" is true (the consequent precedes temporally the antecedent)

Remark: Semantics of Implication "$\rightarrow$" (aka "$\Rightarrow$", "$\supset$")

The semantics of Implication "$\alpha \rightarrow \beta$" may be counter-intuitive

$\alpha \rightarrow \beta$: "the antecedent (aka premise) α implies the consequent (aka conclusion) β" (aka "if α holds, then β holds" (but not vice versa))

- does not require causation or relevance between α and β
  - ex: "5 is odd implies Tokyo is the capital of Japan" is true in p.l. (under standard interpretation of "5", "odd", "Tokyo", "Japan")
  - relation between antecedent & consequent: they are both true
- is true whenever its antecedent is false
  - ex: "5 is even implies Sam is smart" is true (regardless the smartness of Sam)
  - ex: "5 is even implies Tokyo is in Italy" is true (!)
  - relation between antecedent & consequent: the former is false
- does not require temporal precedence of α wrt. β
  - ex: "the grass is wet implies it must have rained" is true (the consequent precedes temporally the antecedent)

Remark: Semantics of Implication "$\rightarrow$" (aka "$\Rightarrow$", "$\supset$")

The semantics of Implication "$\alpha \rightarrow \beta$" may be counter-intuitive

$\alpha \rightarrow \beta$: "the antecedent (aka premise) α implies the consequent (aka conclusion) β" (aka "if α holds, then β holds" (but not vice versa))

- does not require causation or relevance between α and β
  - ex: "5 is odd implies Tokyo is the capital of Japan" is true in p.l. (under standard interpretation of "5", "odd", "Tokyo", "Japan")
  - relation between antecedent & consequent: they are both true
- is true whenever its antecedent is false
  - ex: "5 is even implies Sam is smart" is true (regardless the smartness of Sam)
  - ex: "5 is even implies Tokyo is in Italy" is true (!)
  - relation between antecedent & consequent: the former is false
- does not require temporal precedence of α wrt. β
  - ex: "the grass is wet implies it must have rained" is true (the consequent precedes temporally the antecedent)

Syntactic Properties of Boolean Operators

$$
\begin{array}{rcl}
\neg\neg\alpha & \iff & \alpha \\
(\alpha \vee \beta) & \iff & \neg(\neg\alpha \wedge \neg\beta) \\
\neg(\alpha \vee \beta) & \iff & (\neg\alpha \wedge \neg\beta) \\
(\alpha \wedge \beta) & \iff & \neg(\neg\alpha \vee \neg\beta) \\
\neg(\alpha \wedge \beta) & \iff & (\neg\alpha \vee \neg\beta) \\
(\alpha \rightarrow \beta) & \iff & (\neg\alpha \vee \beta) \\
(\alpha \rightarrow \beta) & \iff & (\neg\beta \wedge \neg\alpha) \\
\neg(\alpha \rightarrow \beta) & \iff & (\alpha \wedge \neg\beta) \\
(\alpha \leftarrow \beta) & \iff & (\alpha \vee \neg\beta) \\
\neg(\alpha \leftarrow \beta) & \iff & (\neg\alpha \wedge \beta) \\
(\alpha \leftrightarrow \beta) & \iff & ((\alpha \rightarrow \beta) \wedge (\alpha \leftarrow \beta)) \\
 & \iff & ((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)) \\
\neg(\alpha \leftrightarrow \beta) & \iff & (\neg\alpha \leftrightarrow \beta) \\
 & \iff & (\alpha \leftrightarrow \neg\beta) \\
 & \iff & ((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)) \\
(\alpha \oplus \beta) & \iff & \neg(\alpha \leftrightarrow \beta)
\end{array}
$$

Boolean logic can be expressed in terms of $\{\neg, \wedge\}$ (or $\{\neg, \vee\}$) only!

Syntactic Properties of Boolean Operators

$$
\begin{array}{rcl}
\neg\neg\alpha & \iff & \alpha \\
(\alpha \vee \beta) & \iff & \neg(\neg\alpha \wedge \neg\beta) \\
\neg(\alpha \vee \beta) & \iff & (\neg\alpha \wedge \neg\beta) \\
(\alpha \wedge \beta) & \iff & \neg(\neg\alpha \vee \neg\beta) \\
\neg(\alpha \wedge \beta) & \iff & (\neg\alpha \vee \neg\beta) \\
(\alpha \rightarrow \beta) & \iff & (\neg\alpha \vee \beta) \\
(\alpha \rightarrow \beta) & \iff & (\neg\beta \wedge \neg\alpha) \\
\neg(\alpha \rightarrow \beta) & \iff & (\alpha \wedge \neg\beta) \\
(\alpha \leftarrow \beta) & \iff & (\alpha \vee \neg\beta) \\
\neg(\alpha \leftarrow \beta) & \iff & (\neg\alpha \wedge \beta) \\
(\alpha \leftrightarrow \beta) & \iff & ((\alpha \rightarrow \beta) \wedge (\alpha \leftarrow \beta)) \\
 & \iff & ((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)) \\
\neg(\alpha \leftrightarrow \beta) & \iff & (\neg\alpha \leftrightarrow \beta) \\
 & \iff & (\alpha \leftrightarrow \neg\beta) \\
 & \iff & ((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)) \\
(\alpha \oplus \beta) & \iff & \neg(\alpha \leftrightarrow \beta)
\end{array}
$$

Boolean logic can be expressed in terms of $\{\neg, \wedge\}$ (or $\{\neg, \vee\}$) only!

Tree & DAG Representations of Formulas

- Formulas can be represented either as trees or as DAGS (Directed Acyclic Graphs)
- DAG representation can be up to exponentially smaller
  - in particular, when $\leftrightarrow$'s are involved

$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$

Tree & DAG Representations of Formulas

- Formulas can be represented either as trees or as DAGS (Directed Acyclic Graphs)
- DAG representation can be up to exponentially smaller
  - in particular, when $\leftrightarrow$'s are involved

$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$
$$\Downarrow$$
$$(((A_1 \leftrightarrow A_2) \rightarrow (A_3 \leftrightarrow A_4)) \wedge$$
$$((A_3 \leftrightarrow A_4) \rightarrow (A_1 \leftrightarrow A_2)))$$

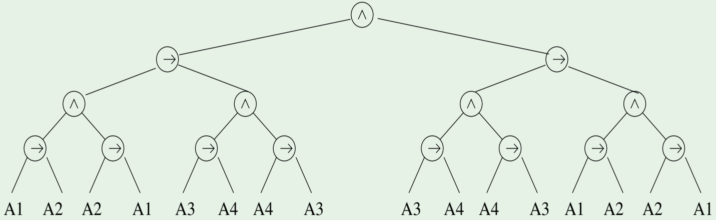
Tree & DAG Representations of Formulas

- Formulas can be represented either as trees or as DAGS (Directed Acyclic Graphs)
- DAG representation can be up to exponentially smaller
  - in particular, when $\leftrightarrow$'s are involved

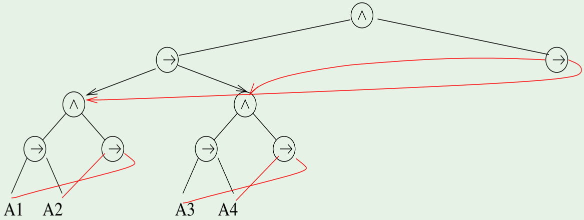
$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$
$$\Downarrow$$
$$(((A_1 \leftrightarrow A_2) \rightarrow (A_3 \leftrightarrow A_4)) \wedge$$
$$((A_3 \leftrightarrow A_4) \rightarrow (A_1 \leftrightarrow A_2)))$$
$$\Downarrow$$
$$(((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3))) \wedge$$
$$(((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow (((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1))))$$

Tree & DAG Representations of Formulas: Example

$$(((A_1 \to A_2) \wedge (A_2 \to A_1)) \to ((A_3 \to A_4) \wedge (A_4 \to A_3))) \wedge$$
$$(((A_3 \to A_4) \wedge (A_4 \to A_3)) \to (((A_1 \to A_2) \wedge (A_2 \to A_1))))$$



Tree Representation



DAG Representation

Basic Definitions and Notation [cont.]

- Total truth assignment μ for φ:
  $\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$.
  - represents a possible world or a possible state of the world
- Partial Truth assignment μ for φ:
  $\mu : \mathcal{A} \longmapsto \{\top, \bot\}, \mathcal{A} \subset Atoms(\varphi)$.
  - represents 2^k total assignments, k is $\#$ unassigned variables
- Notation: set and formula representations of an assignment
  - μ can be represented as a set of literals:
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies \{A_1, \neg A_2\}$
  - μ can be represented as a formula (cube):
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies (A_1 \wedge \neg A_2)$

Basic Definitions and Notation [cont.]

- A total truth assignment μ satisfies φ (μ is a model of φ, $\mu \models \varphi$):

$$\begin{array}{l}
\mu \models A_i \iff \mu(A_i) = \top \\
\mu \models \neg\varphi \iff \textit{not } \mu \models \varphi \\
\mu \models \alpha \wedge \beta \iff \mu \models \alpha \textit{ and } \mu \models \beta \\
\mu \models \alpha \vee \beta \iff \mu \models \alpha \textit{ or } \mu \models \beta \\
\mu \models \alpha \rightarrow \beta \iff \textit{ if } \mu \models \alpha, \textit{ then } \mu \models \beta \\
\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \textit{ iff } \mu \models \beta \\
\mu \models \alpha \oplus \beta \iff \mu \models \alpha \textit{ iff not } \mu \models \beta
\end{array}$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (the set of models of φ)
- A partial truth assignment μ satisfies φ iff all its total extensions satisfy φ
  - (Ex: $\{A_1\} \models (A_1 \vee A_2)$) because $\{A_1, A_2\} \models (A_1 \vee A_2)$ and $\{A_1, \neg A_2\} \models (A_1 \vee A_2)$)
- φ is satisfiable iff $\mu \models \varphi$ for some μ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all μs, $\mu \models \alpha \Longrightarrow \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mu \models \varphi$ forall μs (i.e., $\mu \in M(\varphi)$ forall μs)

Basic Definitions and Notation [cont.]

- A total truth assignment μ satisfies φ (μ is a model of φ, $\mu \models \varphi$):

$$
\begin{array}{l}
\mu \models A_i \iff \mu(A_i) = \top \\
\mu \models \neg\varphi \iff \textit{not } \mu \models \varphi \\
\mu \models \alpha \wedge \beta \iff \mu \models \alpha \textit{ and } \mu \models \beta \\
\mu \models \alpha \vee \beta \iff \mu \models \alpha \textit{ or } \mu \models \beta \\
\mu \models \alpha \rightarrow \beta \iff \textit{ if } \mu \models \alpha, \textit{ then } \mu \models \beta \\
\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \textit{ iff } \mu \models \beta \\
\mu \models \alpha \oplus \beta \iff \mu \models \alpha \textit{ iff not } \mu \models \beta
\end{array}
$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (the set of models of φ)
- A partial truth assignment μ satisfies φ iff all its total extensions satisfy φ
  - (Ex: $\{A_1\} \models (A_1 \vee A_2)$) because $\{A_1, A_2\} \models (A_1 \vee A_2)$ and $\{A_1, \neg A_2\} \models (A_1 \vee A_2)$)
- φ is satisfiable iff $\mu \models \varphi$ for some μ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all μs, $\mu \models \alpha \Longrightarrow \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mu \models \varphi$ forall μs (i.e., $\mu \in M(\varphi)$ forall μs)

Basic Definitions and Notation [cont.]

- A total truth assignment μ satisfies φ (μ is a model of φ, $\mu \models \varphi$):

$$\mu \models A_i \iff \mu(A_i) = \top$$
$$\mu \models \neg\varphi \iff \textit{not } \mu \models \varphi$$
$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \textit{ and } \mu \models \beta$$
$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \textit{ or } \mu \models \beta$$
$$\mu \models \alpha \rightarrow \beta \iff \textit{ if } \mu \models \alpha, \textit{ then } \mu \models \beta$$
$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \textit{ iff } \mu \models \beta$$
$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \textit{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (the set of models of φ)
- A partial truth assignment μ satisfies φ iff all its total extensions satisfy φ
  - (Ex: $\{A_1\} \models (A_1 \vee A_2)$) because $\{A_1, A_2\} \models (A_1 \vee A_2)$ and $\{A_1, \neg A_2\} \models (A_1 \vee A_2)$)
- φ is satisfiable iff $\mu \models \varphi$ for some μ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all μs, $\mu \models \alpha \Longrightarrow \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mu \models \varphi$ forall μs (i.e., $\mu \in M(\varphi)$ forall μs)

Basic Definitions and Notation [cont.]

- A total truth assignment μ satisfies φ (μ is a model of φ, $\mu \models \varphi$):

$$\begin{array}{l}
\mu \models A_i \iff \mu(A_i) = \top \\
\mu \models \neg\varphi \iff \textit{not } \mu \models \varphi \\
\mu \models \alpha \wedge \beta \iff \mu \models \alpha \textit{ and } \mu \models \beta \\
\mu \models \alpha \vee \beta \iff \mu \models \alpha \textit{ or } \mu \models \beta \\
\mu \models \alpha \rightarrow \beta \iff \textit{ if } \mu \models \alpha, \textit{ then } \mu \models \beta \\
\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \textit{ iff } \mu \models \beta \\
\mu \models \alpha \oplus \beta \iff \mu \models \alpha \textit{ iff not } \mu \models \beta
\end{array}$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (the set of models of φ)
- A partial truth assignment μ satisfies φ iff all its total extensions satisfy φ
  - (Ex: $\{A_1\} \models (A_1 \vee A_2)$) because $\{A_1, A_2\} \models (A_1 \vee A_2)$ and $\{A_1, \neg A_2\} \models (A_1 \vee A_2)$)
- φ is satisfiable iff $\mu \models \varphi$ for some μ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all μs, $\mu \models \alpha \Longrightarrow \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mu \models \varphi$ forall μs (i.e., $\mu \in M(\varphi)$ forall μs)

Basic Definitions and Notation [cont.]

- A total truth assignment μ satisfies φ (μ is a model of φ, $\mu \models \varphi$):

$$\begin{array}{l}
\mu \models A_i \iff \mu(A_i) = \top \\
\mu \models \neg\varphi \iff \textit{not } \mu \models \varphi \\
\mu \models \alpha \wedge \beta \iff \mu \models \alpha \textit{ and } \mu \models \beta \\
\mu \models \alpha \vee \beta \iff \mu \models \alpha \textit{ or } \mu \models \beta \\
\mu \models \alpha \rightarrow \beta \iff \textit{ if } \mu \models \alpha, \textit{ then } \mu \models \beta \\
\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \textit{ iff } \mu \models \beta \\
\mu \models \alpha \oplus \beta \iff \mu \models \alpha \textit{ iff not } \mu \models \beta
\end{array}$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (the set of models of φ)
- A partial truth assignment μ satisfies φ iff all its total extensions satisfy φ
  - (Ex: $\{A_1\} \models (A_1 \vee A_2)$) because $\{A_1, A_2\} \models (A_1 \vee A_2)$ and $\{A_1, \neg A_2\} \models (A_1 \vee A_2)$)
- φ is satisfiable iff $\mu \models \varphi$ for some μ (i.e. $M(\varphi) \neq \emptyset$)
- α entails β ($\alpha \models \beta$) iff, for all μs, $\mu \models \alpha \Longrightarrow \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is valid ($\models \varphi$) iff $\mu \models \varphi$ forall μs (i.e., $\mu \in M(\varphi)$ forall μs)

Properties & Results

Property

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

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid ($\models \alpha \rightarrow \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

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

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \to \beta$ is valid ($\models \alpha \to \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

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

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid ($\models \alpha \rightarrow \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Properties & Results

Property

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

Deduction Theorem

$\alpha \models \beta$ iff $\alpha \to \beta$ is valid ($\models \alpha \to \beta$)

Corollary

$\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Validity and entailment checking can be straightforwardly reduced to (un)satisfiability checking!

Complexity

- For N variables, there are up to 2^N truth assignments to be checked.
- The problem of deciding the satisfiability of a propositional formula is NP-complete

$\Longrightarrow$ The most important logical problems (validity, inference, entailment, equivalence, ...) can be straightforwardly reduced to (un)satisfiability, and are thus (co)NP-complete.

$$\Downarrow$$

No existing worst-case-polynomial algorithm.

Conjunctive Normal Form (CNF)

- φ is in Conjunctive normal form iff it is a conjunction of disjunctions of literals:

$$\bigwedge_{i=1}^{L} \bigvee_{j_i=1}^{K_i} l_{j_i}$$

- the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called clauses
- Easier to handle: list of lists of literals.
  $\Longrightarrow$ no reasoning on the recursive structure of the formula

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \to \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\mathrm{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$ $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \rightarrow \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||CNF(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \to \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||CNF(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \rightarrow \beta \quad \Longrightarrow \quad \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \quad \Longrightarrow \quad (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \quad \Longrightarrow \quad \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \quad \Longrightarrow \quad \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \quad \Longrightarrow \quad \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\text{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \rightarrow \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\mathrm{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee \ldots \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee \ldots \vee l_{N1}) \wedge \ldots \wedge (l_{12} \vee l_{22} \vee \ldots \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \rightarrow \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\text{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$ $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \rightarrow \beta \quad \Longrightarrow \quad \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \quad \Longrightarrow \quad (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \quad \Longrightarrow \quad \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \quad \Longrightarrow \quad \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \quad \Longrightarrow \quad \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\text{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  (i) expanding implications and equivalences:
  $$\alpha \to \beta \implies \neg\alpha \vee \beta$$
  $$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
  (ii) pushing down negations recursively:
  $$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
  $$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
  $$\neg\neg\alpha \implies \alpha$$
  (iii) applying recursively the DeMorgan's Rule:
  $$(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
- Resulting formula worst-case exponential:
  - ex: $||\mathrm{CNF}(\bigvee_{i=1}^{N}(l_{i1} \wedge l_{i2})|| =$
    $||(l_{11} \vee l_{21} \vee ... \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee ... \vee l_{N1}) \wedge ... \wedge (l_{12} \vee l_{22} \vee ... \vee l_{N2})|| = 2^N$
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ is equivalent to φ: $M(CNF(\varphi)) = M(\varphi)$
- Rarely used in practice.

Labeling CNF conversion $CNF_{label}(\varphi)$

Labeling CNF conversion $CNF_{label}(\varphi)$ (aka Tseitin's conversion)

- Every φ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:

$$\begin{array}{lll} \varphi & \Longrightarrow & \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j)) \\ \varphi & \Longrightarrow & \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j)) \\ \varphi & \Longrightarrow & \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j)) \end{array}$$

  l_i, l_j being literals and B being a "new" variable.
- Worst-case linear!
- $Atoms(CNF_{label}(\varphi)) \supseteq Atoms(\varphi)$
- $CNF_{label}(\varphi)$ is equi-satisfiable w.r.t. φ:
  $M(CNF(\varphi)) \neq \emptyset$ iff $M(\varphi) \neq \emptyset$
- Much more used than classic conversion in practice.

Labeling CNF conversion $CNF_{label}(\varphi)$

Labeling CNF conversion $CNF_{label}(\varphi)$ (aka Tseitin's conversion)

- Every φ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:

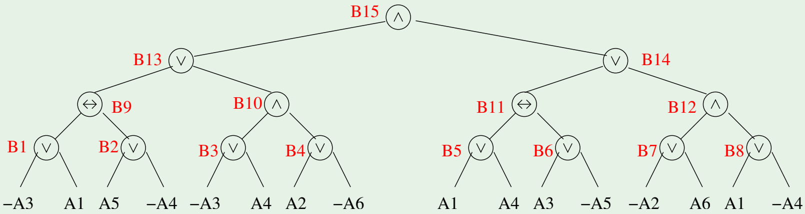
$$\begin{array}{lll} \varphi & \Longrightarrow & \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j)) \\ \varphi & \Longrightarrow & \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j)) \\ \varphi & \Longrightarrow & \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j)) \end{array}$$

  l_i, l_j being literals and B being a "new" variable.
- Worst-case linear!
- $Atoms(CNF_{label}(\varphi)) \supseteq Atoms(\varphi)$
- $CNF_{label}(\varphi)$ is equi-satisfiable w.r.t. φ:
  $M(CNF(\varphi)) \neq \emptyset$ iff $M(\varphi) \neq \emptyset$
- Much more used than classic conversion in practice.

Labeling CNF conversion $CNF_{label}(\varphi)$ (cont.)

$CNF(B \leftrightarrow (l_i \vee l_j))$	$\Longleftrightarrow$	$(\neg B \vee l_i \vee l_j) \wedge$
		$(B \vee \neg l_i) \wedge$
		$(B \vee \neg l_j)$
$CNF(B \leftrightarrow (l_i \wedge l_j))$	$\Longleftrightarrow$	$(\neg B \vee l_i) \wedge$
		$(\neg B \vee l_j) \wedge$
		$(B \vee \neg l_i \neg l_j)$
$CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$	$\Longleftrightarrow$	$(\neg B \vee \neg l_i \vee l_j) \wedge$
		$(\neg B \vee l_i \vee \neg l_j) \wedge$
		$(B \vee l_i \vee l_j) \wedge$
		$(B \vee \neg l_i \vee \neg l_j)$

Labeling CNF Conversion CNF_{label} – Example



$$
\begin{array}{l}
CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1))\wedge \\
\ldots\wedge \\
CNF(B_8 \leftrightarrow (A_1 \vee \neg A_4))\wedge \\
CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2))\wedge \\
\\
\ldots\wedge \\
CNF(B_{12} \leftrightarrow (B_7 \wedge B_8))\wedge \\
CNF(B_{13} \leftrightarrow (B_9 \vee B_{10}))\wedge \\
CNF(B_{14} \leftrightarrow (B_{11} \vee B_{12}))\wedge \\
CNF(B_{15} \leftrightarrow (B_{13} \wedge B_{14}))\wedge \\
B_{15}
\end{array}
=
\begin{array}{l}
(\neg B_1 \vee \neg A_3 \vee A_1) \wedge (B_1 \vee A_3) \wedge (B_1 \vee \neg A_1)\wedge \\
\ldots\wedge \\
(\neg B_8 \vee A_1 \vee \neg A_4) \wedge (B_8 \vee \neg A_1) \wedge (B_8 \vee A_4)\wedge \\
(\neg B_9 \vee \neg B_1 \vee B_2) \wedge (\neg B_9 \vee B_1 \vee \neg B_2)\wedge \\
(B_9 \vee B_1 \vee B_2) \wedge (B_9 \vee \neg B_1 \vee \neg B_2)\wedge \\
\ldots\wedge \\
(B_{12} \vee \neg B_7 \vee \neg B_8) \wedge (\neg B_{12} \vee B_7) \wedge (\neg B_{12} \vee B_8)\wedge \\
(\neg B_{13} \vee B_9 \vee B_{10}) \wedge (B_{13} \vee \neg B_9) \wedge (B_{13} \vee \neg B_{10})\wedge \\
(\neg B_{14} \vee B_{11} \vee B_{12}) \wedge (B_{14} \vee \neg B_{11}) \wedge (B_{14} \vee \neg B_{12})\wedge \\
(B_{15} \vee \neg B_{13} \vee \neg B_{14}) \wedge (\neg B_{15} \vee B_{13}) \wedge (\neg B_{15} \vee B_{14})\wedge \\
B_{15}
\end{array}
$$

Outline

1. Propositional Logic
2. Propositional Reasoning
   - Resolution
   - DPLL
   - Modern CDCL SAT Solvers
   - Reasoning with Horn Formulas
   - Local Search
3. Knowledge-Based Agents
4. Agents Based on Propositional Reasoning

Propositional Reasoning: Generalities

- Automated Reasoning in Propositional Logic fundamental task
  - AI, formal verification, circuit synthesis, operational research,....
- Important in AI: $KB \models \alpha$: entail fact α from knowledge base KR (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
  - typically $KB >> \alpha$
  - sometimes KB set of variable implications $(A_1 \wedge ... \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
  - $KR \models \alpha \Longrightarrow$ SAT($KR \wedge \neg\alpha$)=false
  - input formula CNF-ized and fed to a SAT solver
- Current SAT solvers dramatically efficient:
  - handle industrial problems with $10^6 - 10^7$ variables & clauses!
  - used as backend engines in a variety of systems (not only AI)

Propositional Reasoning: Generalities

- Automated Reasoning in Propositional Logic fundamental task
  - AI, formal verification, circuit synthesis, operational research,....
- Important in AI: $KB \models \alpha$: entail fact α from knowledge base KR (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
  - typically $KB >> \alpha$
  - sometimes KB set of variable implications $(A_1 \wedge ... \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
  - $KR \models \alpha \Longrightarrow$ SAT($KR \wedge \neg\alpha$)=false
  - input formula CNF-ized and fed to a SAT solver
- Current SAT solvers dramatically efficient:
  - handle industrial problems with $10^6 - 10^7$ variables & clauses!
  - used as backend engines in a variety of systems (not only AI)

Propositional Reasoning: Generalities

- Automated Reasoning in Propositional Logic fundamental task
  - AI, formal verification, circuit synthesis, operational research,....
- Important in AI: $KB \models \alpha$: entail fact α from knowledge base KR (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
  - typically $KB >> \alpha$
  - sometimes KB set of variable implications $(A_1 \wedge ... \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
  - $KR \models \alpha \Longrightarrow \text{SAT}(KR \wedge \neg\alpha)\text{=false}$
  - input formula CNF-ized and fed to a SAT solver
- Current SAT solvers dramatically efficient:
  - handle industrial problems with $10^6 - 10^7$ variables & clauses!
  - used as backend engines in a variety of systems (not only AI)

Propositional Reasoning: Generalities

- Automated Reasoning in Propositional Logic fundamental task
  - AI, formal verification, circuit synthesis, operational research,....
- Important in AI: $KB \models \alpha$: entail fact α from knowledge base KR (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
  - typically $KB >> \alpha$
  - sometimes KB set of variable implications $(A_1 \wedge ... \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
  - $KR \models \alpha \Longrightarrow$ SAT($KR \wedge \neg\alpha$)=false
  - input formula CNF-ized and fed to a SAT solver
- Current SAT solvers dramatically efficient:
  - handle industrial problems with $10^6 - 10^7$ variables & clauses!
  - used as backend engines in a variety of systems (not only AI)

Propositional Reasoning: Generalities

- Automated Reasoning in Propositional Logic fundamental task
  - AI, formal verification, circuit synthesis, operational research,....
- Important in AI: $KB \models \alpha$: entail fact α from knowledge base KR (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
  - typically $KB >> \alpha$
  - sometimes KB set of variable implications $(A_1 \wedge ... \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
  - $KR \models \alpha \Longrightarrow$ SAT($KR \wedge \neg\alpha$)=false
  - input formula CNF-ized and fed to a SAT solver
- Current SAT solvers dramatically efficient:
  - handle industrial problems with $10^6 - 10^7$ variables & clauses!
  - used as backend engines in a variety of systems (not only AI)

Outline

The Resolution Rule

- **Resolution**: deduction of a new clause from a pair of clauses with exactly one incompatible variable (resolvent):

$$\frac{(\overbrace{l_1 \vee \ldots \vee l_k}^{common} \vee \overbrace{l}^{resolvent} \vee \overbrace{l'_{k+1} \vee \ldots \vee l'_m}^{C'}) \qquad (\overbrace{l_1 \vee \ldots \vee l_k}^{common} \vee \overbrace{\neg l}^{resolvent} \vee \overbrace{l''_{k+1} \vee \ldots \vee l''_n}^{C''})}{(\underbrace{l_1 \vee \ldots \vee l_k}_{common} \vee \underbrace{l'_{k+1} \vee \ldots \vee l'_m}_{C'} \vee \underbrace{l''_{k+1} \vee \ldots \vee l''_n}_{C''})}$$

- Ex: $\frac{(A \vee B \vee C \vee D \vee E) \qquad (A \vee B \vee \neg C \vee F)}{(A \vee B \vee D \vee E \vee F)}$

- Note: many standard inference rules subcases of resolution: (recall that $\alpha \rightarrow \beta \Longleftrightarrow \neg\alpha \vee \beta$)

$$\frac{A \rightarrow B \quad B \rightarrow C}{A \rightarrow C}\ (trans.) \qquad \frac{A \quad A \rightarrow B}{B}\ (m.\ ponens) \qquad \frac{\neg B \quad A \rightarrow B}{\neg A}\ (m.\ tollens)$$

The Resolution Rule

- Resolution: deduction of a new clause from a pair of clauses with exactly one incompatible variable (resolvent):

$$\frac{(\overbrace{l_1 \vee \ldots \vee l_k}^{common} \vee \overbrace{l}^{resolvent} \vee \overbrace{l'_{k+1} \vee \ldots \vee l'_m}^{C'}) \qquad (\overbrace{l_1 \vee \ldots \vee l_k}^{common} \vee \overbrace{\neg l}^{resolvent} \vee \overbrace{l''_{k+1} \vee \ldots \vee l''_n}^{C''})}{(\underbrace{l_1 \vee \ldots \vee l_k}_{common} \vee \underbrace{l'_{k+1} \vee \ldots \vee l'_m}_{C'} \vee \underbrace{l''_{k+1} \vee \ldots \vee l''_n}_{C''})}$$

- Ex: $\dfrac{(A \vee B \vee C \vee D \vee E) \qquad (A \vee B \vee \neg C \vee F)}{(A \vee B \vee D \vee E \vee F)}$

- Note: many standard inference rules subcases of resolution: (recall that $\alpha \to \beta \iff \neg\alpha \vee \beta$)

$$\frac{A \to B \quad B \to C}{A \to C}\,(trans.) \qquad \frac{A \quad A \to B}{B}\,(m.\ ponens) \qquad \frac{\neg B \quad A \to B}{\neg A}\,(m.\ tollens)$$

The Resolution Rule

- Resolution: deduction of a new clause from a pair of clauses with exactly one incompatible variable (resolvent):

$$\frac{(\overbrace{l_1 \vee ... \vee l_k}^{common} \vee \overbrace{l}^{resolvent} \vee \overbrace{l'_{k+1} \vee ... \vee l'_m}^{C'}) \qquad (\overbrace{l_1 \vee ... \vee l_k}^{common} \vee \overbrace{\neg l}^{resolvent} \vee \overbrace{l''_{k+1} \vee ... \vee l''_n}^{C''})}{(\underbrace{l_1 \vee ... \vee l_k}_{common} \vee \underbrace{l'_{k+1} \vee ... \vee l'_m}_{C'} \vee \underbrace{l''_{k+1} \vee ... \vee l''_n}_{C''})}$$

- Ex: $\dfrac{(A \vee B \vee C \vee D \vee E) \qquad (A \vee B \vee \neg C \vee F)}{(A \vee B \vee D \vee E \vee F)}$
- Note: many standard inference rules subcases of resolution: (recall that $\alpha \rightarrow \beta \iff \neg\alpha \vee \beta$)

$$\frac{A \rightarrow B \quad B \rightarrow C}{A \rightarrow C}\ (trans.) \qquad \frac{A \quad A \rightarrow B}{B}\ (m.\ ponens) \qquad \frac{\neg B \quad A \rightarrow B}{\neg A}\ (m.\ tollens)$$

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \land \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Basic Propositional Inference: Resolution

- Assume input formula in CNF
  - if not, apply Tseitin CNF-ization first

$\Longrightarrow$ φ is represented as a set of clauses

- Search for a refutation of φ (is φ unsatisfiable?)
  - recall: $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ unsatisfiable
- Basic idea: apply iteratively the resolution rule to pairs of clauses with a conflicting literal, producing novel clauses, until either
  - a false clause is generated, or
  - the resolution rule is no more applicable
- Correct: if returns an empty clause, then φ unsat ($\alpha \models \beta$)
- Complete: if φ unsat ($\alpha \models \beta$), then it returns an empty clause
- Time-inefficient
- Very Memory-inefficient (exponential in memory)
- Many different strategies

Very-Basic PL-Resolution Procedure

```
function PL-RESOLUTION(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each pair of clauses C_i, C_j in clauses do
         resolvents ← PL-RESOLVE(C_i, C_j)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \quad \left(e.g., \quad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \qquad \left(e.g., \qquad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \quad \left(e.g., \quad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \quad \left(e.g., \quad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \quad \left(e.g., \quad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \quad \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Improvements: Subsumption & Unit Propagation

Alternative "set" notation (Γ clause set):

$$\frac{\Gamma, \phi_1, ..\phi_n}{\Gamma, \phi'_1, ..\phi'_{n'}} \quad \left(e.g., \quad \frac{\Gamma, C_1 \vee p, C_2 \vee \neg p}{\Gamma, C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2,} \right)$$

- Clause Subsumption (C clause):

$$\frac{\Gamma \wedge C \wedge (C \vee \bigvee_i l_i)}{\Gamma \wedge (C)}$$

- Unit Resolution:

$$\frac{\Gamma \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma \wedge (l) \wedge (\bigvee_i l_i)}$$

- Unit Subsumption:

$$\frac{\Gamma \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma \wedge (l)}$$

- Unit Propagation = Unit Resolution + Unit Subsumption

Outline

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The Davis-Putnam-Longemann-Loveland Procedure

- Tries to build an assignment μ satisfying φ
- At each step assigns a truth value to (all instances of) one atom
- Performs deterministic choices (mostly unit-propagation) first
- The grandfather of the most efficient SAT solvers
- Correct and complete
- Much more efficient than PL-Resolution
- Requires polynomial space

The DPLL Procedure [cont.]

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ 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*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
      DPLL($clauses, rest, model \cup \{P{=}false\}$))

Pure-Symbol Rule out of date, no more used in modern solvers.

The DPLL Procedure [cont.]

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*

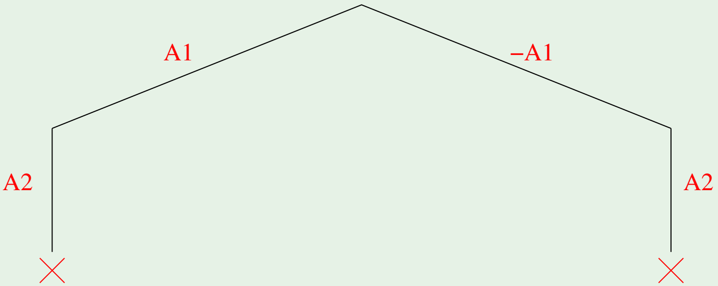
  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P,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

Pure-Symbol Rule out of date, no more used in modern solvers.

DPLL: Example

DPLL search tree

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$



Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - "static": scores updated only at the end of a branch
  - "local": privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on "similar" problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - “static”: scores updated only at the end of a branch
  - “local”: privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on “similar” problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

Modern CDCL SAT Solvers

- Non-recursive, stack-based implementations
- Based on Conflict-Driven Clause-Learning (CDCL) schema
  - inspired to conflict-driven backjumping and learning in CSPs
  - learns implied clauses as nogoods
- Random restarts
  - abandon the current search tree and restart on top level
  - previously-learned clauses maintained
- Smart literal selection heuristics (ex: VSIDS)
  - “static”: scores updated only at the end of a branch
  - “local”: privileges variable in recently learned clauses
- Smart preprocessing/inprocessing technique to simplify formulas
- Smart indexing techniques (e.g. 2-watched literals)
  - efficiently do/undo assignments and reveal unit clauses
- Allow Incremental Calls (stack-based interface)
  - allow for reusing previous search on “similar” problems

Can handle industrial problems with $10^6 - 10^7$ variables and clauses!

DPLL Chronological Backtracking: Drawbacks

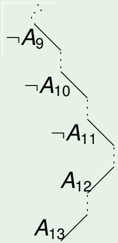
Chronological backtracking always backtracks to the most recent branching point, even though a higher backtrack could be possible
$\Longrightarrow$ lots of useless search!

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...

DPLL Chronological Backtracking: Example

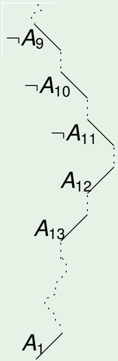
$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ...\}$
(initial assignment)

DPLL Chronological Backtracking: Example

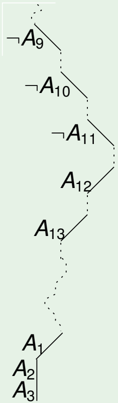
$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ $\surd$
$c_8 : A_1 \vee A_8$ $\surd$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1\}$
... (branch on A_1)

DPLL Chronological Backtracking: Example

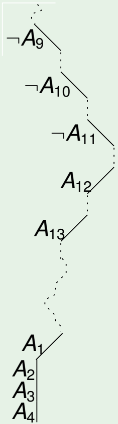
$c_1 : \neg A_1 \vee A_2$ √
$c_2 : \neg A_1 \vee A_3 \vee A_9$ √
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √
$c_8 : A_1 \vee A_8$ √
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1, A_2, A_3\}$
(unit A_2, A_3)

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \vee A_2$ √
$c_2 : \neg A_1 \vee A_3 \vee A_9$ √
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$ √
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √
$c_8 : A_1 \vee A_8$ √
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1, A_2, A_3, A_4\}$
(unit A_4)

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \lor A_2$ $\surd$

$c_2 : \neg A_1 \lor A_3 \lor A_9$ $\surd$

$c_3 : \neg A_2 \lor \neg A_3 \lor A_4$ $\surd$

$c_4 : \neg A_4 \lor A_5 \lor A_{10}$ $\surd$

$c_5 : \neg A_4 \lor A_6 \lor A_{11}$ $\surd$

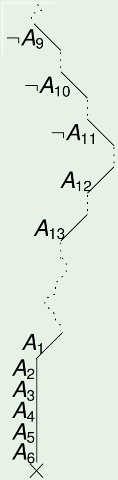
$c_6 : \neg A_5 \lor \neg A_6$ $\times$

$c_7 : A_1 \lor A_7 \lor \neg A_{12}$ $\surd$

$c_8 : A_1 \lor A_8$ $\surd$

$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$

...

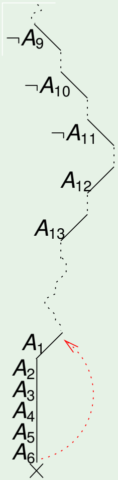


$\{..., \neg A_9, \neg A_{10}, \neg A_{1\neg A_4 1}, A_{12}, A_{13}, ..., A_1, A_2, A_3, A_4, A_5, A_6\}$

(unit A_5, A_6) $\Longrightarrow$ conflict

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ...\}$
$\Longrightarrow$ backtrack up to A_1

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \vee A_2$ √

$c_2 : \neg A_1 \vee A_3 \vee A_9$ √

$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$

$c_4 : \neg A_4 \vee A_5 \vee A_{10}$

$c_5 : \neg A_4 \vee A_6 \vee A_{11}$

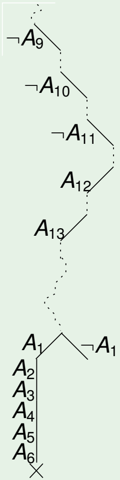
$c_6 : \neg A_5 \vee \neg A_6$

$c_7 : A_1 \vee A_7 \vee \neg A_{12}$

$c_8 : A_1 \vee A_8$

$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$

...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., \neg A_1\}$
(unit $\neg A_1$)

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \lor A_2$ ✓

$c_2 : \neg A_1 \lor A_3 \lor A_9$ ✓

$c_3 : \neg A_2 \lor \neg A_3 \lor A_4$

$c_4 : \neg A_4 \lor A_5 \lor A_{10}$

$c_5 : \neg A_4 \lor A_6 \lor A_{11}$

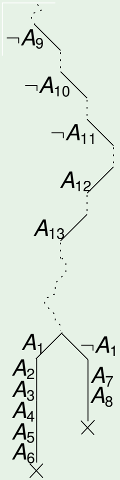
$c_6 : \neg A_5 \lor \neg A_6$

$c_7 : A_1 \lor A_7 \lor \neg A_{12}$ ✓

$c_8 : A_1 \lor A_8$ ✓

$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$ ×

...

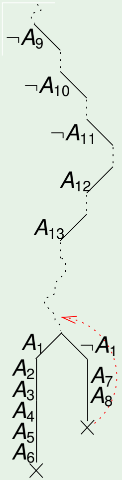


$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., \neg A_1, A_7, A_8\}$

(unit A_7, A_8) $\Longrightarrow$ conflict

DPLL Chronological Backtracking: Example

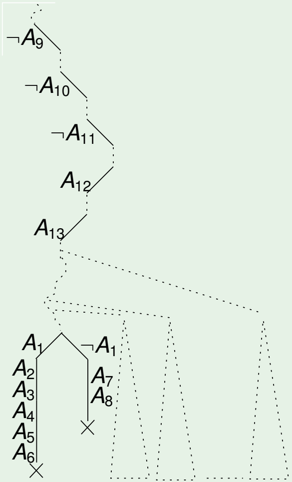
$c_1 : \neg A_1 \lor A_2$
$c_2 : \neg A_1 \lor A_3 \lor A_9$
$c_3 : \neg A_2 \lor \neg A_3 \lor A_4$
$c_4 : \neg A_4 \lor A_5 \lor A_{10}$
$c_5 : \neg A_4 \lor A_6 \lor A_{11}$
$c_6 : \neg A_5 \lor \neg A_6$
$c_7 : A_1 \lor A_7 \lor \neg A_{12}$
$c_8 : A_1 \lor A_8$
$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ...\}$
$\Longrightarrow$ backtrack to the most recent open branching point

DPLL Chronological Backtracking: Example

$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ...\}$
$\Longrightarrow$ lots of useless search before backtracking up to A_{13}!

Outline

Backjumping and Learning (Original Strategy)

- Idea: when a branch μ fails,
  (i) conflict analysis: find the conflict set $\eta \subseteq \mu$ by generating the conflict clause $C \stackrel{\text{def}}{=} \neg\eta$ via resolution from the falsified clause, using the "Decision" criterion;
  (ii) learning: add the conflict clause C to the clause set
  (iii) backjumping: backtrack to the most recent branching point s.t. the stack does not fully contain η, and then unit-propagate the unassigned literal on C

Backjumping and Learning (Original Strategy)

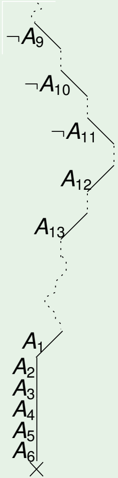
- Idea: when a branch μ fails,
  (i) conflict analysis: find the conflict set $\eta \subseteq \mu$ by generating the conflict clause $C \stackrel{\text{def}}{=} \neg\eta$ via resolution from the falsified clause, using the "Decision" criterion;
  (ii) learning: add the conflict clause C to the clause set
  (iii) backjumping: backtrack to the most recent branching point s.t. the stack does not fully contain η, and then unit-propagate the unassigned literal on C

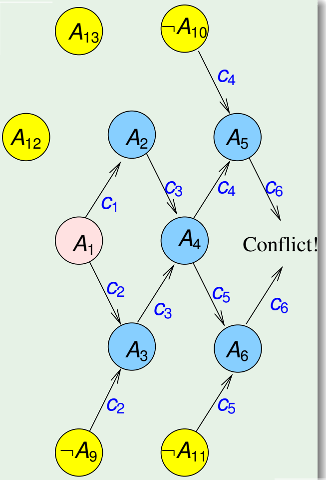
Backjumping and Learning (Original Strategy)

- Idea: when a branch μ fails,
  (i) conflict analysis: find the conflict set $\eta \subseteq \mu$ by generating the conflict clause $C \stackrel{\text{def}}{=} \neg\eta$ via resolution from the falsified clause, using the "Decision" criterion;
  (ii) learning: add the conflict clause C to the clause set
  (iii) backjumping: backtrack to the most recent branching point s.t. the stack does not fully contain η, and then unit-propagate the unassigned literal on C

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$ √
$c_2 : \neg A_1 \vee A_3 \vee A_9$ √
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$ √
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$ √
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$ √
$c_6 : \neg A_5 \vee \neg A_6$ ×
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √
$c_8 : A_1 \vee A_8$ √
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
...





$\{..., \neg A_9, \neg A_{10}, \neg A_{1_{\neg A_4}1}, A_{12}, A_{13}, ..., A_1, A_2, A_3, A_4, A_5, A_6\}$
(unit A_5, A_6) $\Longrightarrow$ conflict;
falsified clause: $\neg A_5 \vee \neg A_6$

Conflict analysis

Criterium: "decision"

1. $C :=$ falsified clause
2. repeat
   (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in C
      - antecedent clause for l: causing the unit-propagation of on l

   until C contains only decision literals ("decision" criterion)

Example

Falsified cl.

$$\frac{\neg A_4 \lor A_6 \lor A_{11} \qquad \neg A_5 \lor \neg A_6}{\neg A_4 \lor \neg A_5 \lor A_{11}} \ (A_6)$$

$$\frac{\neg A_4 \lor A_5 \lor A_{10} \qquad \neg A_4 \lor \neg A_5 \lor A_{11}}{\neg A_4 \lor A_{10} \lor A_{11}} \ (A_5)$$

$$\frac{\neg A_2 \lor \neg A_3 \lor A_4 \qquad \neg A_4 \lor A_{10} \lor A_{11}}{\neg A_2 \lor \neg A_3 \lor A_{10} \lor A_{11}} \ (A_4)$$

$$\frac{\neg A_1 \lor A_3 \lor A_9 \qquad \neg A_2 \lor \neg A_3 \lor A_{10} \lor A_{11}}{\neg A_2 \lor \neg A_1 \lor A_9 \lor A_{10} \lor A_{11}} \ (A_3)$$

$$\frac{\neg A_1 \lor A_2 \qquad \neg A_2 \lor \neg A_1 \lor A_9 \lor A_{10} \lor A_{11}}{\neg A_1 \lor A_9 \lor A_{10} \lor A_{11}} \ (A_2)$$

Conflict analysis

Criterium: "decision"

1. $C :=$ falsified clause
2. repeat
   (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in C
      - antecedent clause for l: causing the unit-propagation of on l

   until C contains only decision literals ("decision" criterion)

Example

Falsified cl.

$$\frac{\neg A_4 \vee A_6 \vee A_{11} \qquad \neg A_5 \vee \neg A_6}{\neg A_4 \vee \neg A_5 \vee A_{11}}\ (A_6)$$

$$\frac{\neg A_4 \vee A_5 \vee A_{10} \qquad \neg A_4 \vee \neg A_5 \vee A_{11}}{\neg A_4 \vee A_{10} \vee A_{11}}\ (A_5)$$

$$\frac{\neg A_2 \vee \neg A_3 \vee A_4 \qquad \neg A_4 \vee A_{10} \vee A_{11}}{\neg A_2 \vee \neg A_3 \vee A_{10} \vee A_{11}}\ (A_4)$$

$$\frac{\neg A_1 \vee A_3 \vee A_9 \qquad \neg A_2 \vee \neg A_3 \vee A_{10} \vee A_{11}}{\neg A_2 \vee \neg A_1 \vee A_9 \vee A_{10} \vee A_{11}}\ (A_3)$$

$$\frac{\neg A_1 \vee A_2 \qquad \neg A_2 \vee \neg A_1 \vee A_9 \vee A_{10} \vee A_{11}}{\neg A_1 \vee A_9 \vee A_{10} \vee A_{11}}\ (A_2)$$

Conflict analysis

Criterium: "decision"

1. $C :=$ falsified clause
2. repeat
   (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in C
      - antecedent clause for l: causing the unit-propagation of on l

   until C contains only decision literals ("decision" criterion)

Example

$$\frac{\neg A_4 \lor A_6 \lor A_{11} \qquad \overbrace{\neg A_5 \lor \neg A_6}^{\textit{Falsified cl.}}}{\neg A_4 \lor \neg A_5 \lor A_{11}}\ (A_6)$$

$$\frac{\neg A_4 \lor A_5 \lor A_{10} \qquad \neg A_4 \lor \neg A_5 \lor A_{11}}{\neg A_4 \lor A_{10} \lor A_{11}}\ (A_5)$$

$$\frac{\neg A_2 \lor \neg A_3 \lor A_4 \qquad \neg A_4 \lor A_{10} \lor A_{11}}{\neg A_2 \lor \neg A_3 \lor A_{10} \lor A_{11}}\ (A_4)$$

$$\frac{\neg A_1 \lor A_3 \lor A_9 \qquad \neg A_2 \lor \neg A_3 \lor A_{10} \lor A_{11}}{\neg A_2 \lor \neg A_1 \lor A_9 \lor A_{10} \lor A_{11}}\ (A_3)$$

$$\frac{\neg A_1 \lor A_2 \qquad \neg A_2 \lor \neg A_1 \lor A_9 \lor A_{10} \lor A_{11}}{\neg A_1 \lor A_9 \lor A_{10} \lor A_{11}}\ (A_2)$$

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$ √

$c_2 : \neg A_1 \vee A_3 \vee A_9$ √

$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$ √

$c_4 : \neg A_4 \vee A_5 \vee A_{10}$ √

$c_5 : \neg A_4 \vee A_6 \vee A_{11}$ √

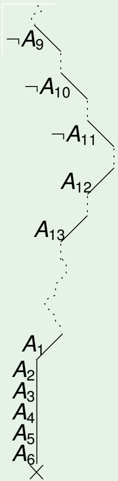
$c_6 : \neg A_5 \vee \neg A_6$ ×

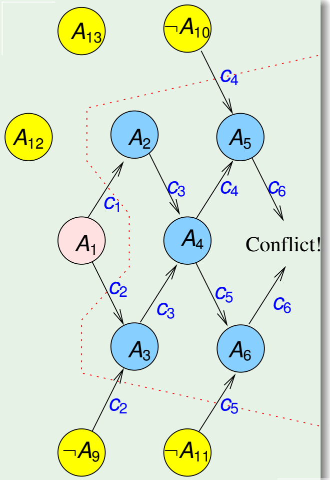
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √

$c_8 : A_1 \vee A_8$ √

$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$

...



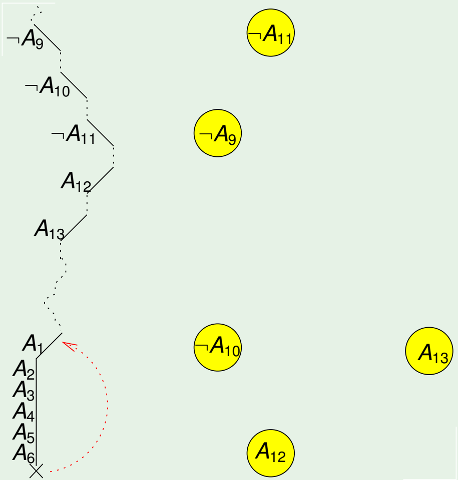


$\Longrightarrow$ Conflict set: $\{\neg A_9, \neg A_{10}, \neg A_{11}, A_1\}$ ("decision" schema)

$\Longrightarrow$ learn the conflict clause $c_{10} := A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$

The Original Backjumping Strategy: Example

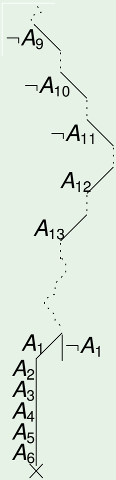
$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$
...

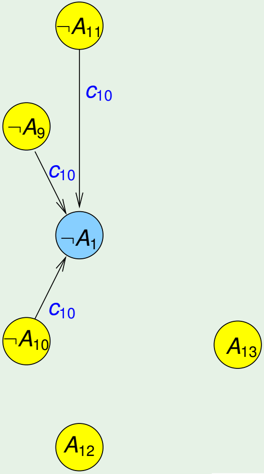


$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ...\}$
$\Longrightarrow$ backtrack up to A_1

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$ √
$c_2 : \neg A_1 \vee A_3 \vee A_9$ √
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$ √
...

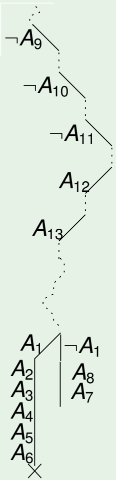


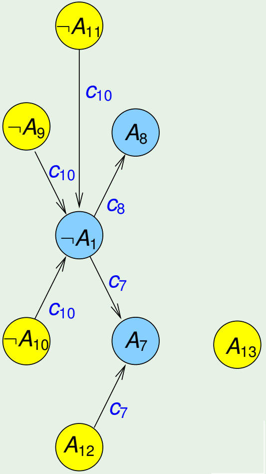


$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., \neg A_1\}$
(unit $\neg A_1$)

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$ √
$c_2 : \neg A_1 \vee A_3 \vee A_9$ √
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √
$c_8 : A_1 \vee A_8$ √
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$ √
...





$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., \neg A_1, A_7, A_8\}$
(unit A_7, A_8)

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$ √

$c_2 : \neg A_1 \vee A_3 \vee A_9$ √

$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$

$c_4 : \neg A_4 \vee A_5 \vee A_{10}$

$c_5 : \neg A_4 \vee A_6 \vee A_{11}$

$c_6 : \neg A_5 \vee \neg A_6$

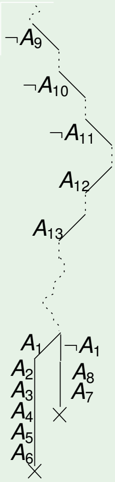
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$ √

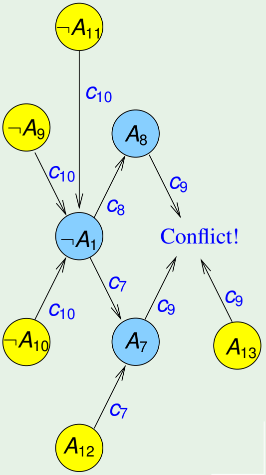
$c_8 : A_1 \vee A_8$ √

$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$ ×

$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$ √

...



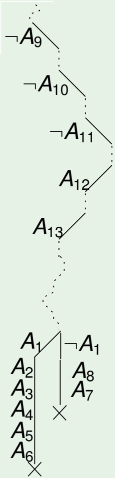


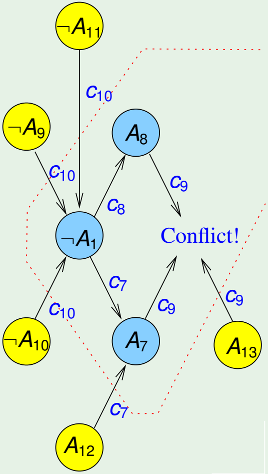
$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., \neg A_1, A_7, A_8\}$

Conflict!

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \lor A_2$ ✓
$c_2 : \neg A_1 \lor A_3 \lor A_9$ ✓
$c_3 : \neg A_2 \lor \neg A_3 \lor A_4$
$c_4 : \neg A_4 \lor A_5 \lor A_{10}$
$c_5 : \neg A_4 \lor A_6 \lor A_{11}$
$c_6 : \neg A_5 \lor \neg A_6$
$c_7 : A_1 \lor A_7 \lor \neg A_{12}$ ✓
$c_8 : A_1 \lor A_8$ ✓
$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$ ×
$c_{10} : A_9 \lor A_{10} \lor A_{11} \lor \neg A_1$ ✓
...

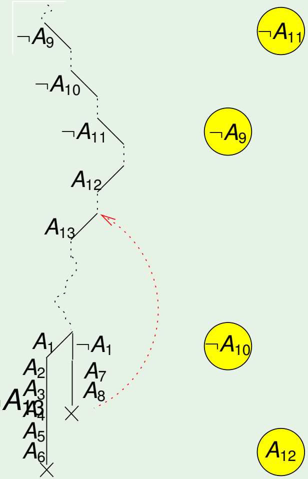




$\Longrightarrow$ conflict set: $\{\neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}\}$.
$\Longrightarrow$ learn $C_{11} := A_9 \lor A_{10} \lor A_{11} \lor \neg A_{12} \lor \neg A_{13}$

The Original Backjumping Strategy: Example

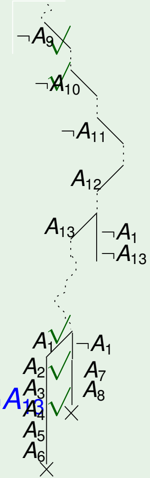
$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$
$c_{11} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_{12} \vee \neg A_{13}$
...

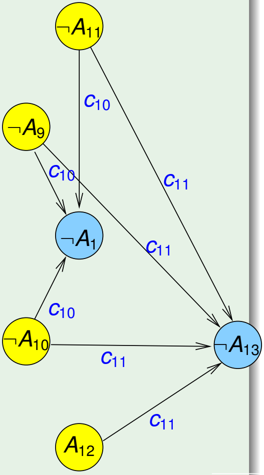


$\Longrightarrow$ backtrack to A_{13} $\Longrightarrow$ Lots of search saved!

The Original Backjumping Strategy: Example

$c_1 : \neg A_1 \vee A_2$
$c_2 : \neg A_1 \vee A_3 \vee A_9$
$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$
$c_4 : \neg A_4 \vee A_5 \vee A_{10}$
$c_5 : \neg A_4 \vee A_6 \vee A_{11}$
$c_6 : \neg A_5 \vee \neg A_6$
$c_7 : A_1 \vee A_7 \vee \neg A_{12}$
$c_8 : A_1 \vee A_8$
$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
$c_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$
$c_{11} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_{12} \vee \neg A_{13}$
...





$\Longrightarrow$ backtrack to A_{13}, then set A_{13} and A_1 to $\bot$,...

Learning

Idea: When a conflict set η is revealed, then $C \stackrel{\text{def}}{=} \neg\eta$ added to φ
$\Longrightarrow$ the solver will no more generate an assignment containing η: when $|\eta| - 1$ literals in η are assigned, the other is set $\perp$ by unit-propagation on C
$\Longrightarrow$ Drastic pruning of the search!

Learning – example

$c_1 : \neg A_1 \lor A_2$

$c_2 : \neg A_1 \lor A_3 \lor A_9$

$c_3 : \neg A_2 \lor \neg A_3 \lor A_4$

$c_4 : \neg A_4 \lor A_5 \lor A_{10}$

$c_5 : \neg A_4 \lor A_6 \lor A_{11}$

$c_6 : \neg A_5 \lor \neg A_6$

$c_7 : A_1 \lor A_7 \lor \neg A_{12}$

$c_8 : A_1 \lor A_8$

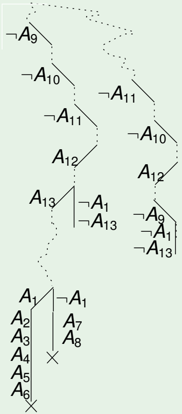
$c_9 : \neg A_7 \lor \neg A_8 \lor \neg A_{13}$ √

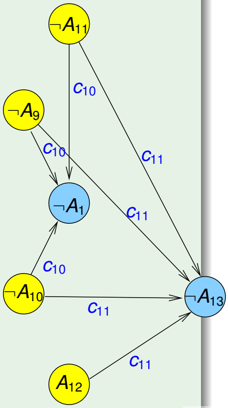
$c_{10} : A_9 \lor A_{10} \lor A_{11} \lor \neg A_1$ √

$c_{11} : A_9 \lor A_{10} \lor A_{11} \lor \neg A_{12} \lor \neg A_{13}$ √

...

$\Longrightarrow$ Unit: $\{\neg A_1, \neg A_{13}\}$





Outline

1. Propositional Logic
2. Propositional Reasoning
   - Resolution
   - DPLL
   - Modern CDCL SAT Solvers
   - Reasoning with Horn Formulas
   - Local Search
3. Knowledge-Based Agents
4. Agents Based on Propositional Reasoning

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:
$$\begin{array}{rl} A_1 \lor \neg A_2 & \text{// } \textit{definite} \\ A_2 \lor \neg A_3 \lor \neg A_4 & \text{// } \textit{definite} \\ \neg A_5 \lor \neg A_3 \lor \neg A_4 & \text{// } \textit{goal} \\ A_3 & \text{// } \textit{definite} \end{array}$$
- Intuition: implications between positive Boolean variables:
$$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \land A_4) \rightarrow & A_2 \\ (A_5 \land A_3 \land A_4) \rightarrow & \bot \\ & A_3 \end{array}$$
- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11$; $(\neg In11 \lor \neg MoveFrom11To12 \lor In12)$;
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:
$$\begin{array}{rl} A_1 \vee \neg A_2 & // \textit{definite} \\ A_2 \vee \neg A_3 \vee \neg A_4 & // \textit{definite} \\ \neg A_5 \vee \neg A_3 \vee \neg A_4 & // \textit{goal} \\ A_3 & // \textit{definite} \end{array}$$
- Intuition: implications between positive Boolean variables:
$$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \wedge A_4) \rightarrow & A_2 \\ (A_5 \wedge A_3 \wedge A_4) \rightarrow & \bot \\ & A_3 \end{array}$$
- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11;\ (\neg In11 \vee \neg MoveFrom11To12 \vee In12);$
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:
  $$\begin{array}{ll} A_1 \vee \neg A_2 & // \textit{ definite} \\ A_2 \vee \neg A_3 \vee \neg A_4 & // \textit{ definite} \\ \neg A_5 \vee \neg A_3 \vee \neg A_4 & // \textit{ goal} \\ A_3 & // \textit{ definite} \end{array}$$
- Intuition: implications between positive Boolean variables:
  $$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \wedge A_4) \rightarrow & A_2 \\ (A_5 \wedge A_3 \wedge A_4) \rightarrow & \bot \\ & A_3 \end{array}$$
- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11$; $(\neg In11 \vee \neg MoveFrom11To12 \vee In12)$;
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:

$$\begin{array}{ll} A_1 \vee \neg A_2 & // \textit{ definite} \\ A_2 \vee \neg A_3 \vee \neg A_4 & // \textit{ definite} \\ \neg A_5 \vee \neg A_3 \vee \neg A_4 & // \textit{ goal} \\ A_3 & // \textit{ definite} \end{array}$$

- Intuition: implications between positive Boolean variables:

$$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \wedge A_4) \rightarrow & A_2 \\ (A_5 \wedge A_3 \wedge A_4) \rightarrow & \bot \\ & A_3 \end{array}$$

- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11$; $(\neg In11 \vee \neg MoveFrom11To12 \vee In12)$;
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:

$$\begin{array}{ll} A_1 \vee \neg A_2 & //\ \textit{definite} \\ A_2 \vee \neg A_3 \vee \neg A_4 & //\ \textit{definite} \\ \neg A_5 \vee \neg A_3 \vee \neg A_4 & //\ \textit{goal} \\ A_3 & //\ \textit{definite} \end{array}$$

- Intuition: implications between positive Boolean variables:

$$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \wedge A_4) \rightarrow & A_2 \\ (A_5 \wedge A_3 \wedge A_4) \rightarrow & \bot \\ & A_3 \end{array}$$

- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11$; $(\neg In11 \vee \neg MoveFrom11To12 \vee In12)$;
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Horn Formulas

- A Horn clause is a clause containing at most one positive literal
  - a definite clause is a clause containing exactly one positive literal
  - a goal clause is a clause containing no positive literal
- A Horn formula is a conjunction/set of Horn clauses
- Ex:
$$\begin{array}{ll} A_1 \lor \neg A_2 & // \textit{definite} \\ A_2 \lor \neg A_3 \lor \neg A_4 & // \textit{definite} \\ \neg A_5 \lor \neg A_3 \lor \neg A_4 & // \textit{goal} \\ A_3 & // \textit{definite} \end{array}$$
- Intuition: implications between positive Boolean variables:
$$\begin{array}{rl} A_2 \rightarrow & A_1 \\ (A_3 \land A_4) \rightarrow & A_2 \\ (A_5 \land A_3 \land A_4) \rightarrow & \bot \\ & A_3 \end{array}$$
- Often allow to represent knowledge-base entailment $KB \models \alpha$:
  - knowledge base KB written as sets of definite clauses
    ex: $In11$; $(\neg In11 \lor \neg MoveFrom11To12 \lor In12)$;
  - goal $\neg\alpha$ as a goal clause
    ex: $\neg In12$

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
  1. Eliminate unit clauses by propagating their value;
  2. If an empty clause is generated, return unsat
  3. Otherwise, every clause contains at least one negative literal

  $\Longrightarrow$ Assign all variables to $\bot$; return the assignment
- Alternatively: run DPLL, selecting always negative literals first

A simple polynomial procedure for Horn-SAT

```
function Horn_SAT(formula φ, assignment & μ) {
    Unit_Propagate(φ, μ);
    if (φ == ⊥)
        then return UNSAT;
    else {
        μ := μ ∪ ⋃_{A_i ∉ μ} {¬A_i};
        return SAT;
} }

function Unit_Propagate(formula & φ, assignment & μ)
    while (φ ≠ ⊤ and φ ≠ ⊥ and {a unit clause (l) occurs in φ}) do {
        φ = assign(φ, l);
        μ := μ ∪ {l};
} }
```

Example

$$\begin{array}{llll} \neg A_1 & \vee\ A_2 & \vee\neg A_3 \\ A_1 & \vee\neg A_3 & \vee\neg A_4 \\ \neg A_2 & \vee\neg A_4 & \\ A_3 & \vee\neg A_4 & \\ A_4 & & \end{array}$$

Example

$$
\begin{array}{llll}
\neg A_1 & \vee \; A_2 & \vee \neg A_3 \\
A_1 & \vee \neg A_3 & \vee \neg A_4 \\
\neg A_2 & \vee \neg A_4 & \\
A_3 & \vee \neg A_4 & \\
A_4 & &
\end{array}
$$

$\mu := \{A_4 := \top\}$

Example

$$
\begin{array}{rlll}
\neg A_1 & \vee \; A_2 & \vee \neg A_3 \\
A_1 & \vee \neg A_3 & \vee \neg A_4 \\
\neg A_2 & \vee \neg A_4 & \\
A_3 & \vee \neg A_4 & \\
A_4 & &
\end{array}
$$

$\mu := \{A_4 := \top, A_3 := \top\}$

Example

$$\begin{array}{rll} \neg A_1 & \vee\ A_2 & \vee \neg A_3 \\ A_1 & \vee \neg A_3 & \vee \neg A_4 \\ \neg A_2 & \vee \neg A_4 & \\ A_3 & \vee \neg A_4 & \\ A_4 & & \end{array}$$

$\mu := \{A_4 := \top, A_3 := \top, A_2 := \bot\}$

Example

$$\begin{array}{llll} \neg A_1 & \vee A_2 & \vee \neg A_3 & \times \\ A_1 & \vee \neg A_3 & \vee \neg A_4 & \\ \neg A_2 & \vee \neg A_4 & & \\ A_3 & \vee \neg A_4 & & \\ A_4 & & & \end{array}$$

$\mu := \{A_4 := \top, A_3 := \top, A_2 := \bot, A_1 := \top\} \Longrightarrow$ UNSAT

Example 2

$$
\begin{array}{lll}
A_1 & \vee\neg A_2 & \\
A_2 & \vee\neg A_5 & \vee\neg A_4 \\
A_4 & \vee\neg A_3 & \\
A_3 & &
\end{array}
$$

Example 2

$$\begin{array}{lll} A_1 & \vee \neg A_2 & \\ A_2 & \vee \neg A_5 & \vee \neg A_4 \\ A_4 & \vee \neg A_3 & \\ A_3 & & \end{array}$$

$\mu := \{A_3 := \top\}$

Example 2

$$\begin{array}{lll} A_1 & \vee\neg A_2 & \\ A_2 & \vee\neg A_5 & \vee\neg A_4 \\ A_4 & \vee\neg A_3 & \\ A_3 & & \end{array}$$

$\mu := \{A_3 := \top, A_4 := \top\}$

Example 2

$$\begin{array}{lll} A_1 & \vee \neg A_2 & \\ A_2 & \vee \neg A_5 & \vee \neg A_4 \\ A_4 & \vee \neg A_3 & \\ A_3 & & \end{array}$$

$\mu := \{A_3 := \top, A_4 := \top\} \Longrightarrow$ SAT

Outline

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random
  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random
  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - “neighbour states” differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause *C*
  - Variable selection:

  prob. p: flip variable from *C* at random
  prob. 1-p: flip variable from *C* causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random

  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random
  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random

  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause *C*
  - Variable selection:

  prob. p: flip variable from *C* at random
  prob. 1-p: flip variable from *C* causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

Local Search with SAT

- Similar to Local Search for CSPs
- Input: set of clauses
- Use total truth assignments
  - allow states with unsatisfied clauses
  - "neighbour states" differ for one variable truth value
  - steps: reassign variable truth values
- Cost: # of unsatisfied clauses
- Stochastic local search [see Ch. 4] applies to SAT as well
  - random walk, simulated annealing, GAs, taboo search, ...
- The WalkSAT stochastic local search
  - Clause selection: randomly select an unsatisfied clause C
  - Variable selection:

  prob. p: flip variable from C at random

  prob. 1-p: flip variable from C causing a minimum number of unsat clauses
- Note: can detect only satisfiability, not unsatisfiability
- Many variants

The WalkSAT Procedure

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*

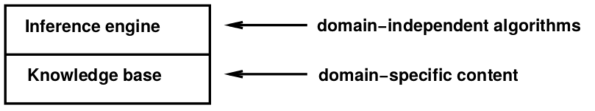
Outline

A Quote

> You can think about deep learning as equivalent to ... our visual cortex or auditory cortex. But, of course, true intelligence is a lot more than just that, you have to recombine it into higher-level thinking and symbolic reasoning, a lot of the things classical AI tried to deal with in the 80s. ... We would like to build up to this symbolic level of reasoning - maths, language, and logic. So that's a big part of our work.
>
> Demis Hassabis, CEO of Google Deepmind

Knowledge Representation and Reasoning

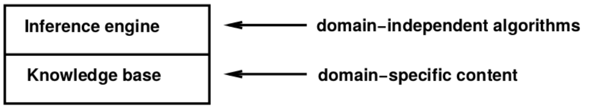
- Knowledge Representation & Reasoning (KR&R): the field of AI dedicated to representing knowledge of the world in a form a computer system can utilize to solve complex tasks
- The class of systems/agents that derive from this approach are called knowledge based (KB) systems/agents
- A KB agent maintains a knowledge base (KB) of facts
  - collection of domain-specific facts believed by the agent
  - expressed in a formal language (e.g. propositional logic)
  - represent the agent's representation of the world
  - initially contains the background knowledge
  - KB queries and updates via logical entailment, performed by an inference engine
- Inference engine allows for inferring actions and new knowledge
  - domain-independent algorithms, can answer any question



(© S. Russell & P. Norwig, AIMA)

Knowledge Representation and Reasoning

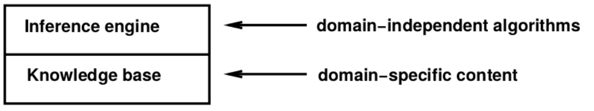
- Knowledge Representation & Reasoning (KR&R): the field of AI dedicated to representing knowledge of the world in a form a computer system can utilize to solve complex tasks
- The class of systems/agents that derive from this approach are called knowledge based (KB) systems/agents
- A KB agent maintains a knowledge base (KB) of facts
  - collection of domain-specific facts believed by the agent
  - expressed in a formal language (e.g. propositional logic)
  - represent the agent's representation of the world
  - initially contains the background knowledge
  - KB queries and updates via logical entailment, performed by an inference engine
- Inference engine allows for inferring actions and new knowledge
  - domain-independent algorithms, can answer any question



(© S. Russell & P. Norwig, AIMA)

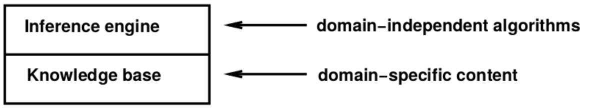
Knowledge Representation and Reasoning

- Knowledge Representation & Reasoning (KR&R): the field of AI dedicated to representing knowledge of the world in a form a computer system can utilize to solve complex tasks
- The class of systems/agents that derive from this approach are called knowledge based (KB) systems/agents
- A KB agent maintains a knowledge base (KB) of facts
  - collection of domain-specific facts believed by the agent
  - expressed in a formal language (e.g. propositional logic)
  - represent the agent's representation of the world
  - initially contains the background knowledge
  - KB queries and updates via logical entailment, performed by an inference engine
- Inference engine allows for inferring actions and new knowledge
  - domain-independent algorithms, can answer any question



Knowledge Representation and Reasoning

- Knowledge Representation & Reasoning (KR&R): the field of AI dedicated to representing knowledge of the world in a form a computer system can utilize to solve complex tasks
- The class of systems/agents that derive from this approach are called knowledge based (KB) systems/agents
- A KB agent maintains a knowledge base (KB) of facts
  - collection of domain-specific facts believed by the agent
  - expressed in a formal language (e.g. propositional logic)
  - represent the agent's representation of the world
  - initially contains the background knowledge
  - KB queries and updates via logical entailment, performed by an inference engine
- Inference engine allows for inferring actions and new knowledge
  - domain-independent algorithms, can answer any question



(© S. Russell & P. Norwig, AIMA)

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): “Patient x is allergic to medication m”
  - (KB general rule): “Anybody allergic to m is also allergic to m’.”
  - (KB general rule): “If x is allergic to m’, do not prescribe m’ for x.”
  - (query): “Prescribe m’ for x?”
  - (answer) No (because patient x is allergic to medication m’)
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \land \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): "Patient x is allergic to medication m"
  - (KB general rule): "Anybody allergic to m is also allergic to m'."
  - (KB general rule): "If x is allergic to m', do not prescribe m' for x."
  - (query): "Prescribe m' for x?"
  - (answer) No (because patient x is allergic to medication m')
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): “Patient x is allergic to medication m”
  - (KB general rule): “Anybody allergic to m is also allergic to m’.”
  - (KB general rule): “If x is allergic to m’, do not prescribe m’ for x.”
  - (query): “Prescribe m’ for x?”
  - (answer) No (because patient x is allergic to medication m’)
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Reasoning

- Reasoning: formal manipulation of the symbols representing a collection of beliefs to produce representations of new ones
- Logical entailment ($KB \models \alpha$) is the fundamental operation
- Ex:
  - (KB acquired fact): “Patient x is allergic to medication m”
  - (KB general rule): “Anybody allergic to m is also allergic to m’.”
  - (KB general rule): “If x is allergic to m’, do not prescribe m’ for x.”
  - (query): “Prescribe m’ for x?”
  - (answer) No (because patient x is allergic to medication m’)
- Other forms of reasoning (last part of this course)
  - Probablistic reasoning
- Other forms of reasoning (not addressed in this course)
  - Abductive reasoning (aka diagnosis): given KB and β, conjecture hypotheses α s.t $(KB \wedge \alpha) \models \beta$
  - Abductive reasoning: from a set of observation find a general rule

Knowledge-Based Agents (aka Logic Agents)

- Logic agents: combine domain knowledge with current percepts to infer hidden aspects of current state prior to selecting actions
  - Crucial in partially observable environments
- KB Agent must be able to:
  - represent states and actions
  - incorporate new percepts
  - update internal representation of the world
  - deduce hidden properties of the world
  - deduce appropriate actions
- Agents can be described at different levels
  - knowledge level (declarative approach): behaviour completely described by the sentences stored in the KB
  - implementation level (procedural approach): behaviour described as program code
- Declarative approach to building an agent (or other system):
  - Tell the KB what it needs to know (update KB)
  - Ask what to do (answers should follow logically from KB & query)

Knowledge-Based Agents (aka Logic Agents)

- Logic agents: combine domain knowledge with current percepts to infer hidden aspects of current state prior to selecting actions
  - Crucial in partially observable environments
- KB Agent must be able to:
  - represent states and actions
  - incorporate new percepts
  - update internal representation of the world
  - deduce hidden properties of the world
  - deduce appropriate actions
- Agents can be described at different levels
  - knowledge level (declarative approach): behaviour completely described by the sentences stored in the KB
  - implementation level (procedural approach): behaviour described as program code
- Declarative approach to building an agent (or other system):
  - Tell the KB what it needs to know (update KB)
  - Ask what to do (answers should follow logically from KB & query)

Knowledge-Based Agents (aka Logic Agents)

- Logic agents: combine domain knowledge with current percepts to infer hidden aspects of current state prior to selecting actions
  - Crucial in partially observable environments
- KB Agent must be able to:
  - represent states and actions
  - incorporate new percepts
  - update internal representation of the world
  - deduce hidden properties of the world
  - deduce appropriate actions
- Agents can be described at different levels
  - knowledge level (declarative approach): behaviour completely described by the sentences stored in the KB
  - implementation level (procedural approach): behaviour described as program code
- Declarative approach to building an agent (or other system):
  - Tell the KB what it needs to know (update KB)
  - Ask what to do (answers should follow logically from KB & query)

Knowledge-Based Agents (aka Logic Agents)

- Logic agents: combine domain knowledge with current percepts to infer hidden aspects of current state prior to selecting actions
  - Crucial in partially observable environments
- KB Agent must be able to:
  - represent states and actions
  - incorporate new percepts
  - update internal representation of the world
  - deduce hidden properties of the world
  - deduce appropriate actions
- Agents can be described at different levels
  - knowledge level (declarative approach): behaviour completely described by the sentences stored in the KB
  - implementation level (procedural approach): behaviour described as program code
- Declarative approach to building an agent (or other system):
  - Tell the KB what it needs to know (update KB)
  - Ask what to do (answers should follow logically from KB & query)

Knowledge-Based Agent: General Schema

- Given a percept, the agent
  - Tells the KB of the percept at time step t
  - ASKs the KB for the best action to do at time step t
  - Tells the KB that it has in fact taken that action
- Details hidden in three functions: MAKE-PERCEPT-SENTENCE, MAKE-ACTION-QUERY, MAKE-ACTION-SENTENCE
  - construct logic sentences
  - implement the interface between sensors/actuators and KRR core
- Tell and Ask may require complex logical inference

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Knowledge-Based Agent: General Schema

- Given a percept, the agent
  - Tells the KB of the percept at time step t
  - ASKs the KB for the best action to do at time step t
  - Tells the KB that it has in fact taken that action
- Details hidden in three functions: MAKE-PERCEPT-SENTENCE, MAKE-ACTION-QUERY, MAKE-ACTION-SENTENCE
  - construct logic sentences
  - implement the interface between sensors/actuators and KRR core
- Tell and Ask may require complex logical inference

function KB-AGENT(*percept*) **returns** an *action*
  persistent: *KB*, a knowledge base
        t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  $t \leftarrow t + 1$
  return *action*

(© S. Russell & P. Norwig, AIMA)

Knowledge-Based Agent: General Schema

- Given a percept, the agent
  - Tells the KB of the percept at time step t
  - ASKs the KB for the best action to do at time step t
  - Tells the KB that it has in fact taken that action
- Details hidden in three functions: MAKE-PERCEPT-SENTENCE, MAKE-ACTION-QUERY, MAKE-ACTION-SENTENCE
  - construct logic sentences
  - implement the interface between sensors/actuators and KRR core
- Tell and Ask may require complex logical inference

```
function KB-AGENT(percept) returns an action
   persistent: KB, a knowledge base
               t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Example: The Wumpus World

Task Environment: PEAS Description

Performance measure:

- gold: +1000, death: -1000
- step: -1, using the arrow: -10

Environment:

- squares adjacent to Wumpus are stenchy
- squares adjacent to pit are breezy
- glitter iff gold is in the same square
- shooting kills Wumpus if you are facing it
- shooting uses up the only arrow
- grabbing picks up gold if in same square
- releasing drops the gold in same square

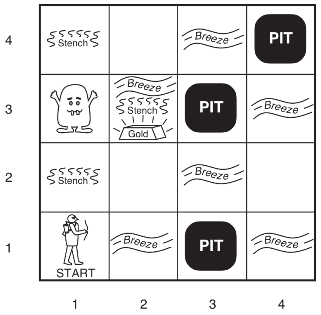
Actuators:

- Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



(© S. Russell & P. Norwig, AIMA)

Example: The Wumpus World

Task Environment: PEAS Description

Performance measure:

- gold: +1000, death: -1000
- step: -1, using the arrow: -10

Environment:

- squares adjacent to Wumpus are stenchy
- squares adjacent to pit are breezy
- glitter iff gold is in the same square
- shooting kills Wumpus if you are facing it
- shooting uses up the only arrow
- grabbing picks up gold if in same square
- releasing drops the gold in same square

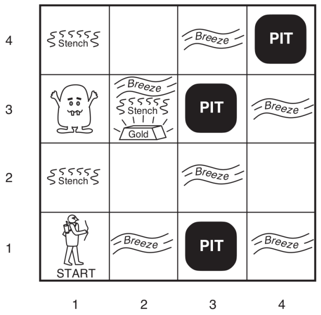
Actuators:

- Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



(© S. Russell & P. Norwig, AIMA)

Example: The Wumpus World

Task Environment: PEAS Description

Performance measure:

- gold: +1000, death: -1000
- step: -1, using the arrow: -10

Environment:

- squares adjacent to Wumpus are stenchy
- squares adjacent to pit are breezy
- glitter iff gold is in the same square
- shooting kills Wumpus if you are facing it
- shooting uses up the only arrow
- grabbing picks up gold if in same square
- releasing drops the gold in same square

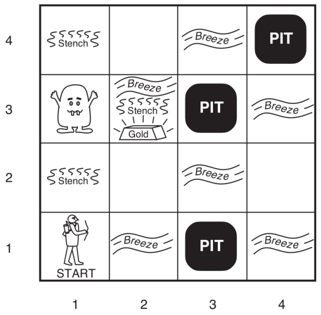
Actuators:

- Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



(© S. Russell & P. Norwig, AIMA)

Example: The Wumpus World

Task Environment: PEAS Description

Performance measure:

- gold: +1000, death: -1000
- step: -1, using the arrow: -10

Environment:

- squares adjacent to Wumpus are stenchy
- squares adjacent to pit are breezy
- glitter iff gold is in the same square
- shooting kills Wumpus if you are facing it
- shooting uses up the only arrow
- grabbing picks up gold if in same square
- releasing drops the gold in same square

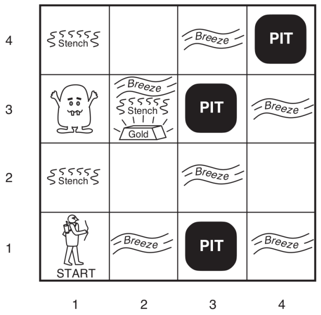
Actuators:

- Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

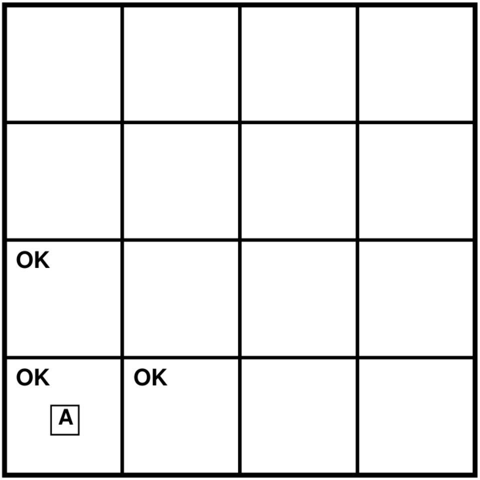
Wumpus World: Characterization

- Fully Observable? No: only local perception
- Deterministic? Yes: outcomes exactly specified
- Episodic? No: actions can have long-term consequences
- Static? Yes: Wumpus and Pits do not move
- Discrete? Yes
- Single-agent? Yes (Wumpus is essentially a natural feature)

Example: Exploring the Wumpus World

- The KB initially contains the rules of the environment.
- Agent is initially in 1,1
- Percepts:
  no stench, no breeze

$\Longrightarrow$ [1,2] and [2,1] OK



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

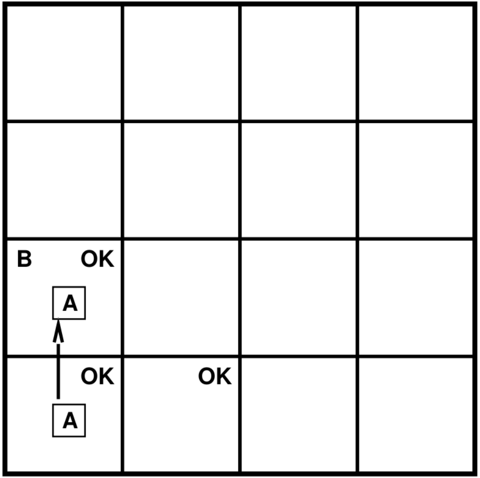
Example: Exploring the Wumpus World

- Agent moves to [2,1]
- perceives a breeze

⟹ Pit in [3,1] or [2,2]

- perceives no stench

⟹ no Wumpus in [3,1], [2,2]



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

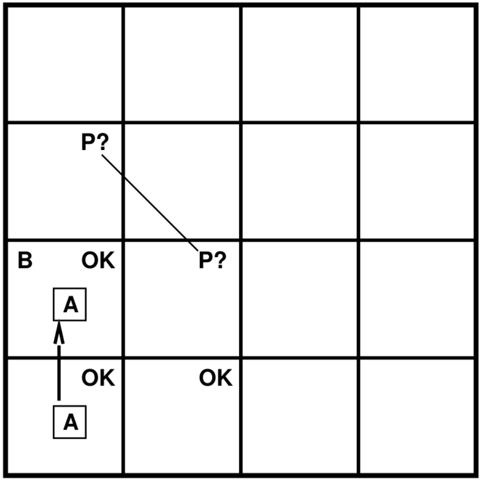
Example: Exploring the Wumpus World

- Agent moves to [2,1]
- perceives a breeze

$\Longrightarrow$ Pit in [3,1] or [2,2]

- perceives no stench

$\Longrightarrow$ no Wumpus in [3,1], [2,2]



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

- Agent moves to [1,1]-[1,2]
- perceives no breeze

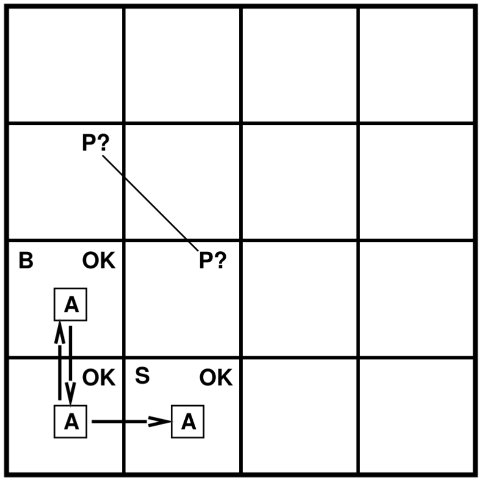
⟹ no Pit in [1,3], [2,2]

⟹ [2,2] OK

⟹ pit in [3,1]

- perceives a stench

⟹ Wumpus in ~~[2,2] or~~ [1,3]!



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

- Agent moves to [1,1]-[1,2]
- perceives no breeze

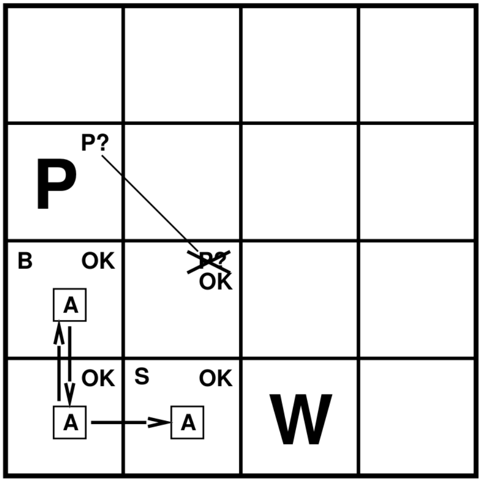
⟹ no Pit in [1,3], [2,2]

⟹ [2,2] OK

⟹ pit in [3,1]

- perceives a stench

⟹ Wumpus in ~~[2,2] or~~ [1,3]!



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

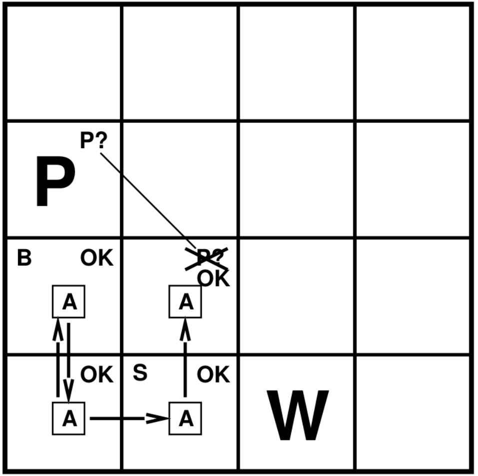
- Agent moves to [2,2]
- perceives no breeze

⟹ no pit in [3,2], [2,3]

- perceives no stench

⟹ no Wumpus in [3,2], [2,3]

⟹ [3,2] and [2,3] OK



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

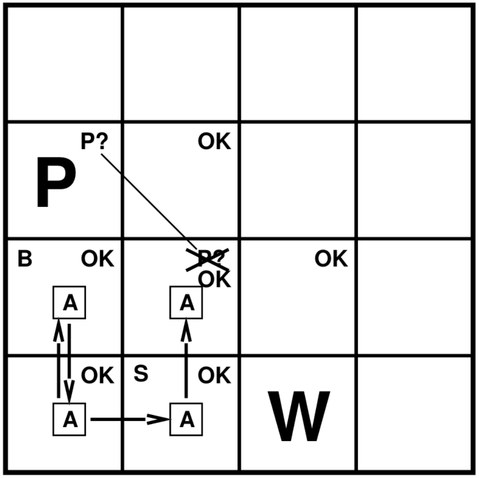
- Agent moves to [2,2]
- perceives no breeze

$\Longrightarrow$ no pit in [3,2], [2,3]

- perceives no stench

$\Longrightarrow$ no Wumpus in [3,2], [2,3]

$\Longrightarrow$ [3,2] and [2,3] OK

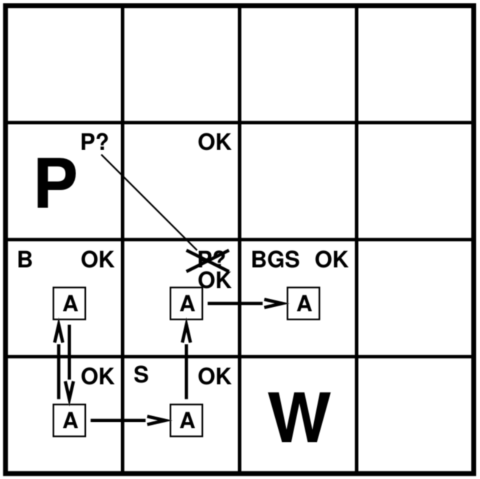


A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

- Agent moves to [2,3]
- perceives a glitter

⟹ bag of gold!



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example 2: Exploring the Wumpus World [see Ch 13]

Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

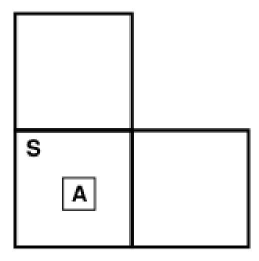
$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there

    $\Longrightarrow$ Wumpus dead

    $\Longrightarrow$ Safe
  - Wumpus wasn't there

    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

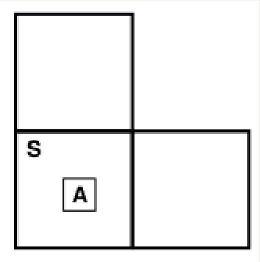
$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there

    $\Longrightarrow$ Wumpus dead

    $\Longrightarrow$ Safe
  - Wumpus wasn't there

    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

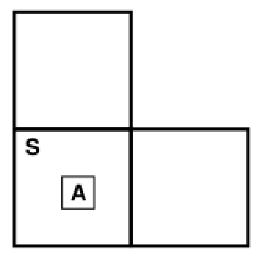
Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there
    $\Longrightarrow$ Wumpus dead
    $\Longrightarrow$ Safe
  - Wumpus wasn't there
    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

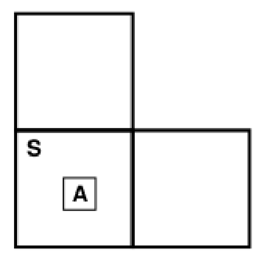
Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there
    $\Longrightarrow$ Wumpus dead
    $\Longrightarrow$ Safe
  - Wumpus wasn't there
    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

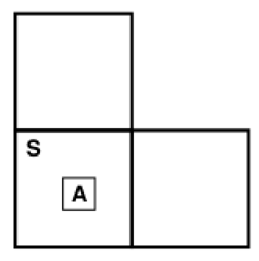
Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there
    $\Longrightarrow$ Wumpus dead
    $\Longrightarrow$ Safe
  - Wumpus wasn't there
    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

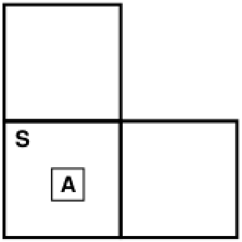
Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there
    $\Longrightarrow$ Wumpus dead
    $\Longrightarrow$ Safe
  - Wumpus wasn't there
    $\Longrightarrow$ Safe



Example 2: Exploring the Wumpus World [see Ch 13]

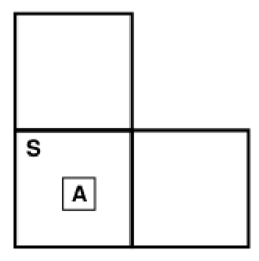
Alternative scenario: apply coercion

- Feel stench in [1,1]

$\Longrightarrow$ Wumpus [1,2] or [2,1]

$\Longrightarrow$ Cannot move

- Apply coercion: shoot ahead
  - Wumpus was there
    $\Longrightarrow$ Wumpus dead
    $\Longrightarrow$ Safe
  - Wumpus wasn't there
    $\Longrightarrow$ Safe



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

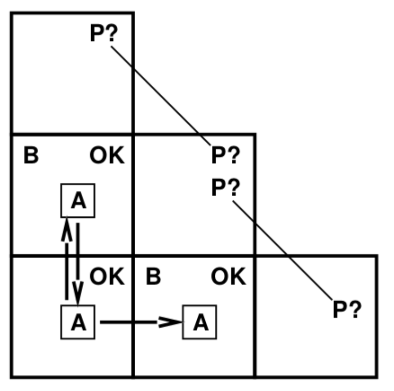
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

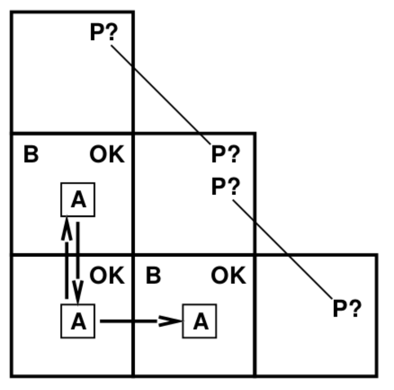
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

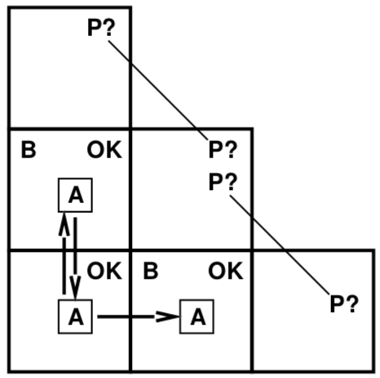
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

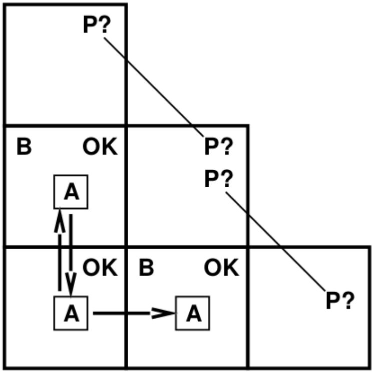
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

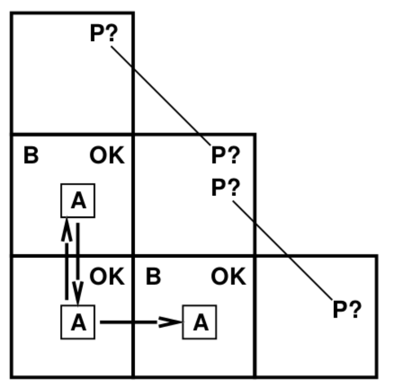
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Example 3: Exploring the Wumpus World [see Ch 13]

Alternative scenario: probabilistic solution (hints)

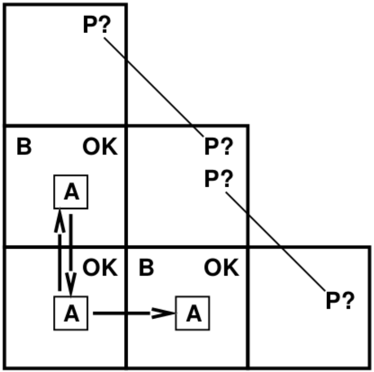
- Feel breeze in [1,2] and [2,1]

$\Longrightarrow$ pit in [1,3] or [2,2] or [3,1]

$\Longrightarrow$ no safe action

- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
  $P(pit \in [2,2]) = 0.86$
  $P(pit \in [1,3]) = 0.31$
  $P(pit \in [3,1]) = 0.31$

$\Longrightarrow$ better choose [1,3] or [3,1]



Outline

Propositional Logic Agents

- Kind of Logic agents
- Language: propositional logic
  - represent KB as set of propositional formulas
  - percepts and actions are (collections of) propositional atoms
  - in practice: sets of clauses
- Perform propositional logic inference
  - model checking, entailment
  - in practice: incremental calls to a SAT solver

Propositional Logic Agents

- Kind of Logic agents
- Language: propositional logic
  - represent KB as set of propositional formulas
  - percepts and actions are (collections of) propositional atoms
  - in practice: sets of clauses
- Perform propositional logic inference
  - model checking, entailment
  - in practice: incremental calls to a SAT solver

Propositional Logic Agents

- Kind of Logic agents
- Language: propositional logic
  - represent KB as set of propositional formulas
  - percepts and actions are (collections of) propositional atoms
  - in practice: sets of clauses
- Perform propositional logic inference
  - model checking, entailment
  - in practice: incremental calls to a SAT solver

Propositional Logic Agents

- Kind of Logic agents
- Language: propositional logic
  - represent KB as set of propositional formulas
  - percepts and actions are (collections of) propositional atoms
  - in practice: sets of clauses
- Perform propositional logic inference
  - model checking, entailment
  - in practice: incremental calls to a SAT solver

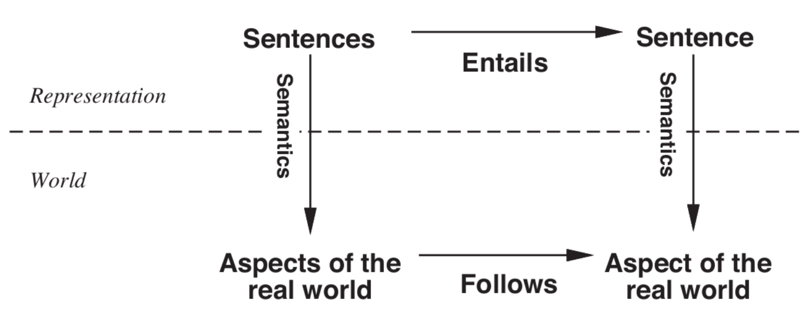
Representation vs. World

Reasoning process (propositional entailment) sound

$\Longrightarrow$ if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

- sentences are configurations of the agent
- reasoning constructs new configurations from old ones

$\Longrightarrow$ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent

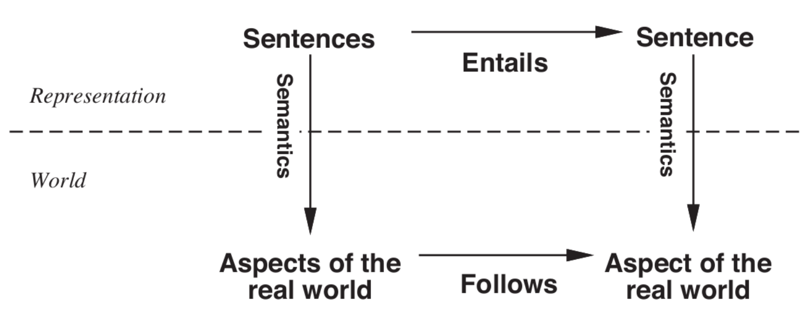


(© S. Russell & P. Norwig, AIMA)

Representation vs. World

Reasoning process (propositional entailment) sound
$\Longrightarrow$ if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

- sentences are configurations of the agent
- reasoning constructs new configurations from old ones
  $\Longrightarrow$ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



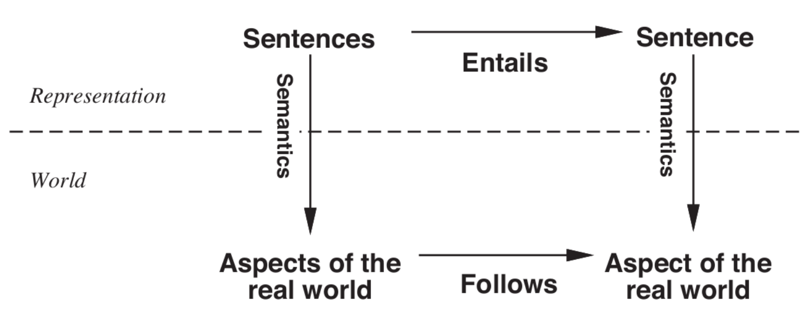
(© S. Russell & P. Norwig, AIMA)

Representation vs. World

Reasoning process (propositional entailment) sound
$\Longrightarrow$ if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

- sentences are configurations of the agent
- reasoning constructs new configurations from old ones

  $\Longrightarrow$ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



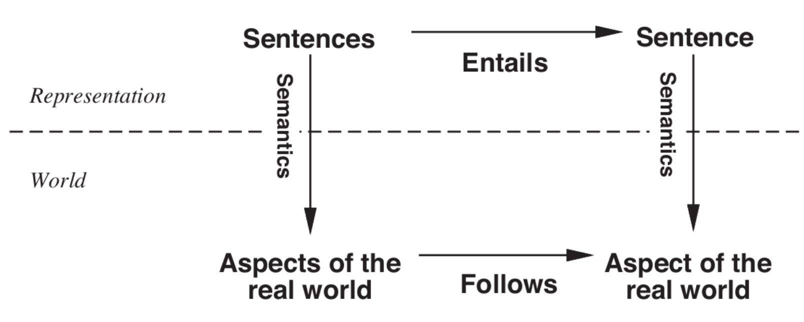
(© S. Russell & P. Norwig, AIMA)

Representation vs. World

Reasoning process (propositional entailment) sound
$\Longrightarrow$ if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

- sentences are configurations of the agent
- reasoning constructs new configurations from old ones

  $\Longrightarrow$ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



(© S. Russell & P. Norwig, AIMA)

Reasoning as Entailment

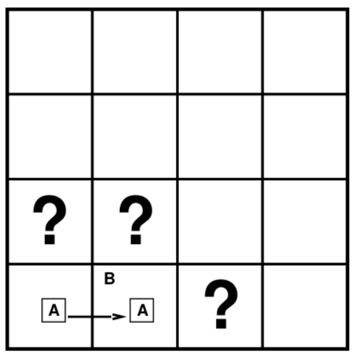
Scenario in Wumpus World

Consider pits (and breezes) only:

- initial: $\neg P_{[1,1]}$
- after detecting nothing in [1,1]: $\neg B_{[1,1]}$
- move to [2,1], detect breeze: $B_{[2,1]}$

Q: are there pits in [1,2], [2,1], [3,1]?

- 3 variables: $P_{[1,2]}, P_{[2,1]}, P_{[3,1]}$, $\Longrightarrow$ 8 possible models



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Reasoning as Entailment

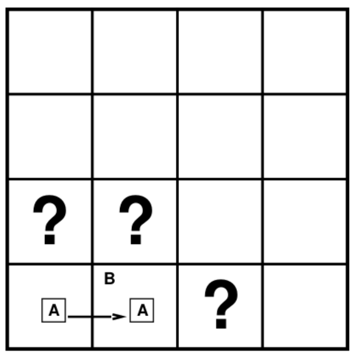
Scenario in Wumpus World

Consider pits (and breezes) only:

- initial: $\neg P_{[1,1]}$
- after detecting nothing in [1,1]: $\neg B_{[1,1]}$
- move to [2,1], detect breeze: $B_{[2,1]}$

Q: are there pits in [1,2], [2,1], [3,1]?

- 3 variables: $P_{[1,2]}, P_{[2,1]}, P_{[3,1]}$, $\Longrightarrow$ 8 possible models



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Reasoning as Entailment

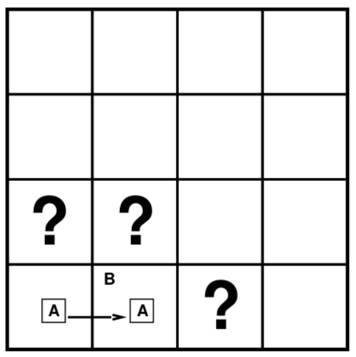
Scenario in Wumpus World

Consider pits (and breezes) only:

- initial: $\neg P_{[1,1]}$
- after detecting nothing in [1,1]: $\neg B_{[1,1]}$
- move to [2,1], detect breeze: $B_{[2,1]}$

Q: are there pits in [1,2], [2,1], [3,1]?

- 3 variables: $P_{[1,2]}, P_{[2,1]}, P_{[3,1]}$, $\Longrightarrow$ 8 possible models



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Reasoning as Entailment

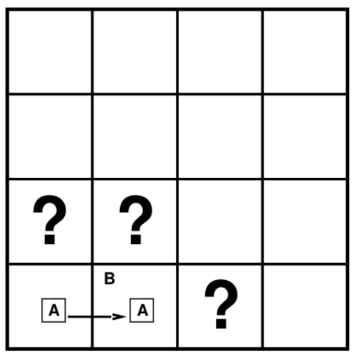
Scenario in Wumpus World

Consider pits (and breezes) only:

- initial: $\neg P_{[1,1]}$
- after detecting nothing in [1,1]: $\neg B_{[1,1]}$
- move to [2,1], detect breeze: $B_{[2,1]}$

Q: are there pits in [1,2], [2,1], [3,1]?

- 3 variables: $P_{[1,2]}, P_{[2,1]}, P_{[3,1]}$, $\Longrightarrow$ 8 possible models



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Reasoning as Entailment

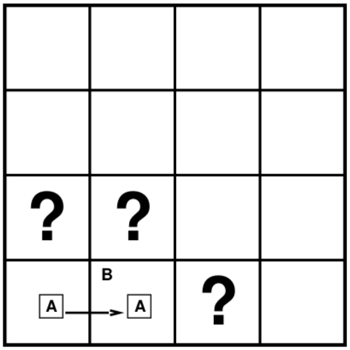
Scenario in Wumpus World

Consider pits (and breezes) only:

- initial: $\neg P_{[1,1]}$
- after detecting nothing in [1,1]: $\neg B_{[1,1]}$
- move to [2,1], detect breeze: $B_{[2,1]}$

Q: are there pits in [1,2], [2,1], [3,1]?

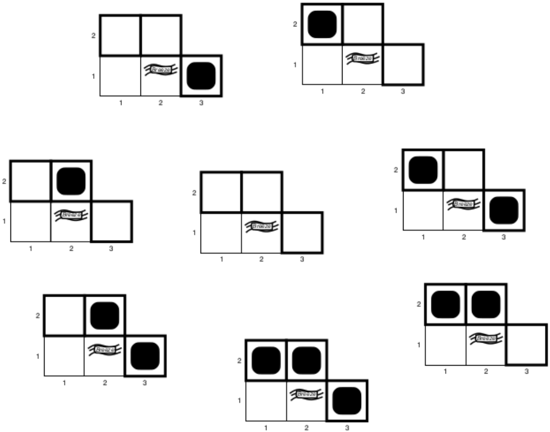
- 3 variables: $P_{[1,2]}, P_{[2,1]}, P_{[3,1]}$, $\Longrightarrow$ 8 possible models



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Reasoning as Entailment [cont.]

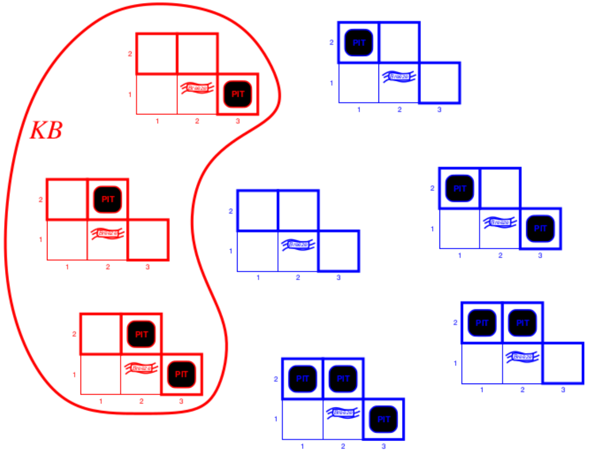
8 possible models



(© S. Russell & P. Norwig, AIMA)

Reasoning as Entailment [cont.]

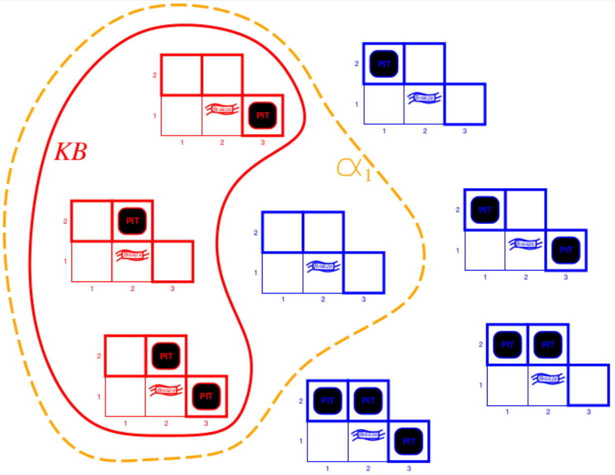
KB: Wumpus World rules + observations $\Longrightarrow$ 3 models



(© S. Russell & P. Norwig, AIMA)

Reasoning as Entailment [cont.]

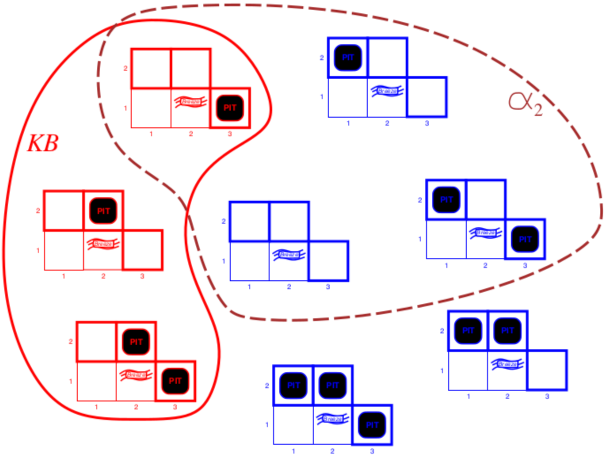
Query α_1 : $\neg P_{[1,2]} \Longrightarrow KB \models \alpha_1$ (i.e $M(KB) \subseteq M(\alpha_1)$)



(© S. Russell & P. Norwig, AIMA)

Reasoning as Entailment [cont.]

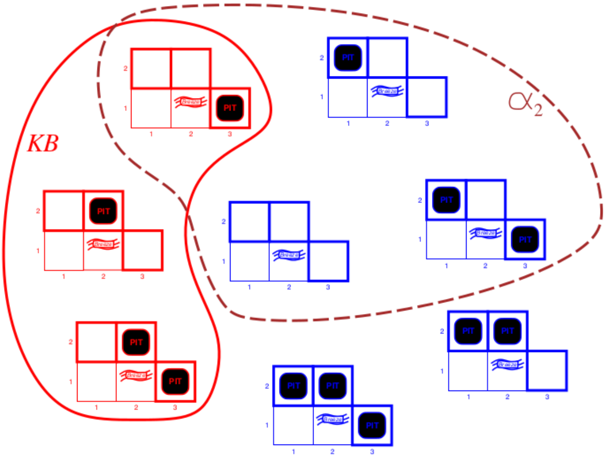
Query $\alpha_2 : \neg P_{[2,2]} \implies KB \not\models \alpha_2$ (i.e $M(KB) \not\subseteq M(\alpha_2)$)



(© S. Russell & P. Norwig, AIMA)

Reasoning as Entailment [cont.]

In practice: $DPLL(CNF(KB \land \neg\alpha_2)) = \text{sat}$

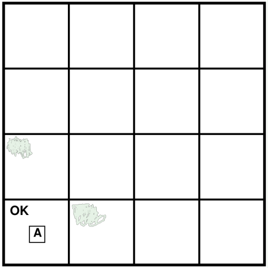


(© S. Russell & P. Norwig, AIMA)

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i, j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours,
  $\forall i, j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[j,i]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in $[1,1]$ no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



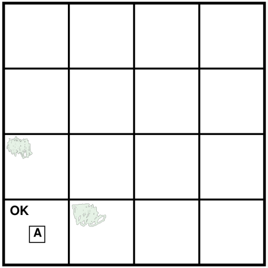
(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i, j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i, j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours, $\forall i, j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[j,i]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in $[1, 1]$ no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



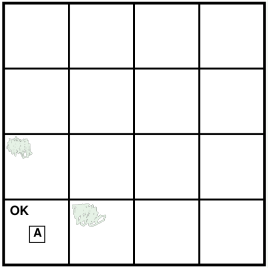
(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i, j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i, j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours, $\forall i, j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[i,j]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in $[1,1]$ no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



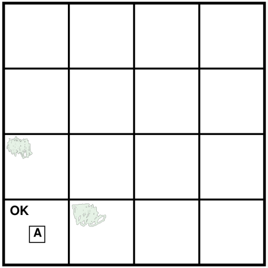
(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i, j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i,j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours, $\forall i,j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[j,i]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in [1, 1] no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



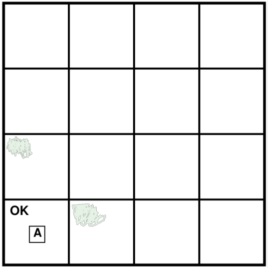
(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i,j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i, j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours, $\forall i, j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[j,i]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in [1, 1] no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



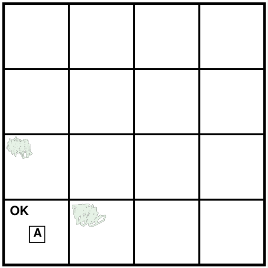
(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i, j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas:

- breeze iff pit in neighbours, $\forall i,j \in [1..4]$
  $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours, $\forall i,j \in [1..4]$
  $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- safe iff no Wumpus and no pit there
  $OK_{[i,j]} \leftrightarrow (\neg W_{[i,j]} \wedge \neg P_{[j,i]})$
- glitter iff pile of gold there
  $G_{[i,j]} \leftrightarrow BGS_{[i,j]}$
- in [1, 1] no Wumpus and no pit $\Longrightarrow$ safe
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$



(implicit: $P_{[i,j]}, W_{[i,j]}, P_{[i,j]}$ false if $i,j \notin [1..4]$)

A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: bag of gold

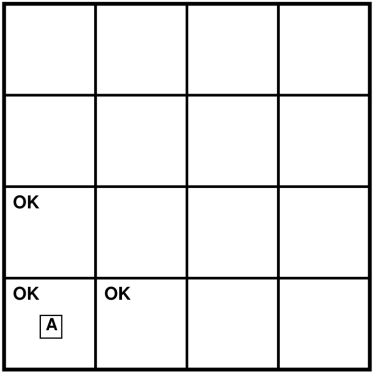
Example: Exploring the Wumpus World

- KB initially contains:
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$
  $B_{[1,1]} \leftrightarrow (P_{[1,2]} \lor P_{[2,1]})$
  $S_{[1,1]} \leftrightarrow (W_{[1,2]} \lor W_{[2,1]})$
  $OK_{[1,2]} \leftrightarrow (\neg W_{[1,2]} \land \neg P_{[2,1]})$
  $OK_{[2,1]} \leftrightarrow (\neg W_{[2,1]} \land \neg P_{[1,2]})$
  ...
- Agent is initially in 1,1
- Percepts (no stench, no breeze):
  $\neg S_{[1,1]}, \neg B_{[1,1]}$

$\Longrightarrow$ $\neg W_{[1,2]}, \neg W_{[2,1]}, \neg P_{[1,2]}, \neg P_{[2,1]}$

$\Longrightarrow$ $OK_{[1,2]}, OK_{[2,1]}$ ([1,2]&[2,1] OK)

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

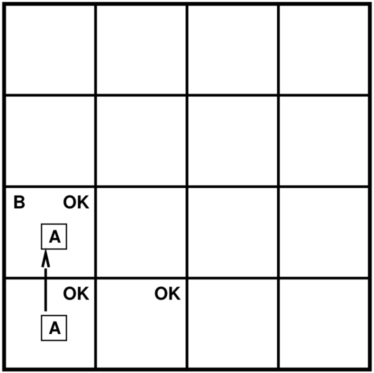
- KB initially contains:
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$
  $B_{[2,1]} \leftrightarrow (P_{[1,1]} \lor P_{[2,2]} \lor P_{[3,1]})$
  $S_{[2,1]} \leftrightarrow (W_{[1,1]} \lor W_{[2,2]} \lor W_{[3,1]})$
- Agent moves to [2,1]
- perceives a breeze: $B_{[2,1]}$

$\Longrightarrow$ $(P_{[3,1]} \lor P_{[2,2]})$ (pit in [3,1] or [2,2])

- perceives no stench $\neg S_{[2,1]}$

$\Longrightarrow$ $\neg W_{[3,1]}, \neg W_{[2,2]}$
(no Wumpus in [3,1], [2,2])

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

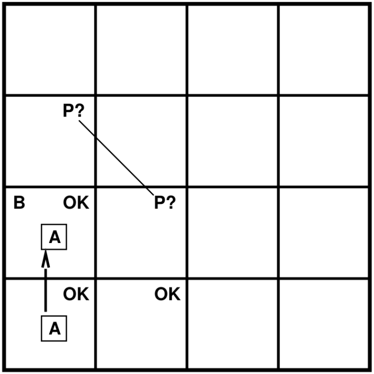
- KB initially contains:
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$
  $B_{[2,1]} \leftrightarrow (P_{[1,1]} \vee P_{[2,2]} \vee P_{[3,1]})$
  $S_{[2,1]} \leftrightarrow (W_{[1,1]} \vee W_{[2,2]} \vee W_{[3,1]})$
- Agent moves to [2,1]
- perceives a breeze: $B_{[2,1]}$

$\Longrightarrow$ $(P_{[3,1]} \vee P_{[2,2]})$ (pit in [3,1] or [2,2])

- perceives no stench $\neg S_{[2,1]}$

$\Longrightarrow$ $\neg W_{[3,1]}, \neg W_{[2,2]}$
(no Wumpus in [3,1], [2,2])

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

- KB initially contains:
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$
  $(P_{[3,1]} \vee P_{[2,2]}), \neg W_{[3,1]}, \neg W_{[2,2]}$
  $B_{[1,2]} \leftrightarrow (P_{[1,1]} \vee P_{[2,2]} \vee P_{[1,3]})$
  $S_{[1,2]} \leftrightarrow (W_{[1,1]} \vee W_{[2,2]} \vee W_{[1,3]})$
  $OK_{[2,2]} \leftrightarrow (\neg W_{[2,2]} \wedge \neg P_{[2,2]})$
- Agent moves to [1,1]-[1,2]
- perceives no breeze: $\neg B_{[1,2]}$

$\Longrightarrow$ $\neg P_{[2,2]}, \neg P_{[1,3]}$ (no pit in [2,2], [1,3])

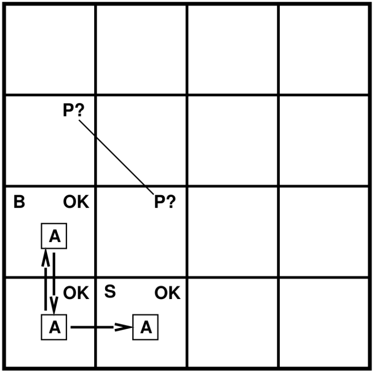
$\Longrightarrow$ $P_{[3,1]}$ (pit in [3,1])

- perceives a stench: $S_{[1,2]}$

$\Longrightarrow$ $W_{[1,3]}$ (Wumpus in [1,3]!)

$\Longrightarrow$ $OK_{[2,2]}$ ([2,2] OK)

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

- KB initially contains:
  $\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$
  $(P_{[3,1]} \vee P_{[2,2]}), \neg W_{[3,1]}, \neg W_{[2,2]}$
  $B_{[1,2]} \leftrightarrow (P_{[1,1]} \vee P_{[2,2]} \vee P_{[1,3]})$
  $S_{[1,2]} \leftrightarrow (W_{[1,1]} \vee W_{[2,2]} \vee W_{[1,3]})$
  $OK_{[2,2]} \leftrightarrow (\neg W_{[2,2]} \wedge \neg P_{[2,2]})$
- Agent moves to [1,1]-[1,2]
- perceives no breeze: $\neg B_{[1,2]}$

$\Longrightarrow$ $\neg P_{[2,2]}, \neg P_{[1,3]}$ (no pit in [2,2], [1,3])

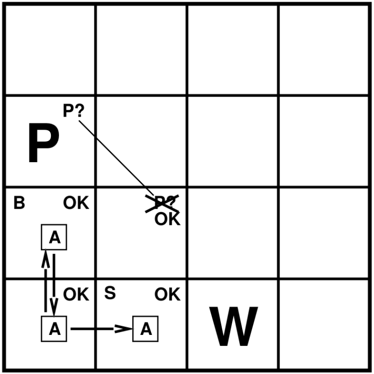
$\Longrightarrow$ $P_{[3,1]}$ (pit in [3,1])

- perceives a stench: $S_{[1,2]}$

$\Longrightarrow$ $W_{[1,3]}$ (Wumpus in [1,3]!)

$\Longrightarrow$ $OK_{[2,2]}$ ([2,2] OK)

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

- KB initially contains:

  $B_{[2,2]} \leftrightarrow (P_{[2,1]} \vee P_{[3,2]} \vee P_{[2,3]} \vee P_{[1,2]})$
  $S_{[2,2]} \leftrightarrow (W_{[2,1]} \vee W_{[3,2]} \vee W_{[2,3]} \vee W_{[1,2]})$
  $OK_{[3,2]} \leftrightarrow (\neg W_{[3,2]} \wedge \neg P_{[3,2]})$
  $OK_{[2,3]} \leftrightarrow (\neg W_{[2,3]} \wedge \neg P_{[2,3]})$
- Agent moves to [2,2]
- perceives no breeze: $\neg B_{[2,2]}$

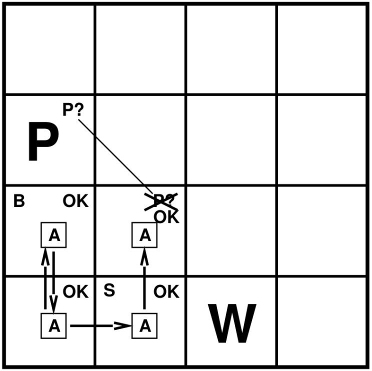
$\Longrightarrow$ $\neg P_{[3,2]}, \neg P_{[3,2]}$ (no pit in [3,2], [2,3])

- perceives no stench: $\neg S_{[2,2]}$

$\Longrightarrow$ $\neg W_{[3,2]}, \neg W_{[3,2]}$ (no Wumpus in [3,2], [2,3])

$\Longrightarrow$ $OK_{[3,2]}, OK_{[2,3]}$, ([3,2] and [2,3] OK)

- Add all them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

- KB initially contains:
  $B_{[2,2]} \leftrightarrow (P_{[2,1]} \vee P_{[3,2]} \vee P_{[2,3]} \vee P_{[1,2]})$
  $S_{[2,2]} \leftrightarrow (W_{[2,1]} \vee W_{[3,2]} \vee W_{[2,3]} \vee W_{[1,2]})$
  $OK_{[3,2]} \leftrightarrow (\neg W_{[3,2]} \wedge \neg P_{[3,2]})$
  $OK_{[2,3]} \leftrightarrow (\neg W_{[2,3]} \wedge \neg P_{[2,3]})$
- Agent moves to [2,2]
- perceives no breeze: $\neg B_{[2,2]}$

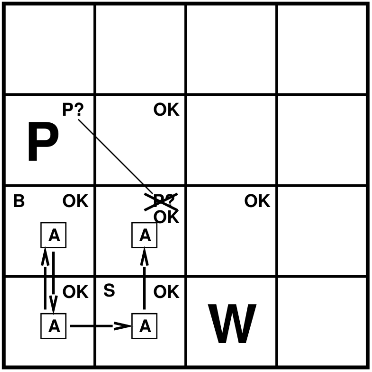
$\Longrightarrow$ $\neg P_{[3,2]}, \neg P_{[3,2]}$ (no pit in [3,2], [2,3])

- perceives no stench: $\neg S_{[2,2]}$

$\Longrightarrow$ $\neg W_{[3,2]}, \neg W_{[3,2]}$ (no Wumpus in [3,2], [2,3])

$\Longrightarrow$ $OK_{[3,2]}, OK_{[2,3]}$, ([3,2] and [2,3] OK)

- Add all them to KB



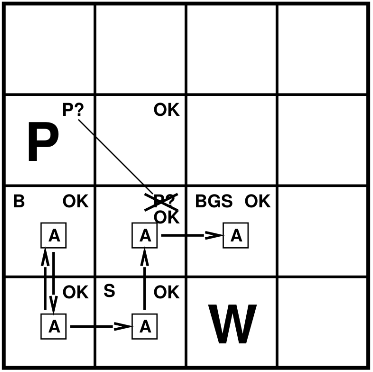
A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Example: Exploring the Wumpus World

- KB initially contains: $G_{[2,3]} \leftrightarrow BGS_{[2,3]}$
- Agent moves to [2,3]
- perceives a glitter: $G_{[2,3]}$

$\Longrightarrow$ $BGS_{[2,3]}$ (bag of gold!)

- Add it them to KB



A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: pit; BGS: glitter, bag of gold

Exercise

Consider the previous example.

1. Convert all formulas from KB into CNF
2. Execute all steps in the example as resolution calls
3. Execute all steps in the example as DPLL calls

Exercise

Consider the previous example.

1. Convert all formulas from KB into CNF
2. Execute all steps in the example as resolution calls
3. Execute all steps in the example as DPLL calls

Exercise

Consider the previous example.

1. Convert all formulas from KB into CNF
2. Execute all steps in the example as resolution calls
3. Execute all steps in the example as DPLL calls

Exercise

Consider the previous example.

1. Convert all formulas from KB into CNF
2. Execute all steps in the example as resolution calls
3. Execute all steps in the example as DPLL calls