

Fundamentals of Artificial Intelligence

Chapter 07: Logical Agents

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it

https://disi.unitn.it/rseba/DIDATTICA/fai_2025/

Teaching assistant:

Paolo Morettin, paolo.morettin@unitn.it, <https://paolomorettin.github.io/>

M.S. Course “Artificial Intelligence Systems”, academic year 2024-2025

Last update: Wednesday 10th September, 2025, 16:22

Copyright notice: Most examples and images displayed in the slides of this course are taken from [\[Russell & Norvig, “Artificial Intelligence, a Modern Approach”, 3rd ed., Pearson\]](#), including explicitly figures from the above-mentioned book, so that their copyright is detained by the authors. A few other material (text, figures, examples) is authored by (in alphabetical order): [Pieter Abbeel](#), [Bonnie J. Dorr](#), [Anca Dragan](#), [Dan Klein](#), [Nikita Kitaev](#), [Tom Lenaerts](#), [Michela Milano](#), [Dana Nau](#), [Maria Simi](#), who detain its copyright. These slides cannot be displayed in public without the permission of the author.

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

Propositional Logic (aka Boolean Logic)



Basic Definitions and Notation

- **Propositional formula** (aka **Boolean formula** or **sentence**)

- $\top, \perp$ are formulas
- a **propositional atom** $A_1, A_2, A_3, \dots$ 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.

- Ex: $\varphi \stackrel{\text{def}}{=} (\neg(A_1 \rightarrow A_2)) \wedge (A_3 \leftrightarrow (\neg A_1 \oplus (A_2 \vee \neg A_4)))$

- **Atoms**(φ): the set $\{A_1, \dots, A_N\}$ 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., $(A_1 \vee \neg A_2 \vee A_3 \vee \dots)$)

- **Cube**: a conjunction of literals $\bigwedge_j l_j$ (e.g., $(A_1 \wedge \neg A_2 \wedge A_3 \wedge \dots)$)

Semantics of Boolean operators

Truth Table

α	β	$\neg\alpha$	$\alpha\wedge\beta$	$\alpha\vee\beta$	$\alpha\rightarrow\beta$	$\alpha\leftarrow\beta$	$\alpha\leftrightarrow\beta$	$\alpha\oplus\beta$
$\perp$	$\perp$	T	$\perp$	$\perp$	T	T	T	$\perp$
$\perp$	T	T	$\perp$	T	T	$\perp$	$\perp$	T
T	$\perp$	$\perp$	$\perp$	T	$\perp$	T	$\perp$	T
T	T	$\perp$	T	T	T	T	T	$\perp$

English Meaning of Boolean Operators

English	Logic
A and B	$A \wedge B$
A if B A when B A whenever B	$A \leftarrow B$
if A, then B A implies B A forces B A requires B	$A \rightarrow B$
A precisely when B A if and only if B	$A \leftrightarrow B$
A or B (or both) A unless B	$A \vee B$ (logical or)
either A or B (but not both)	$A \oplus B$ (exclusive or)

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 the 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 the 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 the 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 the 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)

Properties Boolean Operators

- $\wedge$, $\vee$, $\leftrightarrow$ and $\oplus$ are commutative:

$$(\alpha \wedge \beta) \iff (\beta \wedge \alpha)$$

$$(\alpha \vee \beta) \iff (\beta \vee \alpha)$$

$$(\alpha \leftrightarrow \beta) \iff (\beta \leftrightarrow \alpha)$$

$$(\alpha \oplus \beta) \iff (\beta \oplus \alpha)$$

- $\wedge$, $\vee$, $\leftrightarrow$ and $\oplus$ are associative:

$$((\alpha \wedge \beta) \wedge \gamma) \iff (\alpha \wedge (\beta \wedge \gamma)) \iff (\alpha \wedge \beta \wedge \gamma)$$

$$((\alpha \vee \beta) \vee \gamma) \iff (\alpha \vee (\beta \vee \gamma)) \iff (\alpha \vee \beta \vee \gamma)$$

$$((\alpha \leftrightarrow \beta) \leftrightarrow \gamma) \iff (\alpha \leftrightarrow (\beta \leftrightarrow \gamma)) \iff (\alpha \leftrightarrow \beta \leftrightarrow \gamma)$$

$$((\alpha \oplus \beta) \oplus \gamma) \iff (\alpha \oplus (\beta \oplus \gamma)) \iff (\alpha \oplus \beta \oplus \gamma)$$

- $\rightarrow$, $\leftarrow$ are neither commutative nor associative:

$$(\alpha \rightarrow \beta) \not\iff (\beta \rightarrow \alpha)$$

$$((\alpha \rightarrow \beta) \rightarrow \gamma) \not\iff (\alpha \rightarrow (\beta \rightarrow \gamma))$$

Equivalences with Boolean Operators

$\neg\neg\alpha$	$\iff$	α
$(\alpha \vee \beta)$	$\iff$	$\neg(\neg\alpha \wedge \neg\beta)$
$\neg(\alpha \vee \beta)$	$\iff$	$(\neg\alpha \wedge \neg\beta)$
$(\alpha \wedge \beta)$	$\iff$	$\neg(\neg\alpha \vee \neg\beta)$
$\neg(\alpha \wedge \beta)$	$\iff$	$(\neg\alpha \vee \neg\beta)$
$(\alpha \rightarrow \beta)$	$\iff$	$(\neg\alpha \vee \beta)$
$\neg(\alpha \rightarrow \beta)$	$\iff$	$(\alpha \wedge \neg\beta)$
$(\alpha \leftarrow \beta)$	$\iff$	$(\alpha \vee \neg\beta)$
$\neg(\alpha \leftarrow \beta)$	$\iff$	$(\neg\alpha \wedge \beta)$
$(\alpha \leftrightarrow \beta)$	$\iff$	$((\alpha \rightarrow \beta) \wedge (\alpha \leftarrow \beta))$
	$\iff$	$((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta))$
$\neg(\alpha \leftrightarrow \beta)$	$\iff$	$(\neg\alpha \leftrightarrow \beta)$
	$\iff$	$(\alpha \leftrightarrow \neg\beta)$
	$\iff$	$((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta))$
$(\alpha \oplus \beta)$	$\iff$	$\neg(\alpha \leftrightarrow \beta)$

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

Equivalences with Boolean Operators

$\neg\neg\alpha$	$\iff$	α
$(\alpha \vee \beta)$	$\iff$	$\neg(\neg\alpha \wedge \neg\beta)$
$\neg(\alpha \vee \beta)$	$\iff$	$(\neg\alpha \wedge \neg\beta)$
$(\alpha \wedge \beta)$	$\iff$	$\neg(\neg\alpha \vee \neg\beta)$
$\neg(\alpha \wedge \beta)$	$\iff$	$(\neg\alpha \vee \neg\beta)$
$(\alpha \rightarrow \beta)$	$\iff$	$(\neg\alpha \vee \beta)$
$\neg(\alpha \rightarrow \beta)$	$\iff$	$(\alpha \wedge \neg\beta)$
$(\alpha \leftarrow \beta)$	$\iff$	$(\alpha \vee \neg\beta)$
$\neg(\alpha \leftarrow \beta)$	$\iff$	$(\neg\alpha \wedge \beta)$
$(\alpha \leftrightarrow \beta)$	$\iff$	$((\alpha \rightarrow \beta) \wedge (\alpha \leftarrow \beta))$
	$\iff$	$((\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta))$
$\neg(\alpha \leftrightarrow \beta)$	$\iff$	$(\neg\alpha \leftrightarrow \beta)$
	$\iff$	$(\alpha \leftrightarrow \neg\beta)$
	$\iff$	$((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta))$
$(\alpha \oplus \beta)$	$\iff$	$\neg(\alpha \leftrightarrow \beta)$

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

Exercises

1 For every pair of formulas $\alpha \iff \beta$ below, show that α and β can be rewritten into each other by applying the syntactic properties of the previous slide

- $(A_1 \wedge A_2) \vee A_3 \iff (A_1 \vee A_3) \wedge (A_2 \vee A_3)$
- $(A_1 \vee A_2) \wedge A_3 \iff (A_1 \wedge A_3) \vee (A_2 \wedge A_3)$
- $A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4)) \iff (A_1 \wedge A_2 \wedge A_3) \rightarrow A_4$
- $A_1 \rightarrow (A_2 \wedge A_3) \iff (A_1 \rightarrow A_2) \wedge (A_1 \rightarrow A_3)$
- $(A_1 \vee A_2) \rightarrow A_3 \iff (A_1 \rightarrow A_3) \wedge (A_2 \rightarrow A_3)$
- $A_1 \oplus A_2 \iff (A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$
- $\neg A_1 \leftrightarrow \neg A_2 \iff A_1 \leftrightarrow A_2$
- $A_1 \leftrightarrow A_2 \leftrightarrow A_3 \iff A_1 \oplus A_2 \oplus A_3$

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

$$\begin{aligned} & (A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4) \\ & \quad \Downarrow \\ & (((A_1 \leftrightarrow A_2) \rightarrow (A_3 \leftrightarrow A_4)) \wedge \\ & ((A_3 \leftrightarrow A_4) \rightarrow (A_1 \leftrightarrow A_2))) \end{aligned}$$

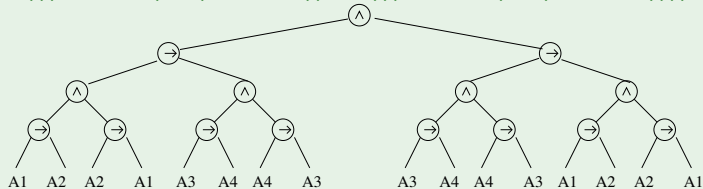
Tree & DAG Representations of Formulas

- 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

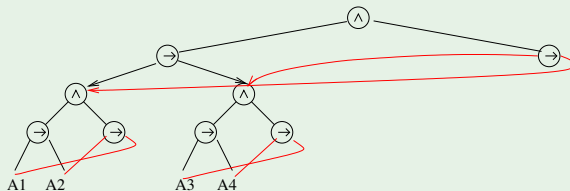
$$\begin{aligned} & (A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4) \\ & \quad \Downarrow \\ & (((A_1 \leftrightarrow A_2) \rightarrow (A_3 \leftrightarrow A_4)) \wedge \\ & \quad ((A_3 \leftrightarrow A_4) \rightarrow (A_1 \leftrightarrow A_2))) \\ & \quad \Downarrow \\ & (((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3))) \wedge \\ & (((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow (((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)))) \end{aligned}$$

Tree & DAG Representations of Formulas: Example

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



Tree Representation



DAG Representation

Basic Definitions and Notation [cont.]

- **Total truth assignment** μ for φ :
 $\mu : \text{Atoms}(\varphi) \mapsto \{\top, \perp\}$.
 - represents a **possible world** or a **possible state of the world**
- **Partial Truth assignment** μ for φ :
 $\mu : \mathcal{A} \mapsto \{\top, \perp\}, \mathcal{A} \subset \text{Atoms}(\varphi)$.
 - 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: $\{\mu(A_1) := \top, \mu(A_2) := \perp\} \implies \{A_1, \neg A_2\}$
 - μ can be represented **as a formula (cube)**:
EX: $\{\mu(A_1) := \top, \mu(A_2) := \perp\} \implies (A_1 \wedge \neg A_2)$

Basic Definitions and Notation [cont.]

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

$$\mu \models A_i \iff \mu(A_i) = \top$$

$$\mu \models \neg\varphi \iff \text{not } \mu \models \varphi$$

$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \text{ and } \mu \models \beta$$

$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \text{ or } \mu \models \beta$$

$$\mu \models \alpha \rightarrow \beta \iff \text{if } \mu \models \alpha, \text{ then } \mu \models \beta$$

$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \text{ iff } \mu \models \beta$$

$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \text{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (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 μ s, $\mu \models \alpha \implies \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is **valid** ($\models \varphi$) iff $\mu \models \varphi$ for all μ s (i.e., $\mu \in M(\varphi)$ for all μ s)

Basic Definitions and Notation [cont.]

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

$$\mu \models A_i \iff \mu(A_i) = \top$$

$$\mu \models \neg\varphi \iff \text{not } \mu \models \varphi$$

$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \text{ and } \mu \models \beta$$

$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \text{ or } \mu \models \beta$$

$$\mu \models \alpha \rightarrow \beta \iff \text{if } \mu \models \alpha, \text{ then } \mu \models \beta$$

$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \text{ iff } \mu \models \beta$$

$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \text{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (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 μ s, $\mu \models \alpha \implies \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is **valid** ($\models \varphi$) iff $\mu \models \varphi$ for all μ s (i.e., $\mu \in M(\varphi)$ for all μ s)

Basic Definitions and Notation [cont.]

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

$$\mu \models A_i \iff \mu(A_i) = \top$$

$$\mu \models \neg\varphi \iff \text{not } \mu \models \varphi$$

$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \text{ and } \mu \models \beta$$

$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \text{ or } \mu \models \beta$$

$$\mu \models \alpha \rightarrow \beta \iff \text{if } \mu \models \alpha, \text{ then } \mu \models \beta$$

$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \text{ iff } \mu \models \beta$$

$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \text{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (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 μ s, $\mu \models \alpha \implies \mu \models \beta$
(i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is **valid** ($\models \varphi$) iff $\mu \models \varphi$ for all μ s (i.e., $\mu \in M(\varphi)$ for all μ s)

Basic Definitions and Notation [cont.]

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

$$\mu \models A_i \iff \mu(A_i) = \top$$

$$\mu \models \neg\varphi \iff \text{not } \mu \models \varphi$$

$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \text{ and } \mu \models \beta$$

$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \text{ or } \mu \models \beta$$

$$\mu \models \alpha \rightarrow \beta \iff \text{if } \mu \models \alpha, \text{ then } \mu \models \beta$$

$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \text{ iff } \mu \models \beta$$

$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \text{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (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 μ s, $\mu \models \alpha \implies \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is **valid** ($\models \varphi$) iff $\mu \models \varphi$ for all μ s (i.e., $\mu \in M(\varphi)$ for all μ s)

Basic Definitions and Notation [cont.]

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

$$\mu \models A_i \iff \mu(A_i) = \top$$

$$\mu \models \neg\varphi \iff \text{not } \mu \models \varphi$$

$$\mu \models \alpha \wedge \beta \iff \mu \models \alpha \text{ and } \mu \models \beta$$

$$\mu \models \alpha \vee \beta \iff \mu \models \alpha \text{ or } \mu \models \beta$$

$$\mu \models \alpha \rightarrow \beta \iff \text{if } \mu \models \alpha, \text{ then } \mu \models \beta$$

$$\mu \models \alpha \leftrightarrow \beta \iff \mu \models \alpha \text{ iff } \mu \models \beta$$

$$\mu \models \alpha \oplus \beta \iff \mu \models \alpha \text{ iff not } \mu \models \beta$$

- $M(\varphi) \stackrel{\text{def}}{=} \{\mu \mid \mu \models \varphi\}$ (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 μ s, $\mu \models \alpha \implies \mu \models \beta$ (i.e., $M(\alpha) \subseteq M(\beta)$)
- φ is **valid** ($\models \varphi$) iff $\mu \models \varphi$ for all μ s (i.e., $\mu \in M(\varphi)$ for all μ s)

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 $\alpha \rightarrow \beta$ 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 $\alpha \rightarrow \beta$ 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 $\alpha \rightarrow \beta$ 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!

Equivalence and Equi-Satisfiability

- α and β are **equivalent** iff, for every μ , $\mu \models \alpha$ iff $\mu \models \beta$
(i.e., if $M(\alpha) = M(\beta)$)
- α and β are **equi-satisfiable** iff exists μ_1 s.t. $\mu_1 \models \alpha$ iff exists μ_2 s.t. $\mu_2 \models \beta$
(i.e., if $M(\alpha) \neq \emptyset$ iff $M(\beta) \neq \emptyset$)
- α, β equivalent
 $\Downarrow \nleftrightarrow$
 α, β equi-satisfiable
- EX: $A_1 \vee A_2$ and $(A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$ are equi-satisfiable, not equivalent.
 $\{\neg A_1, A_2, A_3\} \models (A_1 \vee A_2)$, but $\{\neg A_1, A_2, A_3\} \not\models (A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$
- Typically used when β is the result of applying some transformation T to α : $\beta \stackrel{\text{def}}{=} T(\alpha)$:
 - T is **validity-preserving** [resp. **satisfiability-preserving**] iff
 $T(\alpha)$ and α are equivalent [resp. equi-satisfiable]

Equivalence and Equi-Satisfiability

- α and β are **equivalent** iff, for every μ , $\mu \models \alpha$ iff $\mu \models \beta$
(i.e., if $M(\alpha) = M(\beta)$)
- α and β are **equi-satisfiable** iff exists μ_1 s.t. $\mu_1 \models \alpha$ iff exists μ_2 s.t. $\mu_2 \models \beta$
(i.e., if $M(\alpha) \neq \emptyset$ iff $M(\beta) \neq \emptyset$)
- α, β equivalent
 $\Downarrow \Uparrow$
 α, β equi-satisfiable
- EX: $A_1 \vee A_2$ and $(A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$ are equi-satisfiable, not equivalent.
 $\{\neg A_1, A_2, A_3\} \models (A_1 \vee A_2)$, but $\{\neg A_1, A_2, A_3\} \not\models (A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$
- Typically used when β is the result of applying some transformation T to α : $\beta \stackrel{\text{def}}{=} T(\alpha)$:
 - T is **validity-preserving** [resp. **satisfiability-preserving**] iff
 $T(\alpha)$ and α are equivalent [resp. equi-satisfiable]

Equivalence and Equi-Satisfiability

- α and β are **equivalent** iff, for every μ , $\mu \models \alpha$ iff $\mu \models \beta$
(i.e., if $M(\alpha) = M(\beta)$)
- α and β are **equi-satisfiable** iff exists μ_1 s.t. $\mu_1 \models \alpha$ iff exists μ_2 s.t. $\mu_2 \models \beta$
(i.e., if $M(\alpha) \neq \emptyset$ iff $M(\beta) \neq \emptyset$)
- α, β equivalent
 $\Downarrow \nleftrightarrow$
 α, β equi-satisfiable
- EX: $A_1 \vee A_2$ and $(A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$ are equi-satisfiable, not equivalent.
 $\{\neg A_1, A_2, A_3\} \models (A_1 \vee A_2)$, but $\{\neg A_1, A_2, A_3\} \not\models (A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$
- Typically used when β is the result of applying some transformation T to α : $\beta \stackrel{\text{def}}{=} T(\alpha)$:
 - T is **validity-preserving** [resp. **satisfiability-preserving**] iff
 $T(\alpha)$ and α are equivalent [resp. equi-satisfiable]

Equivalence and Equi-Satisfiability

- α and β are **equivalent** iff, for every μ , $\mu \models \alpha$ iff $\mu \models \beta$
(i.e., if $M(\alpha) = M(\beta)$)
- α and β are **equi-satisfiable** iff exists μ_1 s.t. $\mu_1 \models \alpha$ iff exists μ_2 s.t. $\mu_2 \models \beta$
(i.e., if $M(\alpha) \neq \emptyset$ iff $M(\beta) \neq \emptyset$)
- α, β equivalent
 $\Downarrow \nleftrightarrow$
 α, β equi-satisfiable
- EX: $A_1 \vee A_2$ and $(A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$ are equi-satisfiable, not equivalent.
 $\{\neg A_1, A_2, A_3\} \models (A_1 \vee A_2)$, but $\{\neg A_1, A_2, A_3\} \not\models (A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$
- Typically used when β is the result of applying some transformation T to α : $\beta \stackrel{\text{def}}{=} T(\alpha)$:
 - T is **validity-preserving** [resp. **satisfiability-preserving**] iff
 $T(\alpha)$ and α are equivalent [resp. equi-satisfiable]

Equivalence and Equi-Satisfiability

- α and β are **equivalent** iff, for every μ , $\mu \models \alpha$ iff $\mu \models \beta$
(i.e., if $M(\alpha) = M(\beta)$)
- α and β are **equi-satisfiable** iff exists μ_1 s.t. $\mu_1 \models \alpha$ iff exists μ_2 s.t. $\mu_2 \models \beta$
(i.e., if $M(\alpha) \neq \emptyset$ iff $M(\beta) \neq \emptyset$)
- α, β equivalent
 $\Downarrow \nleftrightarrow$
 α, β equi-satisfiable
- EX: $A_1 \vee A_2$ and $(A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$ are equi-satisfiable, not equivalent.
 $\{\neg A_1, A_2, A_3\} \models (A_1 \vee A_2)$, but $\{\neg A_1, A_2, A_3\} \not\models (A_1 \vee \neg A_3) \wedge (A_3 \vee A_2)$
- Typically used when β is the result of applying some transformation T to α : $\beta \stackrel{\text{def}}{=} T(\alpha)$:
 - T is **validity-preserving** [resp. **satisfiability-preserving**] iff
 $T(\alpha)$ and α are equivalent [resp. equi-satisfiable]

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**

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



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.
 $\implies$ 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:

$$\alpha \rightarrow \beta \implies \neg\alpha \vee \beta$$

$$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$

(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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \wedge \dots \wedge (l_{1N} \vee l_{2N} \vee \dots \vee l_{NN}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee \dots \vee l_{N1}) \wedge \dots \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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: $||CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))|| = ||(l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee \dots \vee l_{N1}) \wedge \dots \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2})|| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \wedge \dots \wedge (l_{1N} \vee l_{2N} \vee \dots \vee l_{NN}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \wedge \dots \wedge (l_{1N} \vee l_{2N} \vee \dots \vee l_{NN}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee \dots \vee l_{N1}) \wedge \dots \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$

- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \wedge \dots \wedge (l_{1N} \vee l_{2N} \vee \dots \vee l_{NN}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$

- $CNF(\varphi)$ 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: $\|CNF(\bigvee_{i=1}^N (l_{i1} \wedge l_{i2}))\| = \| (l_{11} \vee l_{21} \vee \dots \vee l_{N1}) \wedge (l_{12} \vee l_{21} \vee \dots \vee l_{N1}) \wedge \dots \wedge (l_{12} \vee l_{22} \vee \dots \vee l_{N2}) \| = 2^N$

- $Atoms(CNF(\varphi)) = Atoms(\varphi)$
- $CNF(\varphi)$ 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:

$$\varphi \implies \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j))$$

$$\varphi \implies \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j))$$

$$\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$$

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:

$$\varphi \implies \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j))$$

$$\varphi \implies \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j))$$

$$\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$$

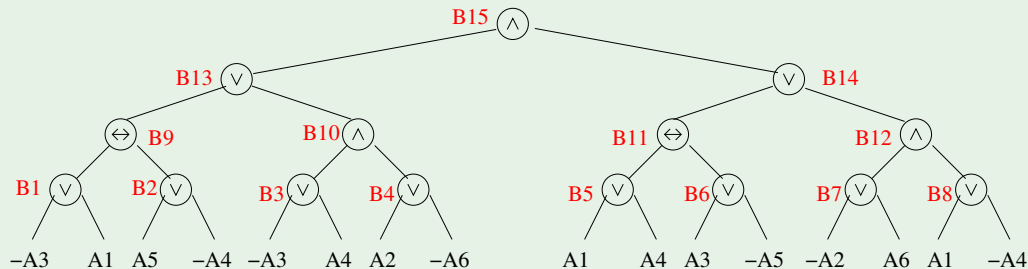
l_i, l_j being literals and B being a “new” variable.

- 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.)

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

Labeling CNF Conversion CNF_{label} – Example



$CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1)) \wedge$

$\dots \wedge$

$CNF(B_8 \leftrightarrow (A_1 \vee \neg A_4)) \wedge$

$CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2)) \wedge$

$\dots \wedge$

$CNF(B_{12} \leftrightarrow (B_7 \wedge B_8)) \wedge$

$CNF(B_{13} \leftrightarrow (B_9 \vee B_{10})) \wedge$

$CNF(B_{14} \leftrightarrow (B_{11} \vee B_{12})) \wedge$

$CNF(B_{15} \leftrightarrow (B_{13} \wedge B_{14})) \wedge$

B_{15}

$(\neg B_1 \vee \neg A_3 \vee A_1) \wedge (B_1 \vee A_3) \wedge (B_1 \vee \neg A_1) \wedge$

$\dots \wedge$

$(\neg B_8 \vee A_1 \vee \neg A_4) \wedge (B_8 \vee \neg A_1) \wedge (B_8 \vee A_4) \wedge$

$(\neg B_9 \vee \neg B_1 \vee B_2) \wedge (\neg B_9 \vee B_1 \vee \neg B_2) \wedge$

$(B_9 \vee B_1 \vee B_2) \wedge (B_9 \vee \neg B_1 \vee \neg B_2) \wedge$

$= \dots \wedge$

$(B_{12} \vee \neg B_7 \vee \neg B_8) \wedge (\neg B_{12} \vee B_7) \wedge (\neg B_{12} \vee B_8) \wedge$

$(\neg B_{13} \vee B_9 \vee B_{10}) \wedge (B_{13} \vee \neg B_9) \wedge (B_{13} \vee \neg B_{10}) \wedge$

$(\neg B_{14} \vee B_{11} \vee B_{12}) \wedge (B_{14} \vee \neg B_{11}) \wedge (B_{14} \vee \neg B_{12}) \wedge$

$(B_{15} \vee \neg B_{13} \vee \neg B_{14}) \wedge (\neg B_{15} \vee B_{13}) \wedge (\neg B_{15} \vee B_{14}) \wedge$

B_{15}

1 Propositional Logic

2 Propositional Reasoning

- Resolution
- DPLL
- Reasoning with Horn Formulas
- Local Search

3 Agents Based on Knowledge Representation & Reasoning

- Knowledge-Based Agents
- Example: the Wumpus World
- Propositional-Logic Agents
- Example: the Wumpus World

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 some knowledge base KB
(aka Model Checking: $M(KB) \subseteq M(\alpha)$)
 - typically $||KB|| \gg ||\alpha||$
 - sometimes KB set of variable implications $(A_1 \wedge \dots \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
 - $KB \models \alpha \implies \text{SAT}(KB \wedge \neg \alpha) = \text{false}$
 - 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 some knowledge base KB (aka Model Checking: $M(KB) \subseteq M(\alpha)$)
 - typically $||KB|| \gg ||\alpha||$
 - sometimes KB set of variable implications $(A_1 \wedge \dots \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
 - $KB \models \alpha \implies \text{SAT}(KB \wedge \neg \alpha) = \text{false}$
 - 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 some knowledge base KB
(aka Model Checking: $M(KB) \subseteq M(\alpha)$)
 - typically $||KB|| \gg ||\alpha||$
 - sometimes KB set of variable implications $(A_1 \wedge \dots \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
 - $KB \models \alpha \implies \text{SAT}(KB \wedge \neg\alpha) = \text{false}$
 - 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 some knowledge base KB
(aka Model Checking: $M(KB) \subseteq M(\alpha)$)
 - typically $||KB|| \gg ||\alpha||$
 - sometimes KB set of variable implications $(A_1 \wedge \dots \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
 - $KB \models \alpha \implies \text{SAT}(KB \wedge \neg \alpha) = \text{false}$
 - 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 some knowledge base KB
(aka Model Checking: $M(KB) \subseteq M(\alpha)$)
 - typically $||KB|| \gg ||\alpha||$
 - sometimes KB set of variable implications $(A_1 \wedge \dots \wedge A_k) \rightarrow B$
- All propositional reasoning tasks reduced to satisfiability (SAT)
 - $KB \models \alpha \implies \text{SAT}(KB \wedge \neg \alpha) = \text{false}$
 - 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)

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

The Resolution Rule

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

$$\begin{array}{c}
 \text{common} \quad \text{resolvent} \quad C' \quad \text{common} \quad \text{resolvent} \quad C'' \\
 (l_1 \vee \dots \vee l_k \vee \underbrace{l}_{\text{resolvent}} \vee \underbrace{l'_{k+1} \vee \dots \vee l'_m}_{C'}) \quad (l_1 \vee \dots \vee l_k \vee \underbrace{\neg l}_{\text{resolvent}} \vee \underbrace{l''_{k+1} \vee \dots \vee l''_n}_{C''}) \\
 \hline
 (\underbrace{l_1 \vee \dots \vee l_k}_{\text{common}} \vee \underbrace{l'_{k+1} \vee \dots \vee l'_m}_{C'} \vee \underbrace{l''_{k+1} \vee \dots \vee l''_n}_{C''})
 \end{array}$$

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

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

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

The Resolution Rule

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

$$\begin{array}{c}
 \text{common} \quad \text{resolvent} \quad C' \quad \text{common} \quad \text{resolvent} \quad C'' \\
 (l_1 \vee \dots \vee l_k \vee \textcolor{red}{l} \vee l'_{k+1} \vee \dots \vee l'_m) \quad (l_1 \vee \dots \vee l_k \vee \textcolor{red}{\neg l} \vee l''_{k+1} \vee \dots \vee l''_n) \\
 \hline
 (l_1 \vee \dots \vee l_k \vee l'_{k+1} \vee \dots \vee l'_m \vee l''_{k+1} \vee \dots \vee l''_n) \\
 \text{common} \quad C' \quad C''
 \end{array}$$

• Ex:
$$\begin{array}{c}
 (A \vee B \vee \textcolor{red}{C} \vee D \vee E) \quad (A \vee B \vee \textcolor{red}{\neg C} \vee F) \\
 \hline
 (A \vee B \vee D \vee E \vee F)
 \end{array}$$

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

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

The Resolution Rule

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

$$\begin{array}{c}
 \text{common} \quad \text{resolvent} \quad C' \quad \text{common} \quad \text{resolvent} \quad C'' \\
 (l_1 \vee \dots \vee l_k \vee \overbrace{l}^{\text{resolvent}} \vee \overbrace{l'_{k+1} \vee \dots \vee l'_m}^{C'}) \quad (l_1 \vee \dots \vee l_k \vee \overbrace{\neg l}^{\text{resolvent}} \vee \overbrace{l''_{k+1} \vee \dots \vee l''_n}^{C''}) \\
 \hline
 (\underbrace{l_1 \vee \dots \vee l_k}_{\text{common}} \vee \underbrace{l'_{k+1} \vee \dots \vee l'_m}_{C'} \vee \underbrace{l''_{k+1} \vee \dots \vee l''_n}_{C''})
 \end{array}$$

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

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

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

Basic Propositional Inference: Resolution

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

⇒ 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

⇒ 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

⇒ 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

⇒ 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

⇒ 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

⇒ 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

⇒ 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

⇒ 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

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

$new \leftarrow \{ \}$

loop do

for each pair of clauses C_i, C_j **in** $clauses$ **do**

$resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)

if $resolvents$ contains the empty clause **then return** *true*

$new \leftarrow new \cup resolvents$

if $new \subseteq clauses$ **then return** *false*

$clauses \leftarrow clauses \cup new$

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Improvements: Subsumption & Unit Propagation

General “set” notation (Γ clause set):

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

- Removal of valid clauses:

$$\frac{\Gamma \wedge (p \vee \neg p \vee C)}{\Gamma}$$

- 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

“Deterministic” rule: applied **before** other “non-deterministic” rules!

Remark

What happens with more than 1 resolvent?

- Common mistake: the following is not a correct application of the resolution rule:

$$\frac{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2)}{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2), (C_1 \vee C_2)}$$

- Rather, a correct application would be:

$$\frac{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2)}{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2), (C_1 \vee l_2 \vee C_2 \vee \neg l_2)}$$

... but $(C_1 \vee l_2 \vee C_2 \vee \neg l_2)$ is valid and should be removed

⇒ no clause is produced

Remark

What happens with more than 1 resolvent?

- Common mistake: the following is not a correct application of the resolution rule:

$$\frac{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2)}{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2), (C_1 \vee C_2)}$$

- Rather, a correct application would be:

$$\frac{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2)}{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2), (C_1 \vee I_2 \vee C_2 \vee \neg I_2)}$$

... but $(C_1 \vee I_2 \vee C_2 \vee \neg I_2)$ is valid and should be removed

⇒ no clause is produced

Remark

What happens with more than 1 resolvent?

- Common mistake: the following is not a correct application of the resolution rule:

$$\frac{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2)}{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2), (C_1 \vee C_2)}$$

- Rather, a correct application would be:

$$\frac{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2)}{\Gamma, (C_1 \vee l_1 \vee l_2), (C_2 \vee \neg l_1 \vee \neg l_2), (C_1 \vee l_2 \vee C_2 \vee \neg l_2)}$$

... but $(C_1 \vee l_2 \vee C_2 \vee \neg l_2)$ is valid and should be removed

⇒ no clause is produced

Remark

What happens with more than 1 resolvent?

- Common mistake: the following is not a correct application of the resolution rule:

$$\frac{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2)}{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2), (C_1 \vee C_2)}$$

- Rather, a correct application would be:

$$\frac{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2)}{\Gamma, (C_1 \vee I_1 \vee I_2), (C_2 \vee \neg I_1 \vee \neg I_2), (C_1 \vee I_2 \vee C_2 \vee \neg I_2)}$$

... but $(C_1 \vee I_2 \vee C_2 \vee \neg I_2)$ is valid and should be removed

$\Rightarrow$ no clause is produced

Resolution: example

Given the following set of propositional clauses Γ :

$(A \vee D \vee \neg F)$
 $(\neg C \vee E)$
 (A)
 $(B \vee E \vee \neg G)$
 $(\neg G)$
 $(\neg E \vee F)$
 $(\neg A \vee \neg B \vee C)$
 (B)
 $(\neg B \vee \neg C \vee D)$
 $(\neg B \vee \neg F \vee G)$

Produce a PL-resolution proof that Γ is unsatisfiable.

Solution:

$[(A), (\neg A \vee \neg B \vee C)] \Rightarrow (\neg B \vee C);$

$[(B), (\neg B \vee C)] \Rightarrow (C);$

$[(C), (\neg C \vee E)] \Rightarrow (E);$

$[(E), (\neg E \vee F)] \Rightarrow (F);$

$[(B), (\neg B \vee \neg F \vee G)] \Rightarrow (\neg F \vee G);$

$[(F), (\neg F \vee G)] \Rightarrow (G);$

$[(\neg G), (G)] \Rightarrow ();$

Hint: resolve always unit clauses first!

Resolution: example

Given the following set of propositional clauses Γ :

$(A \vee D \vee \neg F)$
 $(\neg C \vee E)$
 (A)
 $(B \vee E \vee \neg G)$
 $(\neg G)$
 $(\neg E \vee F)$
 $(\neg A \vee \neg B \vee C)$
 (B)
 $(\neg B \vee \neg C \vee D)$
 $(\neg B \vee \neg F \vee G)$

Produce a PL-resolution proof that Γ is unsatisfiable.

Solution:

$[(A), (\neg A \vee \neg B \vee C)] \implies (\neg B \vee C);$

$[(B), (\neg B \vee C)] \implies (C);$

$[(C), (\neg C \vee E)] \implies (E);$

$[(E), (\neg E \vee F)] \implies (F);$

$[(B), (\neg B \vee \neg F \vee G)] \implies (\neg F \vee G);$

$[(F), (\neg F \vee G)] \implies (G);$

$[(\neg G), (G)] \implies ();$

Hint: resolve always unit clauses first!

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - **DPLL**
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

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*})

(© S. Russell & P. Norwig, AIMA)

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 $\leftarrow$ 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* })

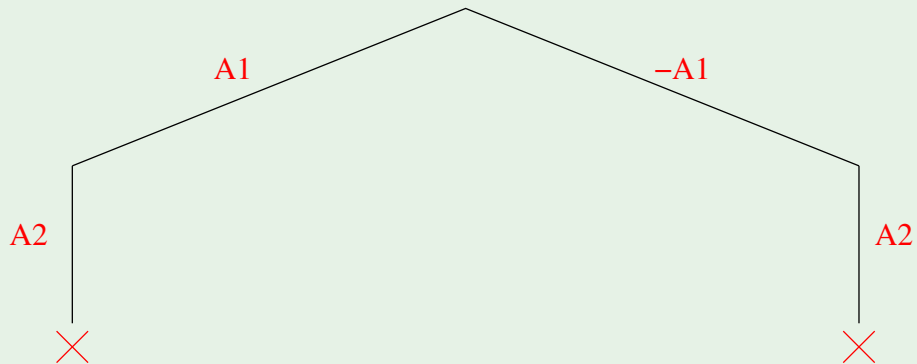
(© S. Russell & P. Norwig, AIMA)

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

DPLL: Example

DPLL search tree

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



DPLL – example

DPLL (without pure-literal rule)

Here “choose-literal” selects variable in alphabetic order, selecting true first.

$$\begin{aligned} & (\neg C) \wedge \\ & (B \vee A \vee C) \wedge \\ & (\neg A \vee D) \wedge \\ & (\neg E \vee \neg A \vee F) \wedge \\ & (\neg E \vee \neg F \vee \neg A) \wedge \\ & (G \vee \neg A \vee E) \wedge \\ & (E \vee \neg G \vee \neg A) \wedge \\ & (A \vee H \vee C) \wedge \\ & (\neg H \vee \neg I \vee A) \wedge \\ & (I \vee L \vee M) \wedge \\ & (\neg L \vee C \vee \neg M) \wedge \\ & (A \vee \neg L \vee M) \wedge \\ & (L \vee N \vee \neg H) \wedge \\ & (I \vee L \vee \neg N) \end{aligned}$$

$\Rightarrow$ UNSAT

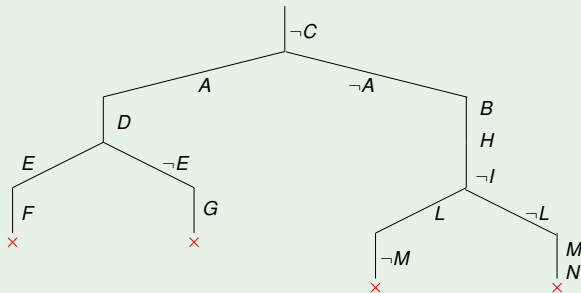
Remark: “choose-literal” selects only variables which still occur in the formula, after simplification. E.g., in the leftmost branch, after assigning $\neg C$, A , D , it does not select B because the clause $(B \vee A \vee C)$ has been simplified into true, and as such is no more part of the formula, so that B does not occur in the formula anymore.

DPLL – example

DPLL (without pure-literal rule)

Here “choose-literal” selects variable in alphabetic order, selecting true first.

$(\neg C) \quad \wedge$
 $(B \vee A \vee C) \wedge$
 $(\neg A \vee D) \wedge$
 $(\neg E \vee \neg A \vee F) \wedge$
 $(\neg E \vee \neg F \vee \neg A) \wedge$
 $(G \vee \neg A \vee E) \wedge$
 $(E \vee \neg G \vee \neg A) \wedge$
 $(A \vee H \vee C) \wedge$
 $(\neg H \vee \neg I \vee A) \wedge$
 $(I \vee L \vee M) \wedge$
 $(\neg L \vee C \vee \neg M) \wedge$
 $(A \vee \neg L \vee M) \wedge$
 $(L \vee N \vee \neg H) \wedge$
 $(I \vee L \vee \neg N)$



$\Rightarrow$ UNSAT

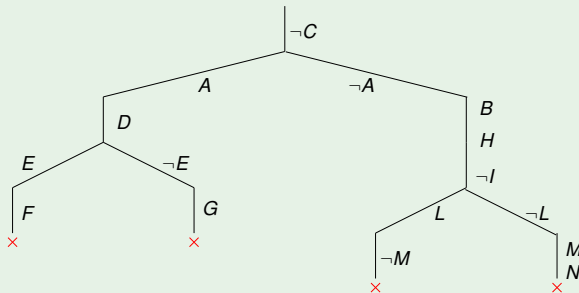
Remark: “choose-literal” selects only variables which still occur in the formula, after simplification. E.g., in the leftmost branch, after assigning $\neg C$, A , D , it does not select B because the clause $(B \vee A \vee C)$ has been simplified into true, and as such is no more part of the formula, so that B does not occur in the formula anymore.

DPLL – example

DPLL (without pure-literal rule)

Here “choose-literal” selects variable in alphabetic order, selecting true first.

$(\neg C) \quad \wedge$
 $(B \vee A \vee C) \wedge$
 $(\neg A \vee D) \wedge$
 $(\neg E \vee \neg A \vee F) \wedge$
 $(\neg E \vee \neg F \vee \neg A) \wedge$
 $(G \vee \neg A \vee E) \wedge$
 $(E \vee \neg G \vee \neg A) \wedge$
 $(A \vee H \vee C) \wedge$
 $(\neg H \vee \neg I \vee A) \wedge$
 $(I \vee L \vee M) \wedge$
 $(\neg L \vee C \vee \neg M) \wedge$
 $(A \vee \neg L \vee M) \wedge$
 $(L \vee N \vee \neg H) \wedge$
 $(I \vee L \vee \neg N)$



$\Rightarrow$ UNSAT

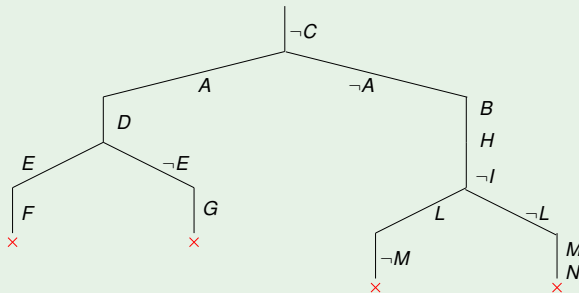
Remark: “choose-literal” selects only variables which still occur in the formula, after simplification. E.g., in the leftmost branch, after assigning $\neg C$, A , D , it does not select B because the clause $(B \vee A \vee C)$ has been simplified into true, and as such is no more part of the formula, so that B does not occur in the formula anymore.

DPLL – example

DPLL (without pure-literal rule)

Here “choose-literal” selects variable in alphabetic order, selecting true first.

$(\neg C) \wedge$
 $(B \vee A \vee C) \wedge$
 $(\neg A \vee D) \wedge$
 $(\neg E \vee \neg A \vee F) \wedge$
 $(\neg E \vee \neg F \vee \neg A) \wedge$
 $(G \vee \neg A \vee E) \wedge$
 $(E \vee \neg G \vee \neg A) \wedge$
 $(A \vee H \vee C) \wedge$
 $(\neg H \vee \neg I \vee A) \wedge$
 $(I \vee L \vee M) \wedge$
 $(\neg L \vee C \vee \neg M) \wedge$
 $(A \vee \neg L \vee M) \wedge$
 $(L \vee N \vee \neg H) \wedge$
 $(I \vee L \vee \neg N)$



$\Rightarrow$ UNSAT

Remark: “choose-literal” selects only variables which still occur in the formula, after simplification. E.g., in the leftmost branch, after assigning $\neg C$, A , D , it does not select B because the clause $(B \vee A \vee C)$ has been simplified into true, and as such is no more part of the formula, so that B does not occur in the formula anymore.

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!

- 1 Propositional Logic
- 2 Propositional Reasoning**
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas**
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

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:
 - $A_1 \vee \neg A_2$ // definite
 - $A_2 \vee \neg A_3 \vee \neg A_4$ // definite
 - $\neg A_5 \vee \neg A_3 \vee \neg A_4$ // goal
 - A_3 // definite

- Intuition: implications between positive Boolean variables:

$$\begin{aligned} A_2 &\rightarrow A_1 \\ (A_3 \wedge A_4) &\rightarrow A_2 \\ (A_5 \wedge A_3 \wedge A_4) &\rightarrow \perp \\ &A_3 \end{aligned}$$

- 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:
 - $A_1 \vee \neg A_2$ // definite
 - $A_2 \vee \neg A_3 \vee \neg A_4$ // definite
 - $\neg A_5 \vee \neg A_3 \vee \neg A_4$ // goal
 - A_3 // definite

- Intuition: implications between positive Boolean variables:

$$\begin{aligned} A_2 &\rightarrow A_1 \\ (A_3 \wedge A_4) &\rightarrow A_2 \\ (A_5 \wedge A_3 \wedge A_4) &\rightarrow \perp \\ &A_3 \end{aligned}$$

- 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:
 - $A_1 \vee \neg A_2$ // definite
 - $A_2 \vee \neg A_3 \vee \neg A_4$ // definite
 - $\neg A_5 \vee \neg A_3 \vee \neg A_4$ // goal
 - A_3 // definite

- Intuition: implications between positive Boolean variables:

$$\begin{aligned} A_2 &\rightarrow A_1 \\ (A_3 \wedge A_4) &\rightarrow A_2 \\ (A_5 \wedge A_3 \wedge A_4) &\rightarrow \perp \\ &A_3 \end{aligned}$$

- 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:
 - $A_1 \vee \neg A_2$ // definite
 - $A_2 \vee \neg A_3 \vee \neg A_4$ // definite
 - $\neg A_5 \vee \neg A_3 \vee \neg A_4$ // goal
 - A_3 // definite

- Intuition: implications between positive Boolean variables:

$$\begin{aligned} A_2 &\rightarrow A_1 \\ (A_3 \wedge A_4) &\rightarrow A_2 \\ (A_5 \wedge A_3 \wedge A_4) &\rightarrow \perp \\ &A_3 \end{aligned}$$

- 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:
 - $A_1 \vee \neg A_2$ // definite
 - $A_2 \vee \neg A_3 \vee \neg A_4$ // definite
 - $\neg A_5 \vee \neg A_3 \vee \neg A_4$ // goal
 - A_3 // definite

- Intuition: implications between positive Boolean variables:

$$\begin{aligned} A_2 &\rightarrow A_1 \\ (A_3 \wedge A_4) &\rightarrow A_2 \\ (A_5 \wedge A_3 \wedge A_4) &\rightarrow \perp \\ &A_3 \end{aligned}$$

- 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: $\neg In12$

Tractability of Horn Formulas

Property

Checking the satisfiability of Horn formulas requires polynomial time:

- Hint:
 - Eliminate unit clauses by propagating their value;
 - If an empty clause is generated, return unsat
 - Otherwise, every clause contains at least one negative literal $\Rightarrow$ Assign all variables to $\perp$; return the assignment
- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting 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

⇒ Assign all variables to $\perp$; return the assignment

- Alternatively: run DPLL/CDCL, selecting negative literals first

A simple polynomial procedure for Horn-SAT

```
function Horn_SAT(formula  $\varphi$ , assignment &  $\mu$ ) {  
  Unit_Propagate( $\varphi$ ,  $\mu$ );  
  if ( $\varphi == \perp$ )  
    then return UNSAT;  
  else {  
     $\mu := \mu \cup \bigcup_{A_i \notin \mu} \{\neg A_i\}$ ;  
    return SAT;  
  } }
```

```
function Unit_Propagate(formula &  $\varphi$ , assignment &  $\mu$ )  
  while ( $\varphi \neq \top$  and  $\varphi \neq \perp$  and {a unit clause ( $l$ ) occurs in  $\varphi$ }) do {  
     $\varphi = \text{assign}(\varphi, l)$ ;  
     $\mu := \mu \cup \{l\}$ ;  
  } }
```

Example

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

Example

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

$$\mu := \{A_4 := \text{T}\}$$

Example

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

$$\mu := \{A_4 := \text{T}, A_3 := \text{T}\}$$

Example

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

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

Example

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

$\mu := \{A_4 := \top, A_3 := \top, A_2 := \perp, A_1 := \top\} \Rightarrow \text{UNSAT}$

Example 2

$A_1 \quad \vee \neg A_2$
 $A_2 \quad \vee \neg A_5 \quad \vee \neg A_4$
 $A_4 \quad \vee \neg A_3$
 A_3

Example 2

$$\begin{array}{ll} A_1 & \vee \neg A_2 \\ A_2 & \vee \neg A_5 \quad \vee \neg A_4 \\ A_4 & \vee \neg A_3 \\ A_3 & \end{array}$$
$$\mu := \{A_3 := \text{T}\}$$

Example 2

$A_1 \quad \vee \neg A_2$
 $A_2 \quad \vee \neg A_5 \quad \vee \neg A_4$
 $A_4 \quad \vee \neg A_3$
 A_3

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

Example 2

$A_1 \quad \vee \neg A_2$
 $A_2 \quad \vee \neg A_5 \quad \vee \neg A_4$
 $A_4 \quad \vee \neg A_3$
 A_3

$\mu := \{A_3 := \text{T}, A_4 := \text{T}\} \implies \text{SAT}$

- 1 Propositional Logic
- 2 Propositional Reasoning**
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search**
- 3 Agents Based on Knowledge Representation & Reasoning
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

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 $\leftarrow$ 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 $\leftarrow$ 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*

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning**
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning**
 - Knowledge-Based Agents**
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World

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
 - represent the agent's **representation of the world**
 - expressed in a **formal language** (e.g. propositional logic)
 - collection of **domain-specific facts** believed by the agent
 - 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
 - represent the agent's **representation of the world**
 - expressed in a **formal language** (e.g. propositional logic)
 - collection of **domain-specific facts** believed by the agent
 - 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
 - represent the agent's **representation of the world**
 - expressed in a **formal language** (e.g. propositional logic)
 - collection of **domain-specific facts** believed by the agent
 - 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



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)
 - **Probabilistic 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)
 - Probabilistic 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)
 - Probabilistic 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)
 - Probabilistic 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)
 - **Probabilistic 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)
 - **Probabilistic 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

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

- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning**
 - Knowledge-Based Agents
 - Example: the Wumpus World**
 - Propositional-Logic Agents
 - Example: the Wumpus World

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

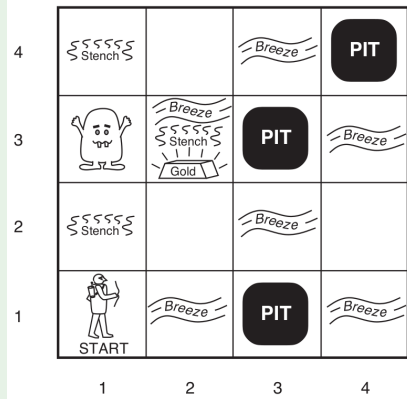
Actuators:

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

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



(© S. Russell & P. Norvig, 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

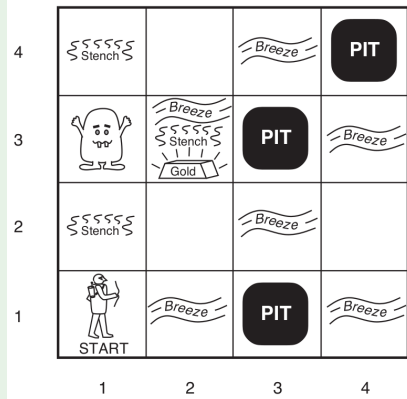
Actuators:

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

Sensors:

- Stench, Breeze, Glitter, Bump, Scream

One possible configuration:



(© S. Russell & P. Norvig, 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

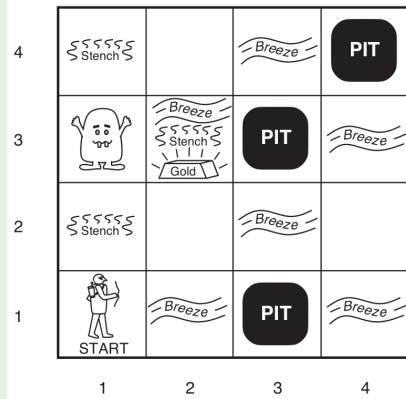
Actuators:

- 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:

4	 Stench		 Breeze	PIT
3		 Breeze  Stench  Gold	PIT	 Breeze
2	 Stench		 Breeze	
1	 START	 Breeze	PIT	 Breeze
	1	2	3	4

(© S. Russell & P. Norwig, AIMA)

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

⇒ [1,2] and [2,1] OK

OK			
OK A	OK		

A: Agent; B: Breeze; G: Glitter; S: Stench

OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

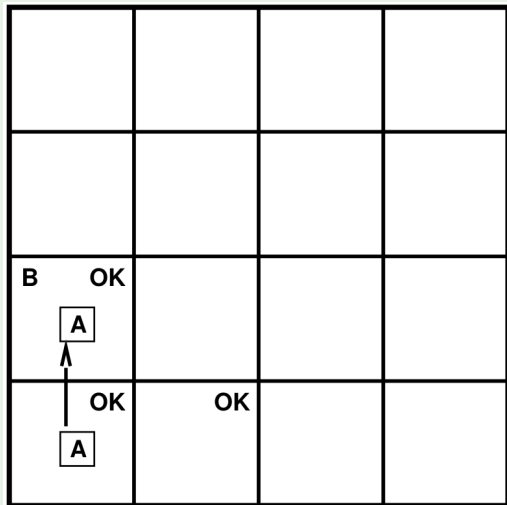
Example: Exploring the Wumpus World

- 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

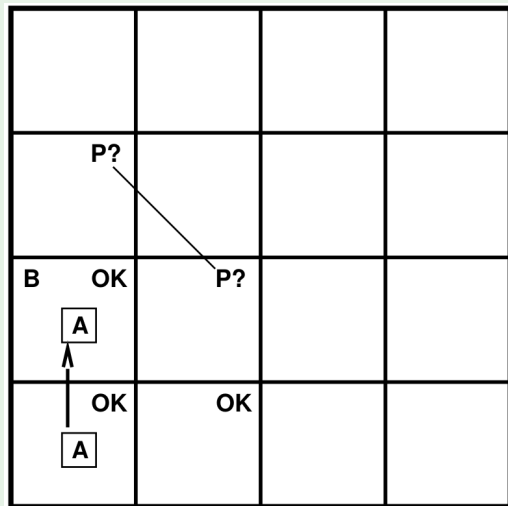
- Agent moves to [2,1]

- perceives a breeze

⇒ 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 [1,1]-[1,2]

- perceives no breeze

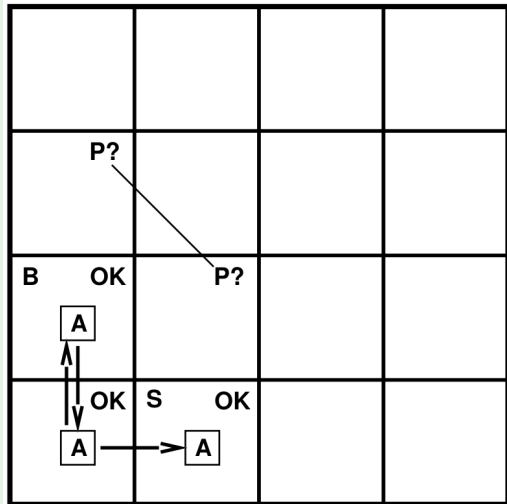
⇒ no Pit in [1,3], [2,2]

⇒ [2,2] OK

⇒ pit in [3,1]

- perceives a stench

⇒ Wumpus in [2,2] or [1,3]!

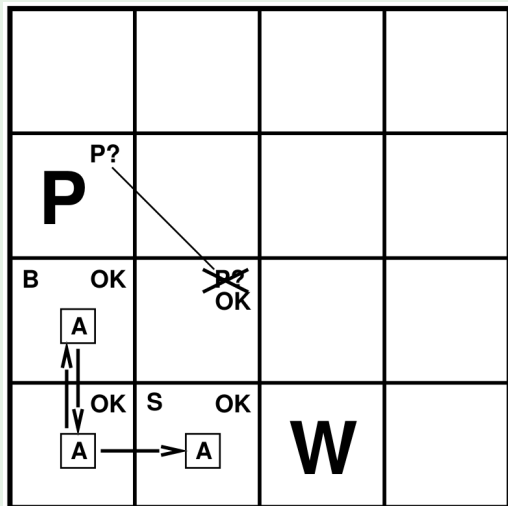


A: Agent; B: Breeze; G: Glitter; S: Stench

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]!

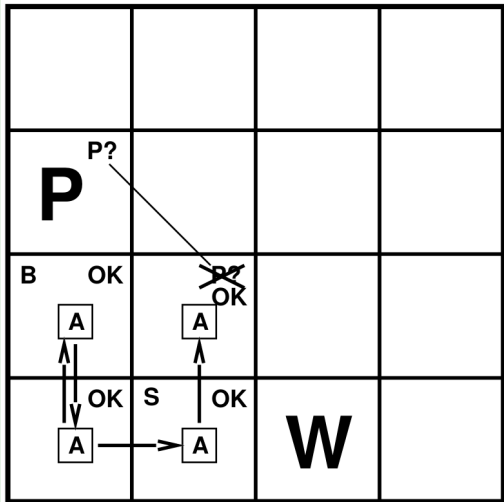


A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

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

- perceives no stench



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]

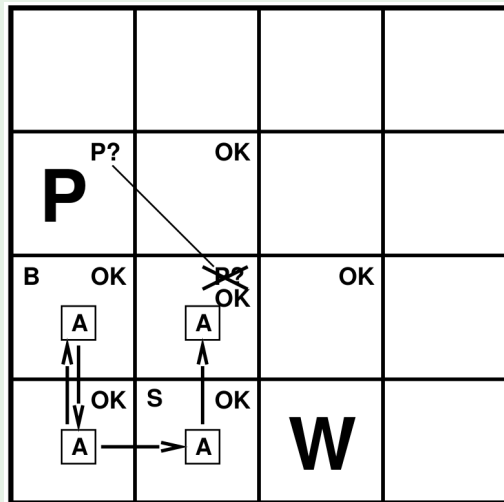
- perceives no breeze

⇒ no pit in [3,2], [2,3]

- perceives no stench

⇒ no Wumpus in [3,2], [2,3]

⇒ [3,2] and [2,3] OK



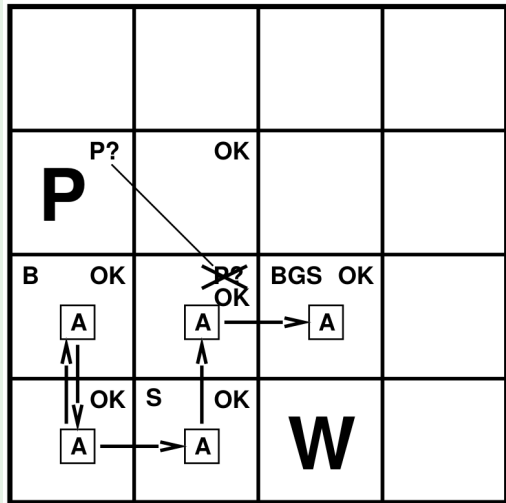
A: Agent; B: Breeze; G: Glitter; S: Stench

OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

Example: Exploring the Wumpus World

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

⇒ bag of gold!

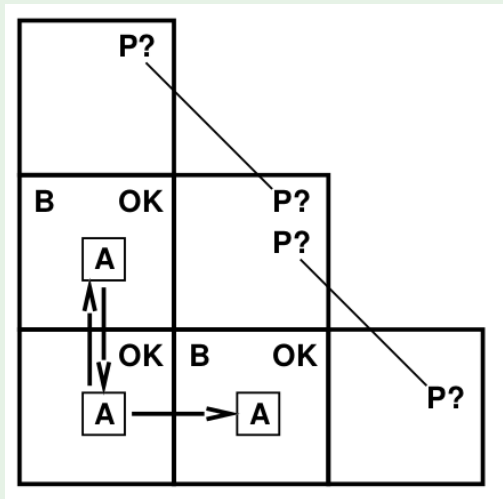


A: Agent; B: Breeze; G: Glitter; S: Stench
OK: safe square; W: Wumpus; P: Pit; BGS: bag of gold

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

Alternative scenario: probabilistic solution (hints)

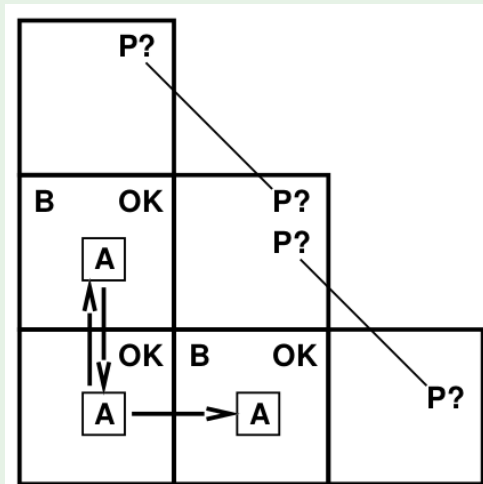
- Feel breeze in [1,2] and [2,1]
- ⇒ pit in [1,3] or [2,2] or [3,1]
- ⇒ **no 100% safe action**
- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
 $P(\text{pit} \in [2,2]) = 0.86$
 $P(\text{pit} \in [1,3]) = 0.31$
 $P(\text{pit} \in [3,1]) = 0.31$
- ⇒ **better choose [1,3] or [3,1]**



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

Alternative scenario: probabilistic solution (hints)

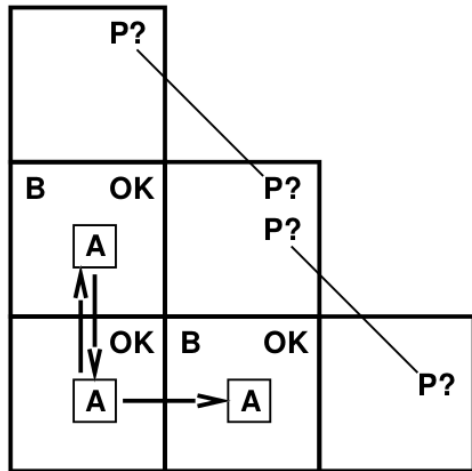
- Feel breeze in [1,2] and [2,1]
- ⇒ pit in [1,3] or [2,2] or [3,1]
- ⇒ **no 100% safe action**
- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
 - $P(\text{pit} \in [2, 2]) = 0.86$
 - $P(\text{pit} \in [1, 3]) = 0.31$
 - $P(\text{pit} \in [3, 1]) = 0.31$
- ⇒ **better choose [1,3] or [3,1]**



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

Alternative scenario: probabilistic solution (hints)

- Feel breeze in [1,2] and [2,1]
- ⇒ pit in [1,3] or [2,2] or [3,1]
- ⇒ **no 100% safe action**
- Probability analysis [see Ch 13] (assuming pits uniformly distributed):
 - $P(\text{pit} \in [2, 2]) = 0.86$
 - $P(\text{pit} \in [1, 3]) = 0.31$
 - $P(\text{pit} \in [3, 1]) = 0.31$
- ⇒ **better choose [1,3] or [3,1]**



- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning**
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents**
 - Example: the Wumpus World

Propositional Logic Agents

- Kind of Logic agents
- Language: **propositional logic, first-order 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, first-order 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**

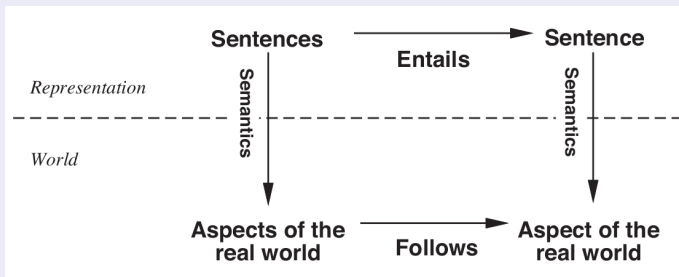
Representation vs. World

Reasoning process (propositional entailment) sound

⇒ 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

⇒ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



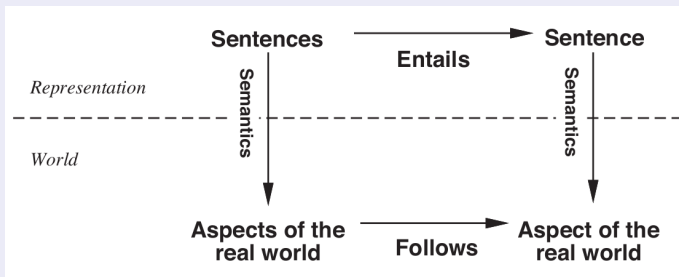
Representation vs. World

Reasoning process (propositional entailment) sound

⇒ 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

⇒ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



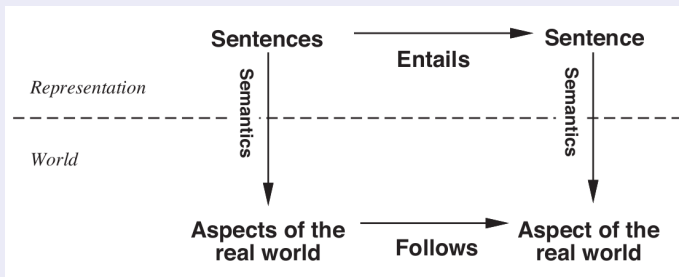
Representation vs. World

Reasoning process (propositional entailment) sound

⇒ 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

⇒ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent

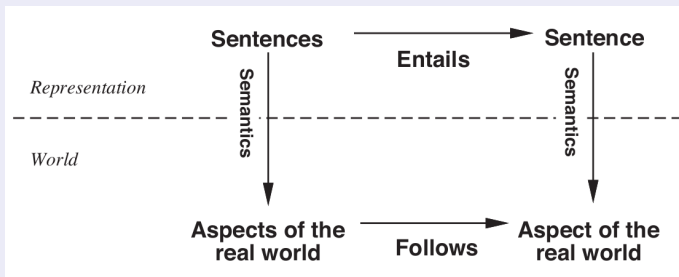


Representation vs. World

Reasoning process (propositional entailment) sound

⇒ 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
- ⇒ the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent



- 1 Propositional Logic
- 2 Propositional Reasoning
 - Resolution
 - DPLL
 - Reasoning with Horn Formulas
 - Local Search
- 3 Agents Based on Knowledge Representation & Reasoning**
 - Knowledge-Based Agents
 - Example: the Wumpus World
 - Propositional-Logic Agents
 - Example: the Wumpus World**

Example: Exploring the Wumpus World

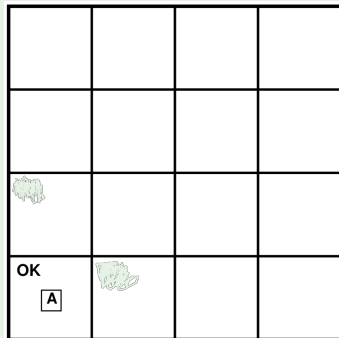
KB initially contains (the CNFized versions of) the following formulas, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours
 $B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$
- stench iff Wumpus in neighbours
 $S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$
- 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 $\implies$ 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, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours

$$B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$$

- stench iff Wumpus in neighbours

$$S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$$

- 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

$$G_{[i,j]} \leftrightarrow BGS_{[i,j]}$$

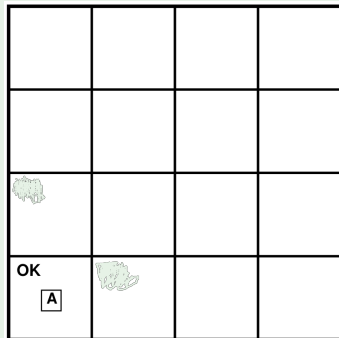
- in $[1, 1]$ no Wumpus and no pit $\implies$ safe

$$\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$$

(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, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours

$$B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$$

- stench iff Wumpus in neighbours

$$S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$$

- 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

$$G_{[i,j]} \leftrightarrow BGS_{[i,j]}$$

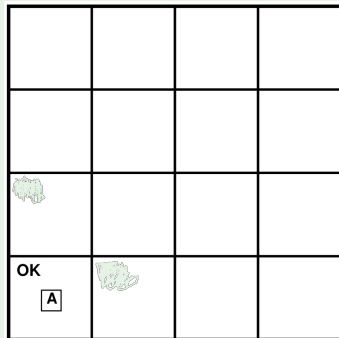
- in $[1, 1]$ no Wumpus and no pit $\implies$ safe

$$\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$$

(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, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours

$$B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$$

- stench iff Wumpus in neighbours

$$S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$$

- 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 $\implies$ 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

			
OK A			

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours

$$B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$$

- stench iff Wumpus in neighbours

$$S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$$

- 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 $\implies$ 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

			
OK A			

Example: Exploring the Wumpus World

KB initially contains (the CNFized versions of) the following formulas, $\forall i, j \in [1..4]$:

- breeze iff pit in neighbours

$$B_{[i,j]} \leftrightarrow (P_{[i,j-1]} \vee P_{[i+1,j]} \vee P_{[i,j+1]} \vee P_{[i-1,j]})$$

- stench iff Wumpus in neighbours

$$S_{[i,j]} \leftrightarrow (W_{[i,j-1]} \vee W_{[i+1,j]} \vee W_{[i,j+1]} \vee W_{[i-1,j]})$$

- 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 $\implies$ 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

			
OK 			

Example: Exploring the Wumpus World

- KB initially contains:

$\neg P_{[1,1]}, \neg W_{[1,1]}, OK_{[1,1]}$

$B_{[1,1]} \leftrightarrow (P_{[1,2]} \vee P_{[2,1]})$

$S_{[1,1]} \leftrightarrow (W_{[1,2]} \vee W_{[2,1]})$

$OK_{[1,2]} \leftrightarrow (\neg W_{[1,2]} \wedge \neg P_{[1,2]})$

$OK_{[2,1]} \leftrightarrow (\neg W_{[2,1]} \wedge \neg P_{[2,1]})$

...

- Agent is initially in 1,1

- Percepts (no stench, no breeze): $\neg S_{[1,1]}, \neg B_{[1,1]}$

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

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

- Add all them to KB

OK			
OK A	OK		

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]

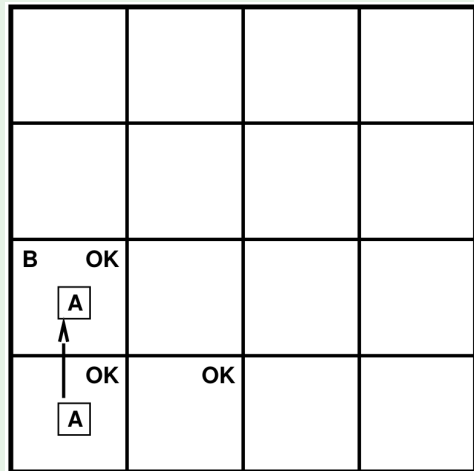
- perceives a breeze: $B_{[2,1]}$

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

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

$\Rightarrow \neg W_{[3,1]}, \neg W_{[2,2]}$
(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: $B_{[2,1]}$

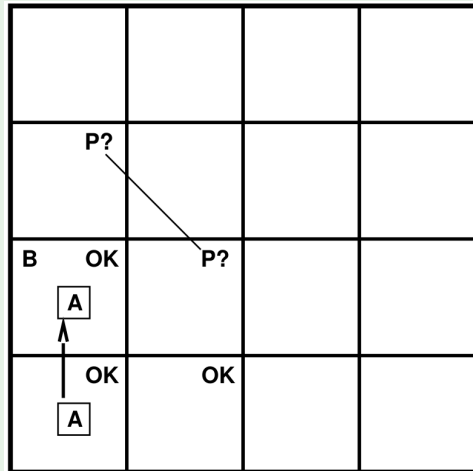
⇒ $(P_{[3,1]} \vee P_{[2,2]})$ (pit in [3,1] or [2,2])

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

⇒ $\neg W_{[3,1]}, \neg W_{[2,2]}$

(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: $\neg B_{[1,2]}$

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

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

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

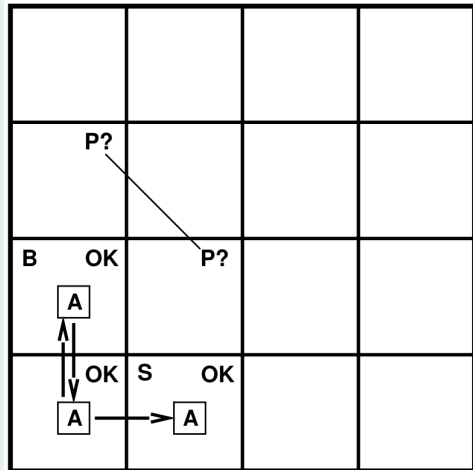
$\Rightarrow W_{[1,3]}$ (Wumpus in [1,3]!)

$\Rightarrow OK_{[2,2]}$ ([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: $\neg B_{[1,2]}$

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

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

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

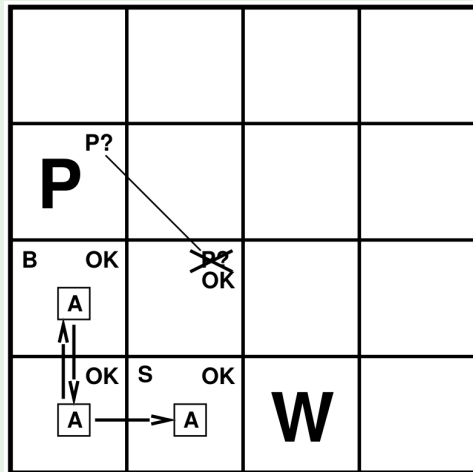
$\Rightarrow W_{[1,3]}$ (Wumpus in $[1,3]$!)

$\Rightarrow OK_{[2,2]}$ ($[2,2]$ 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: $\neg B_{[2,2]}$

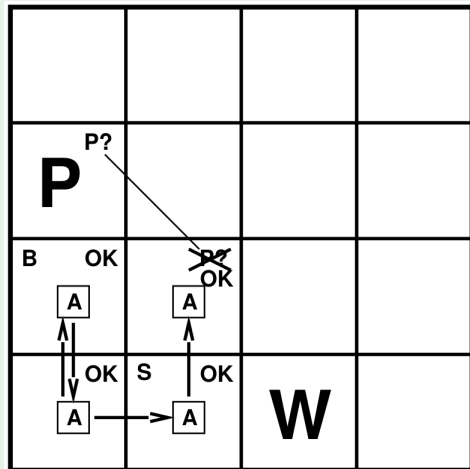
$\Rightarrow \neg P_{[3,2]}, \neg P_{[2,3]}$ (no pit in [3,2], [2,3])

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

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

$\Rightarrow OK_{[3,2]}, OK_{[2,3]}$, ([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: $\neg B_{[2,2]}$

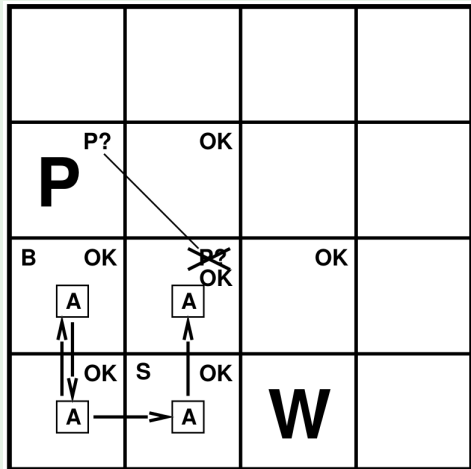
$\Rightarrow \neg P_{[3,2]}, \neg P_{[2,3]}$ (no pit in $[3,2], [2,3]$)

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

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

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

- 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:

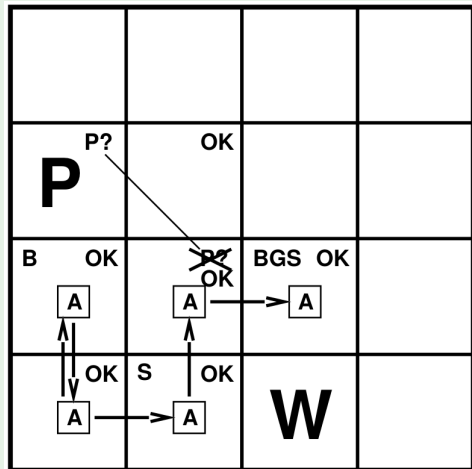
$G_{[2,3]} \leftrightarrow BGS_{[2,3]}$

- Agent moves to [2,3]

- perceives a glitter: $G_{[2,3]}$

⇒ $BGS_{[2,3]}$ (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