Artificial Intelligence

4. Propositional Logic

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

What is propositional logic?

In propositional logic, atomic formulas are propositions, i.e., assertions such as

A := “Aristotle is dead.”
B := “Hildesheim is on the Rhine.”
C := “Logic is fun.”

Atomic formulas are denoted by capital letters $A, B, C,$ etc.

Each atomic formula is assigned a truth value: true (1) or false (0). (In fuzzy logic truth values can be degrees between 0 and 1.)

“Propositional logic is not the study of truth,
but of the relationship between the truth of one statement and that of another” (Hedman 2004).

What is propositional logic?

There are several relationships between propositions that can be expressed:

word	symbol	example		terminus technicus
not	$\neg$	not A	$\neg A$	negation
and	$\wedge$	A and B	$A \wedge B$	conjunction
or	$\vee$	A or B	$A \vee B$	disjunction
implies	$\rightarrow$	A implies B	$A \rightarrow B$	implication
if and only if	$\leftrightarrow$	A if and only if B	$A \leftrightarrow B$	biconditional

What is propositional logic?

The natural language words may have slightly different meanings.

Example:

$A \wedge B$ and $B \wedge A$ should always have the same meaning.

But the sentences

She became sick and she went to the doctor.

and

She went to the doctor and she became sick.

have different meanings.

Syntax

$$
\begin{aligned}
\textit{formula} &:= \textit{atomicFormula} \mid \textit{complexFormula} \\
\textit{atomicFormula} &:= \textbf{True} \mid \textbf{False} \mid \textit{symbol} \\
\textit{symbol} &:= \textbf{P} \mid \textbf{Q} \mid \textbf{R} \mid \ldots \\
\textit{complexFormula} &:= \textbf{(}\ \neg\ \textit{formula}\ \textbf{)} && \text{[negation]} \\
&\mid\ \textbf{(}\ \textit{formula}\ \wedge\ \textit{formula}\ \textbf{)} && \text{[conjunction]}
\end{aligned}
$$

[positive literal] the same as *atomicFormula*.

[negative literal] $\neg$ P where P is an atomic formula.

Formulas also are called sentences or propositions.

Syntax

Let $\mathcal{S}$ be a set of atomic formulas.

$\mathcal{F}(\mathcal{S})$ denotes the **set of formulas over** $\mathcal{S}$,
i.e., the smallest set

- that contains $\mathcal{S}$: $\mathcal{S} \subseteq \mathcal{F}$,
- is closed under $\neg$: for all $F \in \mathcal{F}$, also $\neg F \in \mathcal{F}$, and
- is closed under $\wedge$: for all $F, G \in \mathcal{F}$, also $F \wedge G \in \mathcal{F}$.

Syntax / Abbreviations

Some additional operators are introduced as abbreviations of some complex formulas:

$$
\begin{aligned}
A \vee B &:= \neg(\neg A \wedge \neg B) \\
A \rightarrow B &:= \neg A \vee B && = \neg(\neg\neg A \wedge \neg B) \\
A \leftrightarrow B &:= (A \rightarrow B) \wedge (B \rightarrow A) && = \neg(\neg\neg A \wedge \neg B) \wedge \neg(\neg\neg B \wedge \neg A)
\end{aligned}
$$

Syntax / Bracket Simplification (1/2)

The formal syntax makes extensive use of brackets to make grouping unambiguous:

$$(((\neg P) \vee (Q \wedge R)) \Rightarrow S)$$

With the usual precedence rules

from highest to lowest: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

many brackets can be droppend and formulas be simplified:

$$\neg P \vee Q \wedge R \Rightarrow S$$

Syntax / Bracket Simplification (2/2)

Chaining of the same operator still is ambiguous on syntactic level.

The formulas

$$P \wedge Q \wedge R \quad \text{and} \quad P \vee Q \vee R \quad \text{and} \quad P \Leftrightarrow Q \Leftrightarrow R$$

can safely be used as the semantics (see below) of both possible groupings

$$(P \wedge Q) \wedge R \quad \text{and} \quad P \wedge (Q \wedge R)$$

is the same.

But

$$P \Rightarrow Q \Rightarrow R$$

is not allowed because the semantics depends on the grouping.

Subformulas

Subformulas of a formula F are substrings that are formulas (if all parentheses are present).

But beware if you omit parentheses!
($A \wedge B$ is a substring of $\neg A \wedge B$, but not a subformula!)

Subformulas also can be recursively defined by

$$\text{subformula}(F) := \begin{cases} \{F\}, & \text{if } F \text{ is atomic} \\ \{\neg G\} \cup \text{subformula}(G), & \text{if } F = \neg G \\ \{G \wedge H\} \cup \text{subformula}(G) \cup \text{subformula}(H), & \text{if } F = G \wedge H \end{cases}$$

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Assignments

Let $\mathcal{S}$ be a set of atomic formulas.

A function $\mathcal{A} : \mathcal{S} \to \{0, 1\}$ is called an **assignment of** $\mathcal{S}$ or a **model of** $\mathcal{S}$.

Assignments of complex formulas are defined in terms of assignments of their subformulas:

$$\mathcal{A}(\neg F) := 1 - \mathcal{A}(F)$$
$$\mathcal{A}(F \wedge G) := \mathcal{A}(F)\mathcal{A}(G)$$

These functions usually are presented as **truth tables**:

F	$\neg F$
0	1
1	0

F	G	$F \wedge G$
0	0	0
0	1	0
1	0	0
1	1	1

Assignments

This way, assignments are extended from $\mathcal{A} : \mathcal{S} \to \{0, 1\}$ unambiguously to

$$\mathcal{A} : \mathcal{F}(\mathcal{S}) \to \{0, 1\}$$

Assignments can even be extended to some formulas not in $\mathcal{F}(\mathcal{S})$, but say in $\mathcal{F}(\mathcal{S} \cup \mathcal{S}')$ with some additional atomic formulas $\mathcal{S}'$:

Let $F \in \mathcal{F}(\mathcal{S} \cup \mathcal{S}')$.

If for every extension of $\mathcal{A}$ to $\mathcal{S} \cup \mathcal{S}'$
— i.e., every assignment $\mathcal{A}'$ of $\mathcal{S} \cup \mathcal{S}'$ with $\mathcal{A}'|_{\mathcal{S}} = \mathcal{A}$ —
$\mathcal{A}'(F)$ has the same value,
then $\mathcal{A}(F) := \mathcal{A}'(F)$.

Assignments

Example:

Let $\mathcal{S} := \{A, B\}$
and $\mathcal{A}(A) := 1$ and $\mathcal{A}(B) := 0$ an assignment on $\mathcal{S}$.

Then

$$\begin{aligned}\mathcal{A}(A \wedge B) &= 0\\ \mathcal{A}(A \vee B) &= 1\\ \mathcal{A}(A \wedge (C \vee \neg C)) &= 1\\ \mathcal{A}(A \wedge C \wedge \neg C) &= 0\end{aligned}$$

Validity, Satisfiability, Contradiction

If $\mathcal{A}(F) = 1$, we say

- F **holds under assignment** $\mathcal{A}$ or
- $\mathcal{A}$ **models** F

and write

$$\mathcal{A} \models F$$

A formula F is called

valid / a tautology, written $\models F$
if it holds under every assignment.

satisfiable
if it holds under some assignment.

unsatisfiable / a contradiction
if it holds under no assignment.

Examples:

$C \vee \neg C$ is valid.

$C \wedge \neg C$ is unsatisfiable.

Decision problems

Given a formula $F \in \mathcal{F}(\mathcal{S})$,
decide if

- F is valid ?
- F is satisfiable ?

Simple algorithm:
enumerate all possible assignments $\mathcal{A}$ of $\mathcal{S}$ and compute $\mathcal{A}(F)$.
if $\mathcal{A}(F) = 1$ for all $\mathcal{A}$, F is valid.
if $\mathcal{A}(F) = 1$ for at least one $\mathcal{A}$, F is satisfiable.
otherwise F is unsatisfiable.

If F has n different atomic subformulas,
the runtime complexity of exhaustive enumeration is $O(2^n)$.

Consequence / Entailment

Let $F, G \in \mathcal{F}(\mathcal{S})$ be two formulas.

G is said to be **a consequence of** F or F **to entail** G

$$F \models G$$

if for every assignment $\mathcal{A}$ under which F holds, also G holds,
i.e., if $\mathcal{A} \models F$, then $\mathcal{A} \models G$.

Lemma 1. $F \models G$ *if and only if* $\models F \to G$

Proof.
$\not\models F \to G$, i.e., there exists an assignment $\mathcal{A}$ with $\mathcal{A}(F \to G) = 0$
$\Leftrightarrow 1 = \mathcal{A}(\neg(F \to G)) = \mathcal{A}(\neg(\neg F \vee G)) = \mathcal{A}(F \wedge \neg G)$
$\Leftrightarrow \mathcal{A}(F) = 1$ and $\mathcal{A}(\neg G) = 1$,
i.e., $\mathcal{A} \models F$ but not $\mathcal{A} \models G$,
i.e., $F \not\models G$. □

Equivalence

Let $F, G \in \mathcal{F}(\mathcal{S})$ be two formulas.

G is said to be **equivalent to** F

$$F \equiv G$$

if G is a consequence of F and F is a consequence of G.

Examples:

$$\begin{aligned}
F \wedge (G \vee H) &\equiv (F \wedge G) \vee (F \wedge H) && \text{[distributivity rule]} \\
F \vee (G \wedge H) &\equiv (F \vee G) \wedge (F \vee H) && \text{[distributivity rule]} \\
\neg(F \wedge G) &\equiv \neg F \vee \neg G && \text{[DeMorgan's rule]} \\
\neg(F \vee G) &\equiv \neg F \wedge \neg G && \text{[DeMorgan's rule]}
\end{aligned}$$

Consequences / Entailment of Theories

Let $\mathcal{F} \subseteq \mathcal{F}(\mathcal{S})$ be a set of formulas (also called a **theory** or a **knowledge base**).
An assignment $\mathcal{A}$ **models** $\mathcal{F}$

$$\mathcal{A} \models \mathcal{F}$$

if it models every $F \in \mathcal{F}$.

A formula $G \in \mathcal{F}(\mathcal{S})$ is said to be **a consequence of theory** $\mathcal{F}$ or a theory $\mathcal{F}$ to **entail** G

$$\mathcal{F} \models G$$

if every assignment $\mathcal{A}$ that models $\mathcal{F}$, also models G,
i.e., if $\mathcal{A} \models \mathcal{F}$, then $\mathcal{A} \models G$.

If $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ is finite, then

$$\mathcal{A} \models \{F_1, F_2, \ldots, F_n\} \Leftrightarrow \mathcal{A} \models F_1 \wedge F_2 \wedge \ldots \wedge F_n$$

The Consequence Problem

Given two formulas F and G,
decide if G is a consequence of F: $F \models G$?

Given a theory $\mathcal{F}$ and a formula G,
decide if G is a consequence of $\mathcal{F}$: $\mathcal{F} \models G$?

The problem for finite theories can be reduced to the problem for a single formula via conjunction.

But as for the other problems, exhaustive enumeration is $O(2^n)$ in the number n of atomic formulas.

But what to do if $\mathcal{F}$ is infinite?

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Example (1/2)

Example:

Let

$$\mathcal{F} := \{A, A \to B, B \to C, C \to D, D \to E, E \to F, F \to G\}$$

and say we want to decide if $\mathcal{F}$ entails $F \vee G$?

There are $2^7 = 128$ different assignments for $\mathcal{S} = \{A, B, \ldots, G\}$ that would have to be checked separately.

assignment $\mathcal{A}$							theory $\mathcal{F}$							
A	B	C	D	E	F	G	A	$A \to B$	$B \to C$	$C \to D$	$D \to E$	$E \to F$	$F \to G$	$F \vee G$
0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	0	0	0	1	0	1	1	1	1	1	1	1
⋮							⋮							⋮
1	0	0	0	0	0	0	1	0	1	1	1	1	1	0
⋮							⋮							⋮
1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Example (2/2)

Example:

Let

$$\mathcal{F} := \{A, A \to B, B \to C, C \to D, D \to E, E \to F, F \to G\}$$

and say we want to decide if $\mathcal{F}$ entails $F \vee G$?

On the other hand, we could derive $F \vee G$ from the formulas $\mathcal{F}$ step by step via

$A, A \to B$	therefore B
$B, B \to C$	therefore C
$C, C \to D$	therefore D
$D, D \to E$	therefore E
$E, E \to F$	therefore F
F	therefore $F \vee G$

Proofs

Let $\mathcal{F}$ be a set of formulas and G be a formula.

$\mathcal{F} \vdash G$ denotes that formula G can be derived from formulas $\mathcal{F}$.

A **proof system** consists of a set of **derivation rules** r
that state which derivations follow from which other derivations
(parametrized by variables for subformulas).

A **proof for** $\mathcal{F} \models G$ is a sequence of derivations

$$\begin{aligned}
\mathcal{F}_1 &\vdash G_1 \\
\mathcal{F}_2 &\vdash G_2 \\
&\vdots \\
\mathcal{F}_n &\vdash G_n
\end{aligned}$$

such that each derivation follows from earlier derivations by some rule and $G_n = G$.

Example (1/2)

Let the proof system consist of just three rules:

premise	conclusion	name
$F \in \mathcal{F}$	$\mathcal{F} \vdash F$	assumption
$\mathcal{F} \vdash F$	$\mathcal{F} \vdash F \vee G$	$\vee$-introduction
$\mathcal{F} \vdash F, \mathcal{F} \vdash F \rightarrow G$	$\mathcal{F} \vdash G$	$\rightarrow$ -elimination

Example (2/2)

The we can proof $F \vee G$ from

$$\mathcal{F} := \{A, A \to B, B \to C, C \to D, D \to E, E \to F, F \to G\}$$

by

1. $\mathcal{F} \vdash A$ [assumption]
2. $\mathcal{F} \vdash A \to B$ [assumption]
3. $\mathcal{F} \vdash B$ [$\to$-elimination applied to 1,2]
4. $\mathcal{F} \vdash B \to C$ [assumption]
5. $\mathcal{F} \vdash C$ [$\to$-elimination applied to 3,4]
6. $\mathcal{F} \vdash C \to D$ [assumption]
7. $\mathcal{F} \vdash D$ [$\to$-elimination applied to 5,6]
8. $\mathcal{F} \vdash D \to E$ [assumption]
9. $\mathcal{F} \vdash E$ [$\to$-elimination applied to 7,8]
10. $\mathcal{F} \vdash E \to F$ [assumption]
11. $\mathcal{F} \vdash F$ [$\to$-elimination applied to 9,10]
12. $\mathcal{F} \vdash F \vee G$ [$\vee$-introduction applied to 11]

Basic rules for derivation

premise	conclusion	name
$G \in \mathcal{F}$	$\mathcal{F} \vdash G$	assumption
$\mathcal{F} \vdash G, \mathcal{F} \subseteq \mathcal{F}'$	$\mathcal{F}' \vdash G$	monotonicity
$\mathcal{F} \vdash \neg\neg G$	$\mathcal{F} \vdash G$	double negation
$\mathcal{F} \vdash F, \mathcal{F} \vdash G$	$\mathcal{F} \vdash F \wedge G$	$\wedge$-introduction
$\mathcal{F} \vdash F \wedge G$	$\mathcal{F} \vdash F$	$\wedge$-elimination
$\mathcal{F} \vdash F \wedge G$	$\mathcal{F} \vdash G \wedge F$	$\wedge$-symmetry
$\mathcal{F} \vdash F$	$\mathcal{F} \vdash F \vee G$	$\vee$-introduction
$\mathcal{F} \vdash F \vee G$, $\mathcal{F} \cup \{F\} \vdash H, \mathcal{F} \cup \{G\} \vdash H$	$\mathcal{F} \vdash H$	$\vee$-elimination
$\mathcal{F} \vdash F \vee G$	$\mathcal{F} \vdash G \vee F$	$\vee$-symmetry
$\mathcal{F} \cup \{F\} \vdash G$	$\mathcal{F} \vdash F \to G$	$\to$ -introduction
$\mathcal{F} \vdash F, \mathcal{F} \vdash F \to G$	$\mathcal{F} \vdash G$	$\to$ -elimination
$\mathcal{F} \vdash F$	$\mathcal{F} \vdash (F)$	$()$-introduction
$\mathcal{F} \vdash (F)$	$\mathcal{F} \vdash F$	$()$-elimination
$\mathcal{F} \vdash ((F \wedge G) \wedge H)$	$\mathcal{F} \vdash F \wedge G \wedge H$	$\wedge$-parentheses rule
$\mathcal{F} \vdash ((F \vee G) \vee H)$	$\mathcal{F} \vdash F \vee G \vee H$	$\vee$-parentheses rule

More rules

Additionally one needs rules for definitions of $\vee$, $\rightarrow$ and $\leftrightarrow$:

rule			name
$\mathcal{F} \vdash F \vee G$	if and only if	$\mathcal{F} \vdash \neg(\neg F \wedge \neg G)$	$\vee$-definition
$\mathcal{F} \vdash F \rightarrow G$	if and only if	$\mathcal{F} \vdash \neg F \vee G$	$\rightarrow$-definition
$\mathcal{F} \vdash F \leftrightarrow G$	if and only if	$\mathcal{F} \vdash F \rightarrow G, \mathcal{F} \vdash G \rightarrow F$	$\leftrightarrow$-definition

More rules can be derived from these rules.

E.g.,

premise	conclusion	name
none	$\mathcal{F} \vdash \neg G \vee G$	tautology rule

proof:

1. $\mathcal{F} \cup \{G\} \vdash G$ [assumption]
2. $\mathcal{F} \vdash G \rightarrow G$ [$\rightarrow$-introduction applied to 1]
3. $\mathcal{F} \vdash \neg G \vee G$ [$\rightarrow$-definition applied to 2]

More rules

premise	conclusion	name
none	$\mathcal{F} \vdash \neg G \vee G$	tautology rule

Soundness

A proof system is called **sound** if $\mathcal{F} \vdash G$, then $\mathcal{F} \models G$.

Lemma 2. *The proof system outlined in this section is sound.*

Proof. Show for each rule, that for its conclusion $\mathcal{F} \vdash G$: $\mathcal{F} \models G$.

$\wedge$-elimination:

F	G	$F \wedge G$	$F \wedge G \rightarrow F$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

□

Completeness

A proof system is called **complete** if $\mathcal{F} \models G$, then $\mathcal{F} \vdash G$.

Lemma 3. *The proof system outlined in this section is complete.*

Proof. See Hedmann 2004, p. 47.

□

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Definition Normal Forms

Formulas

$$F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j}$$

where $L_{i,j}$ is a literal (i.e., either an atom or the negation of an atom) are called **conjunctive normal form** (**CNF**).

Formulas

$$F = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} L_{i,j}$$

where $L_{i,j}$ is a literal (i.e., either an atom or the negation of an atom) are called **disjunctive normal form** (**DNF**).

Here we write

$$\bigwedge_{i=1}^{n} F_i := F_1 \wedge F_2 \wedge \ldots \wedge F_n, \quad \bigvee_{i=1}^{n} F_i := F_1 \vee F_2 \vee \ldots \vee F_n$$

Definition Normal Forms / Examples

Examples:

$(A \vee B) \wedge (C \vee D) \wedge (\neg A \vee \neg B \vee \neg D)$ is in CNF

$(\neg A \wedge B) \vee C \vee (B \wedge \neg C \wedge D)$ is in DNF

$(A \vee B) \wedge ((A \wedge C) \vee (B \wedge D))$ is neither

Existence of CNF and DNF Equivalents

Lemma 4. *Each formula is equivalent to a formula in CNF as well as to a formula in DNF.*

Proof. Let F be the formula and proof by induction:
if F is atomic, it clearly is CNF as well as DNF.
if $F = \neg G$, then let $G \equiv \bigwedge_i \bigvee_j L_{i,j}$ by induction.

$$F = \neg G \equiv \neg \bigwedge_i \bigvee_j L_{i,j} \equiv \bigvee_i \bigwedge_j \neg L_{i,j} \equiv \bigvee_i \bigwedge_j L'_{i,j}$$

where $L'_{i,j} := \neg A$ if $L_{i,j} = A$ is atomic and $L'_{i,j} := A$ if $L_{i,j} = \neg A$ is the negation of an atom, i.e., F is equivalent to a DNF. Similiar for CNF.

if $F = G \wedge H$, then F obviously is equivalent to a CNF.

To show that F is also equivalent to a DNF, let $G \equiv \bigvee_i L_i$ and $H \equiv \bigvee_j M_j$ be DNFs, respectively. Then

$$F = G \wedge H \equiv (\bigvee_i L_i) \wedge (\bigvee_j M_j) \equiv \bigvee_i \bigvee_j (L_i \wedge M_j)$$

is in DNF. □

CNF/DNF algorithm

1 cnf(formula F) :
2 Replace all subformulas $(G \leftrightarrow H)$ of F by $((\neg G \vee H) \wedge (\neg H \vee G))$
3 Replace all subformulas $(G \rightarrow H)$ of F by $(\neg G \vee H)$
5 Move $\neg$ inwards:
6 Replace all subformulas $\neg(G \wedge H)$ of F by $(\neg G \vee \neg H)$
7 Replace all subformulas $\neg(G \vee H)$ of F by $(\neg G \wedge \neg H)$
8 Replace all subformulas $\neg\neg G$ of F by G
10 Apply the $\vee$ -distributivity rule:
11 Replace all subformulas $G \vee (H \wedge I)$ of F by $(G \vee H) \wedge (G \vee I)$
12 Replace all subformulas $(H \wedge I) \vee G$ of F by $(H \vee G) \wedge (I \vee G)$
13 **return** F

CNF/DNF algorithm / Example

$$
\begin{aligned}
F =& (A \vee B) \rightarrow (\neg B \wedge A) \\
\equiv& \neg(A \vee B) \vee (\neg B \wedge A) \\
\equiv& (\neg A \wedge \neg B) \vee (\neg B \wedge A) \\
\equiv& ((\neg A \wedge \neg B) \vee \neg B) \wedge ((\neg A \wedge \neg B) \vee A) \\
\equiv& ((\neg A \vee \neg B) \wedge (\neg B \vee \neg B)) \wedge ((\neg A \vee A) \wedge (\neg B \vee A)) \\
\equiv& (\neg A \vee \neg B) \wedge (\neg B \vee \neg B) \wedge (\neg A \vee A) \wedge (\neg B \vee A) \\
\\
\equiv& (\neg A \vee \neg B) \wedge \neg B \wedge (\neg B \vee A) \\
\equiv& \neg B
\end{aligned}
$$

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Clauses

A formula

$$\bigvee_{i=1}^{n} L_i$$

where L_i are literals is called **clause**.

Obviously, each CNF is a conjunction of clauses and determined up to equivalence by its set of clauses

$$\text{clauses}(F) := \{\bigvee_{j=1}^{m_i} L_{i,j} \,|\, i = 1, \ldots, n\}$$

Clauses are often written as sets of literals:

$$\bigvee_{i=1}^{n} L_i \rightsquigarrow \{L_1, L_2, \ldots, L_n\}$$

Resolvents

Let F and G be two clauses (described by sets of literals) with $A \in F$ and $\neg A \in G$. Then the clause

$$(F \setminus \{A\}) \cup (G \setminus \{\neg A\})$$

is called **a resolvent of F and G**.

Example:
Let

$$F := \{A, \neg B, C\}, \quad G := \{B, \neg C, D\}$$

then a resolvent of F and G is

$$(\{A, \neg B, C\} \setminus \{C\}) \cup (\{B, \neg C, D\} \setminus \{\neg C\}) = \{A, \neg B, B, D\}$$

Resolvents are not unique, e.g., another resolvent of F and G is

$$(\{A, \neg B, C\} \setminus \{\neg B\}) \cup (\{B, \neg C, D\} \setminus \{B\}) = \{A, C, \neg C, D\}$$

Resolvents can be derived

Lemma 5. *Resolvents can be derived.*
More detailed: let F and G be two clauses and R a resolvent of F and G, then

$$\{F, G\} \vdash R$$

Resolvents can be derived

Proof. Let

$$F = F' \vee A, \quad G = G' \vee \neg A, \quad R = F' \vee G'$$

Then

1.	$\{F' \vee A, G' \vee \neg A, \neg A\}$	$\vdash F' \vee A$	assumption
2.	$\{F' \vee A, G' \vee \neg A, \neg A\}$	$\vdash \neg A$	assumption
3.	$\{F' \vee A, G' \vee \neg A, \neg A\}$	$\vdash F'$	$\vee$-elimination applied to 1 and 2
4.	$\{F' \vee A, G' \vee \neg A, \neg A\}$	$\vdash F' \vee G'$	$\vee$-introduction applied to 3
5.	$\{F' \vee A, G' \vee \neg A, \neg\neg A\}$	$\vdash G' \vee \neg A$	assumption
6.	$\{F' \vee A, G' \vee \neg A, \neg\neg A\}$	$\vdash \neg\neg A$	assumption
7.	$\{F' \vee A, G' \vee \neg A, \neg\neg A\}$	$\vdash G'$	$\vee$-elimination applied to 5 and 6
8.	$\{F' \vee A, G' \vee \neg A, \neg\neg A\}$	$\vdash G' \vee F'$	$\vee$-introduction applied to 7
9.	$\{F' \vee A, G' \vee \neg A, \neg\neg A\}$	$\vdash F' \vee G'$	$\vee$-symmetry applied to 8
10.	$\{F' \vee A, G' \vee \neg A\}$	$\vdash F' \vee G'$	proof by cases applied to 4 and 9

□

Resolution Proof System

Let F be a CNF. We define

$$\begin{aligned}\text{Res}^0(F) :=& \text{clauses}(F)\\ \text{Res}^i(F) :=& \text{Res}^{i-1}(F)\\ &\cup\{R \mid R \text{ is a resolvent of two clauses } C, D \in \text{Res}^{i-1}(F)\}\end{aligned}$$

As there are only finite many clauses of a finite set of atoms, for some i we have

$$\text{Res}^i(F) = \text{Res}^{i-1}(F)$$

and call this $\text{Res}^*(F)$.

The clauses in $\text{Res}^*(F)$ can be derived by just two rules:

	premise	conclusion	name
1.	C is a clause of F	$F \vdash C$	[assumption]
2.	$F \vdash C, F \vdash D, R$ is a resolvent of C, D	$F \vdash R$	[resolution]

Soundness & Completeness

The resolution proof system claims for a CNF F:

$$F \text{ is unsatisfiable} \iff \emptyset \in \text{Res}^*(F)$$

Lemma 6. *Resolution is sound and complete.*

Proof.
sound: as resolution can be proved.

complete: not so obvious to see; see Hedmann 2004, p. 42. □

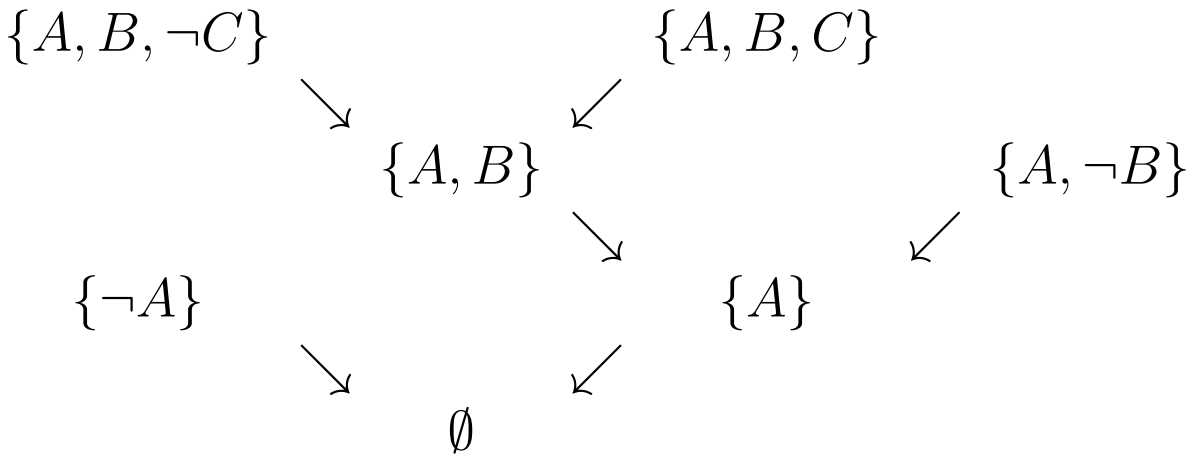
Entailment by Resolvents / Algorithm

```
1  entails(formula F, query formula Q) :
2  C := ∅
3  C' := clauses(cnf(F ∧ ¬Q))
4  while C' ≠ ∅ do
5        C := C ∪ C'
6        C' := ∅
7        for C, D ∈ C do
8              R := {R | R is a resolvent of C, D}
9              if ∅ ∈ R return true fi
10             C' := C' ∪ R
11       od
12 od
13 return false
```

Entailment by Resolvents / Example

Is the following formula satisfiable?

$$F := (A \vee B \vee \neg C) \wedge \neg A \wedge (A \vee B \vee C) \wedge (A \vee \neg B)$$



Therefore, F is unsatisfiable.

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Horn Formulas

A CNF F is called **Horn formula** if
each element of the conjunction contains at most one positive literal, i.e.

$$F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} L_{i,j}$$

where $L_{i,j}$ are literals and for each i there is at most one j with $L_{i,j}$ positive.

Example:

$A \wedge (\neg A \vee \neg B \vee C) \wedge (\neg B \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee \neg D)$ is Horn
$A \vee B$ is not Horn

Different from CNF, not every formula has an equivalent Horn formula.
Example: $A \vee B$

Basic Horn Formulas

A formula

$$\bigvee_{i=1}^{n} L_i$$

where L_i are literals and at most one is positive, is called **Horn clause** or **Basic Horn formula**.

Obviously, each Horn formula is a conjunction of Horn clauses and determined up to equivalence by its sets of clauses

$$\text{clauses}(F) := \{\bigvee_{j=1}^{m_i} L_{i,j} \,|\, i = 1, \ldots, n\}$$

Example:

$A \wedge (\neg A \vee \neg B \vee C) \wedge (\neg B \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee \neg D)$ is Horn, but not basic Horn

$(\neg A \vee \neg B \vee C)$ is basic Horn

Basic Horn Formulas as Implications

Basic Horn formulas can be written as implications of atoms/positive literals:

$$(\bigvee_{i=1}^{n} \neg L_i \vee M) \equiv (\bigwedge_{i=1}^{n} L_i \rightarrow M)$$

$$(\bigvee_{i=1}^{n} \neg L_i) \equiv (\bigwedge_{i=1}^{n} L_i \rightarrow \text{false})$$

$$M \equiv (\text{true} \rightarrow M)$$

where L_i and M are positive literals.

Example:

$$A \wedge (\neg A \vee \neg B \vee C) \wedge (\neg B \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee \neg D)$$
$$\equiv (\text{true} \rightarrow A) \wedge (A \wedge B \rightarrow C) \wedge (B \rightarrow C) \wedge (B \rightarrow D) \wedge (C \wedge D \rightarrow \text{false})$$

For an implication $L \rightarrow M$, we write

$$\text{body}(L \rightarrow M) := L$$
$$\text{head}(L \rightarrow M) := M$$

Entailment for Horn Formulas

The entailment problem for Horn formulas can be solved efficiently, in linear time in the number of clauses.

Idea: build assignment incrementally using clauses for inference:

1. all atoms that occur in clauses without premise are assigned true,
2. for all clauses where all premises are already assigned true, assign also the head true.
3. once the query atom is assigned true, terminate and return true.
4. if there are no more clauses with all premises assigned true, terminate and return false.

Satisfiability:
Horn formulas without a clauses with head false always are satisfiable.
Horn formulas with some clauses with head false are unsatisfiable, if the atom false is entailed.

Entailment for Horn Formulas / Example

Is the following Horn formula satisfiable?

$$(\text{true} \rightarrow A) \wedge (A \wedge B \rightarrow C) \wedge (C \rightarrow D) \wedge (C \wedge D \rightarrow \text{false}) \wedge (\text{true} \rightarrow B)$$

Can we entail the query atom false?

As there are clauses true $\rightarrow A$ and true $\rightarrow B$,
we assign A and B to true.

As now all premises of $A \wedge B \rightarrow C$ are true,
we assign C to true.

As now all premises of $C \rightarrow D$ are true,
we assign D to true.

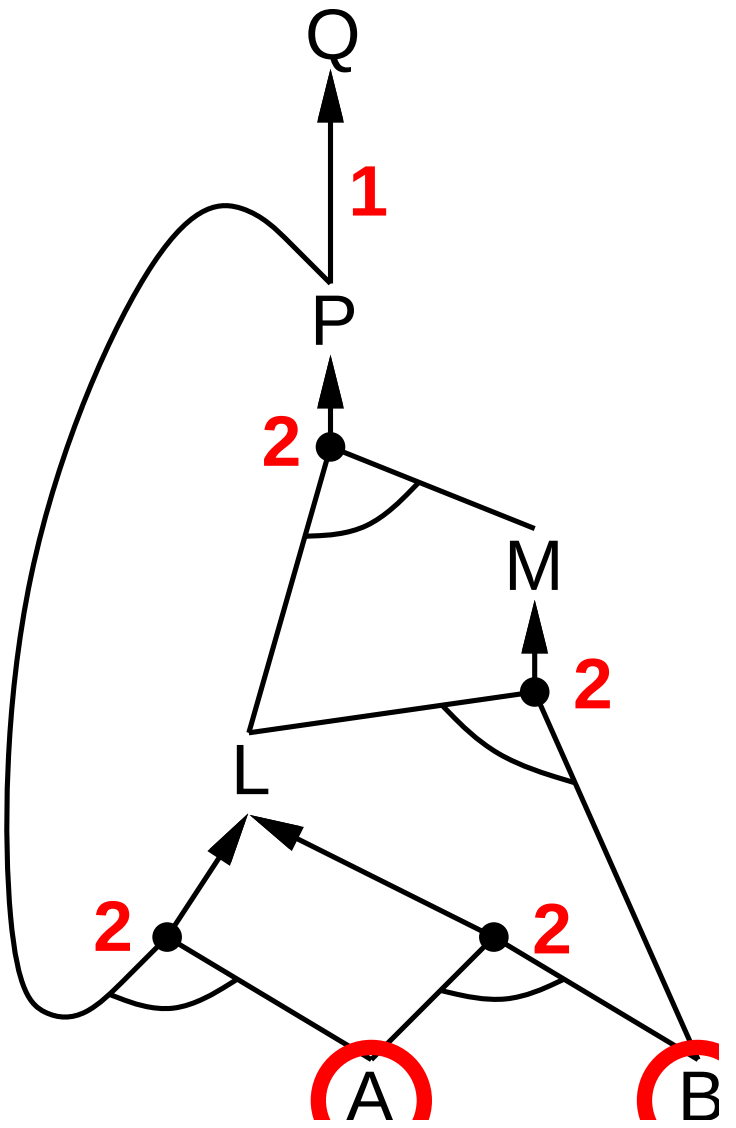
As now all premises of $C \wedge D \rightarrow$ false are true,
we entail the query atom false.

Therefore, the formula is not satisfiable.

And/Or Graphs

Horn clauses:

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



Entailment for Horn Formulas / Algorithm

```
1  entail(horn formula F, query atom Q) :
2  S := atoms(F)
3  A(A) := false,    ∀A ∈ S
4  C := clauses(F)
5  p(C) := number of premises in body(C),    ∀C ∈ C
6  H := {head(C) | C ∈ C : body(C) = ∅}
7  while H ≠ ∅ do
8          A := remove-element(H)
9          if A = Q return true fi
10         if A(A) = false
11           A(A) := true
12           for C ∈ C : A occurs in body(C) do
13               p(C) := p(C) − 1
14               if (p(C) = 0)
15                 append(H, head(C))
16               fi
17           od
18         fi
19 od
20 return false
```

Entailment for Horn Formulas / Soundness & Completeness

Lemma 7. *The inference algorithm for Horn formulas is sound and complete.*

Proof.
sound, i.e., every derived atom is entailed:
in each step only modus ponens / $\rightarrow$-elimination rule applied.

complete, i.e., every entailed atom will be derived:
$\mathcal{A}$ is an assignment for F, i.e., each clause holds:
assume $L_1 \wedge L_2 \wedge \ldots \wedge L_n \rightarrow M$ is clause that does not hold.
Then $\mathcal{A}(L_i) =$ true and $\mathcal{A}(M) =$ false. But in this case M would have been assigned true, so there cannot be a clause that does not hold.

As $\mathcal{A}$ is an assignment that models F, any atom not assigned true in $\mathcal{A}$ is not entailed by F. And all the atoms assigned true by $\mathcal{A}$ will be derived. □

1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Entailment, Satisfiability, Existence of Models

Entailment of a query formula Q by a given formula F can also be computed by **model checking**,
i.e.,

computing a model/assignment that models F but not Q
($F \wedge \neg Q$ is satisfiable)
then F does not entail Q.

show that there is no model that models F but not Q
($F \wedge \neg Q$ is not satisfiable)
then F entails Q.

Approaches

Different approaches:

incremental model building / complete enumeration
compute all assignments and check each individually.

incremental model building / use inference where possible
use inference in some easy to check situations,
otherwise enumerate all possible assignments.

local search
start with a complete random assignment,
update the truth value of some variables in each step to make less clauses false.

Use inference where possible / Early stopping

Transform the formula to check for satisfiability to CNF.

A clause is true, if one of its literals is true.
The formula is true, if each clause is true.

Example:

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

Assume partial assignment: $\mathcal{A}(B) =$ false, $\mathcal{A}(D) =$ true.

Then each clause is already true,
so we do not have to complete the assignment but can stop here.

Use inference where possible / early stopping

A clause is false, if all of its literals are false.
The formula is false, if one of its clauses is false.

Example:

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

Assume partial assignment: $\mathcal{A}(A) =$ false, $\mathcal{A}(D) =$ false.

The last clause is false,
so this partial assignment cannot be completed to a model.

Use inference where possible / pure symbols

If of some symbol A, only positive (or only negative) literals occur, then the symbol safely can be assigned true (or false).

Such symbols are called **pure**.

Example:

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

Assume empty assignment: $\mathcal{A} = \emptyset$.

A is pure as it appears only in positive form.

B is pure as it appears only in negative form.

D is pure as it appears only in positive form.

$\rightsquigarrow$ it is safe to assign $\mathcal{A}(A) =$ true, $\mathcal{A}(B) =$ false and $\mathcal{A}(D) =$ true.

Use inference where possible / unit clauses

If some clause consists only of a single literal not yet assigned to false, then the respective symbol safely can be assigned true (or false).

Such symbolsclauses called **unit clauses**.

Example:

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

Assume partial assignment: $\mathcal{A}(A) =$ false, $\mathcal{A}(C) =$ false.

The first clause consists only of the literal $\neg B$ not yet assigned to false.

$\rightsquigarrow \mathcal{A}(B) =$ false.

Davis-Putnam Algorithm

Model checking using

early stopping
(true if one literal of each clause is true,
false if all literals of one clause are false)

pure symbols
(occurs only as positive or only as negative literal)

unit clauses
(contains only one literal not yet assigned to false)

is called **Davis-Putnam Algorithm** or **DPLL** (after its inventors Martin Davis, Hilary Putnam (1960) and Davis, Logemann and Loveland (1962)).

Davis-Putnam Algorithm

```
1  entails-dpll(formula F, query formula Q) :
2  C := clauses(cnf(F ∧ ¬Q))
3  A(A) := ∅   ∀A ∈ S(C)
4  return not satisfiable-dpll(C, A)
5
6  satisfiable-dpll(set of clauses C, partial assignment A) :
7  if ∀C ∈ C∃A ∈ C : A(A) = true return true fi
8  if ∃C ∈ C∀A ∈ C : A(A) = false return false fi
9
10 if ∃A ∈ A^-1(∅) : ∀C ∈ C : A ∈ C or ¬A ∉ C                    [pure symbol]
11   return satisfiable.dpll(C, A ∪ {A ↦ true})
12 elsif ∃A ∈ A^-1(∅) : ∀C ∈ C : ¬A ∈ C or A ∉ C
13       return satisfiable-dpll(C, A ∪ {A ↦ false})
14 fi
15
16 if ∃C ∈ C, A ∈ A^-1(∅) : A ∈ C and ∀B ∈ C, B ≠ A : A(B) = false    [unit clause]
17   return satisfiable-dpll(C, A ∪ {A ↦ true})
18 elsif ∃C ∈ C, AA^-1(∅) : ¬A ∈ C and ∀B ∈ C, B ≠ ¬A : A(B) = false
19       return satisfiable-dpll(C, A ∪ {A ↦ false})
20 fi
21
22 A = random(A^-1(∅))
23 return satisfiable-dpll(C, A ∪ {A ↦ true}) | satisfiable-dpll(C, A ∪ {A ↦ false})
```

Local Search: WalkSat

Local search algorithm for deciding satisfiability

- start with a random complete assignment,
- stop when all clauses are satisfied,
- otherwise pick an unsatisfied clause C and
  - either flip the assignment of the symbol that maximizes the number of satisfied clauses (min-conflicts step)

    $A := \text{argmax}_{A \in \mathcal{S}(C)} |\{C' \in \mathcal{C} \mid \mathcal{A}|_{\mathcal{S} \setminus \{A\}} \cup \{A \mapsto 1 - \mathcal{A}(A)\} \models C'\}|$
  - or flip a random symbol of C.

Cannot prove insatisfiability,
just indicates that no solution may exist
as it could not be computed in time.

WalkSat Algorithm

satisfiable-walksat(set of clauses $\mathcal{C}$, random walk prob. $p \in [0,1]$, maxSteps) :
$\mathcal{A} :=$ random assignment of $\mathcal{S}(\mathcal{C})$
for $i := 1 \ldots$ maxSteps **do**
    if $\forall C \in \mathcal{C} : \mathcal{A}(C) =$ true **return** true **fi**
    $C :=$ random$\{C \in \mathcal{C} \mid \mathcal{A}(C) =$ false$\}$
    if random$() > p$
        $A := \text{argmax}_{A \in \mathcal{S}(C)} |\{C' \in \mathcal{C} \mid \mathcal{A}|_{\mathcal{S}\setminus\{A\}} \cup \{A \mapsto 1 - \mathcal{A}(A)\} \models C'\}|$
    else
        $A :=$ random $\mathcal{S}(C)$
    fi
    $\mathcal{A}(A) := 1 - \mathcal{A}(A)$
od
return failure

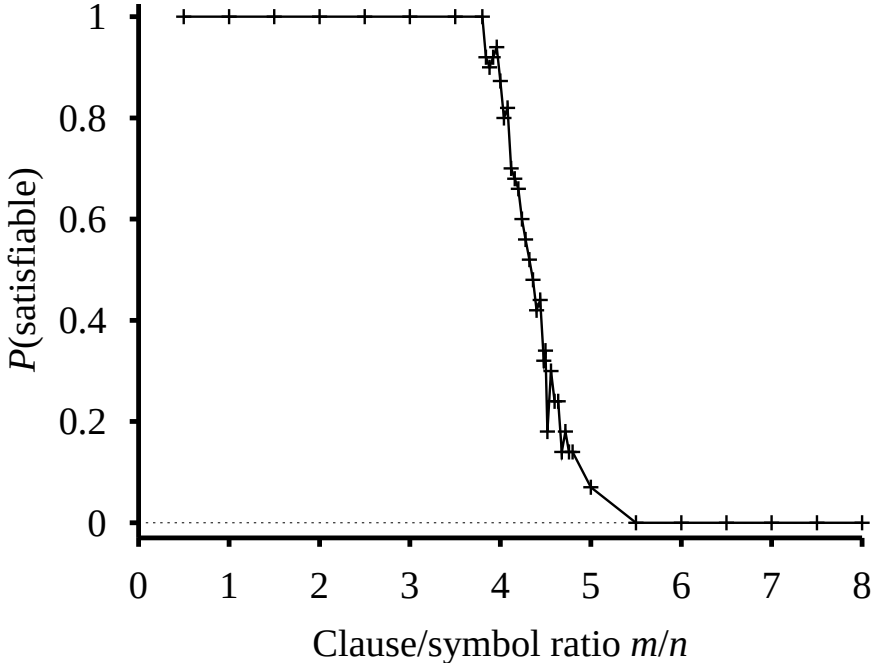
Hard Satisfibility Problems

3-CNF: CNF where each clause consists of exactly 3 literals.

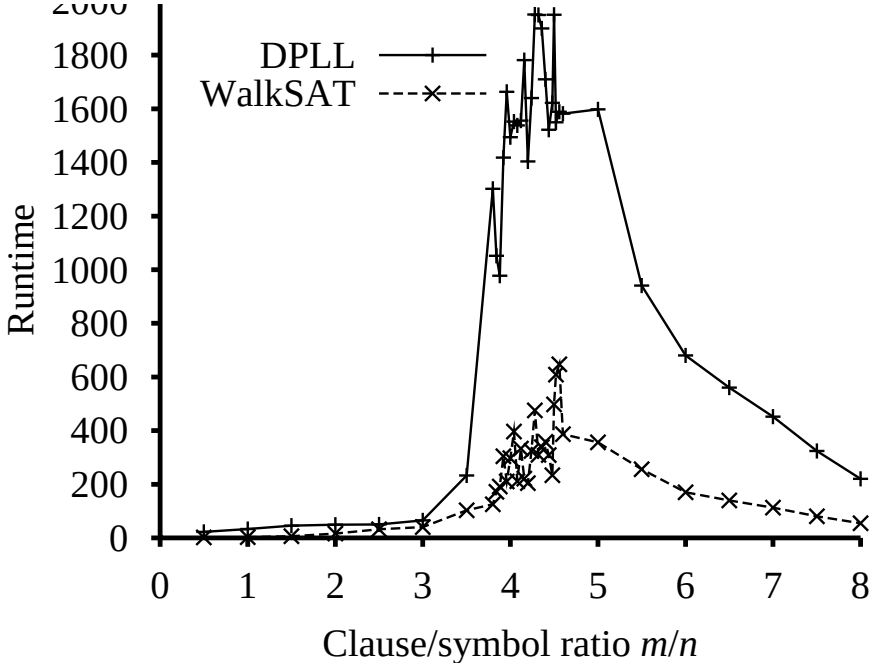
The **satisfiability problem (SAT)** for 3-CNF formulas is called **3-SAT** and still NP-complete.

The more clauses per symbol a SAT problem has,
the more models may be ruled out
and thus the less likely it is satisfiable.

Hard Satisfibility Problems



Probability of a random 3-CNF with 50 symbols to be satisfiable.



Median runtime of DPLL and WalkSat for 100 random satisfiable 3-CNFs with 50 symbols.

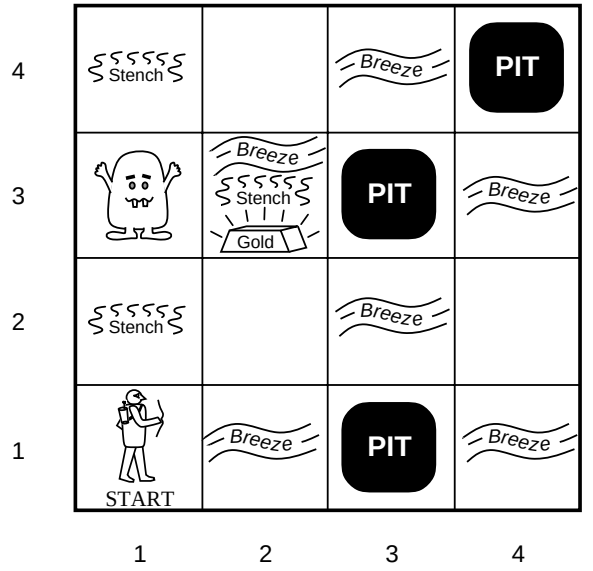
1. Syntax
2. Semantics
3. Formal Proofs
4. Normal Forms
5. Resolution
6. Propositional Horn Formulas
7. Entailment by Model Checking
8. A Silly Example

Wumpus World

Toy example by Gregory Yob (1975), adapted by our textbook.

- 4×4 grid, tiles numbered (1,1) to (4,4),
- the agent starts in (1,1),
- the beast Wumpus sits at a random tile, unknown to the agent,
- a pile of gold sits at another random tile, unknown to the agent,
- some pits are located at random tiles, unknown to the agent.

- if the agent enters the tile of the Wumpus, he will be eaten,
- if the agent enters a pit, he will be trapped,

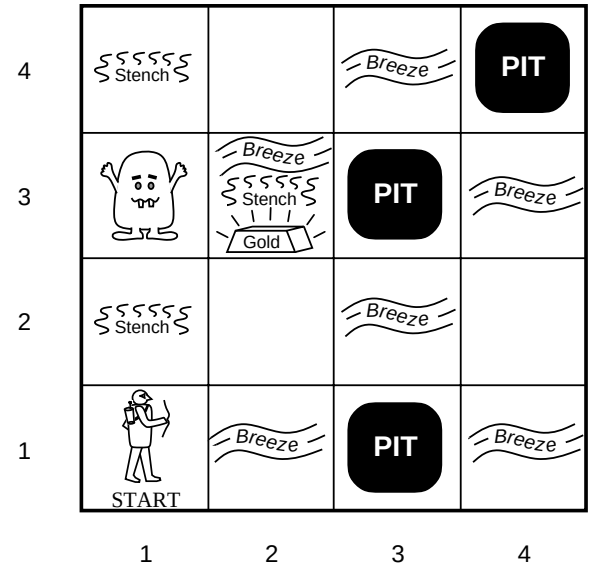


Wumpus World

Toy example by Gregory Yob (1975), adapted by our textbook.

- the agent has a single shot to shoot at a neighboring tile and kill the Wumpus if it is there,
- the agent can pick up the pile of gold if it is at the same tile,
- the agent can move to neighboring tiles,

- the agent perceives a breeze at neighboring tiles of pits,
- the agent perceives stench at neighboring tiles of the Wumpus.



Physics

64 variables:

$P_{x,y}$ tile x, y contains a pit ($x, y = 1, \ldots, 4$).
$W_{x,y}$ tile x, y contains the Wumpus ($x, y = 1, \ldots, 4$).
$B_{x,y}$ tile x, y contains a breeze ($x, y = 1, \ldots, 4$).
$S_{x,y}$ tile x, y contains stench ($x, y = 1, \ldots, 4$).

start is save: (2 formulas)

$$\neg P_{1,1}, \quad \neg W_{1,1}$$

how breeze arises: (16 formulas)

$$B_{x,y} \leftrightarrow P_{x-1,y} \vee P_{x+1,y} \vee P_{x,y-1} \vee P_{x,y+1}, \quad x, y = 1, \ldots, 4$$

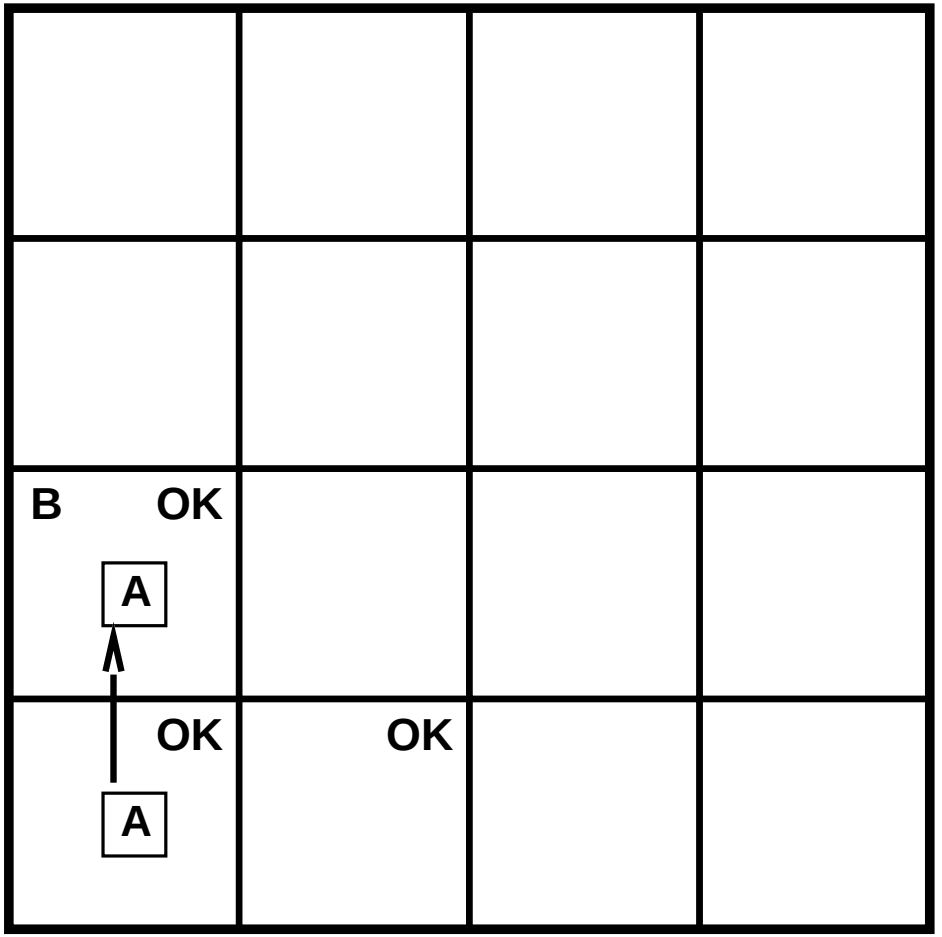
how stench arises: (16 formulas)

$$S_{x,y} \leftrightarrow W_{x-1,y} \vee W_{x+1,y} \vee W_{x,y-1} \vee W_{x,y+1}, \quad x, y = 1, \ldots, 4$$

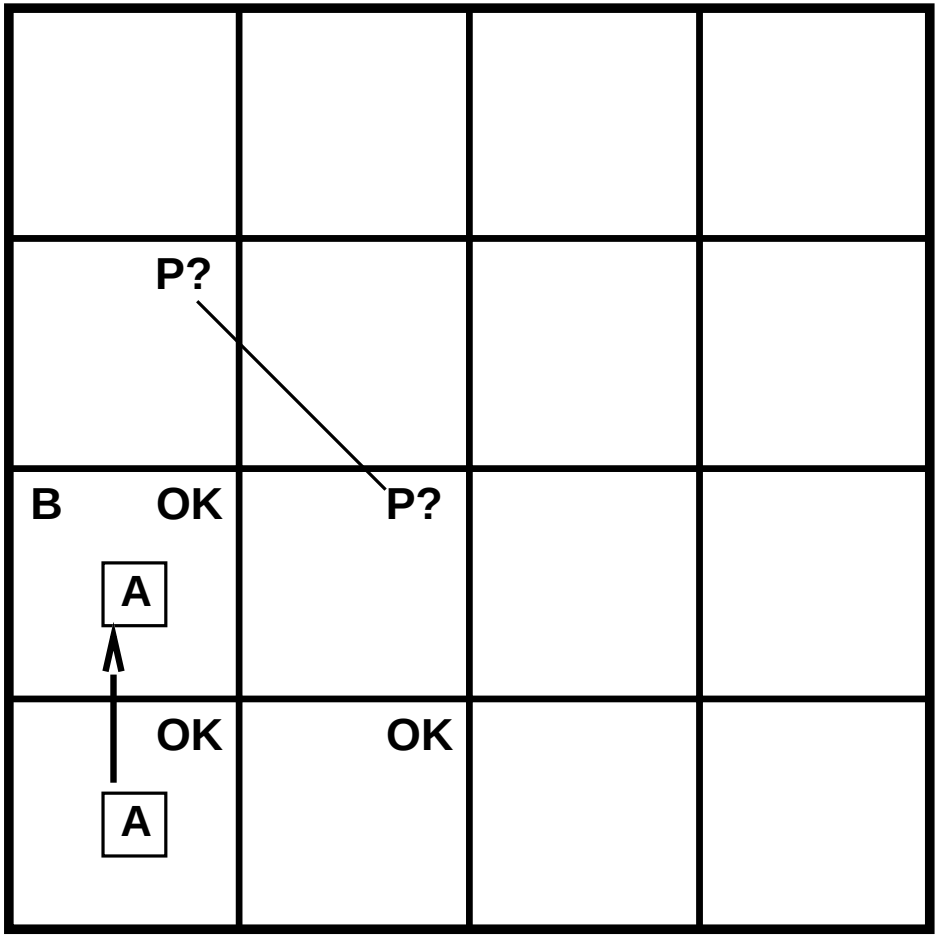
there is exactly one Wumpus: (121 formulas)

$$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$$
$$\neg W_{x,y} \vee \neg W_{x',y'}, \quad x, y, x', y' = 1, \ldots, 4, x \neq x' \text{ or } y \neq y'$$

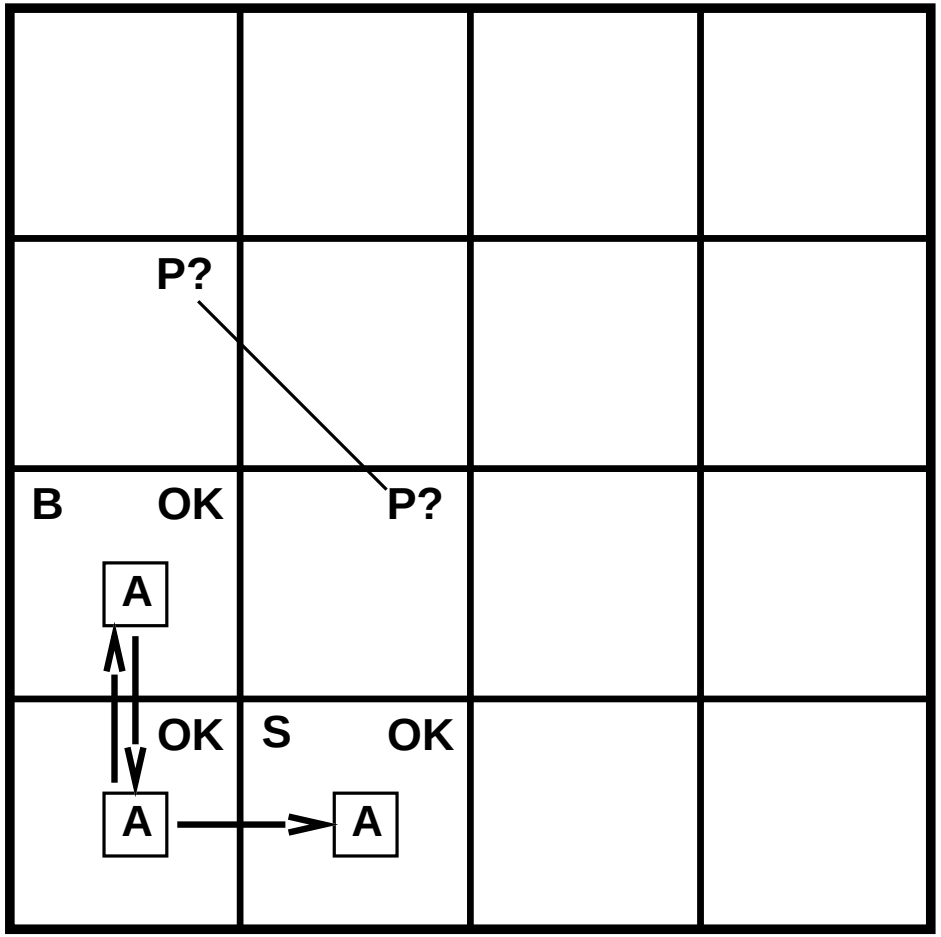
Explore the Wumpus World



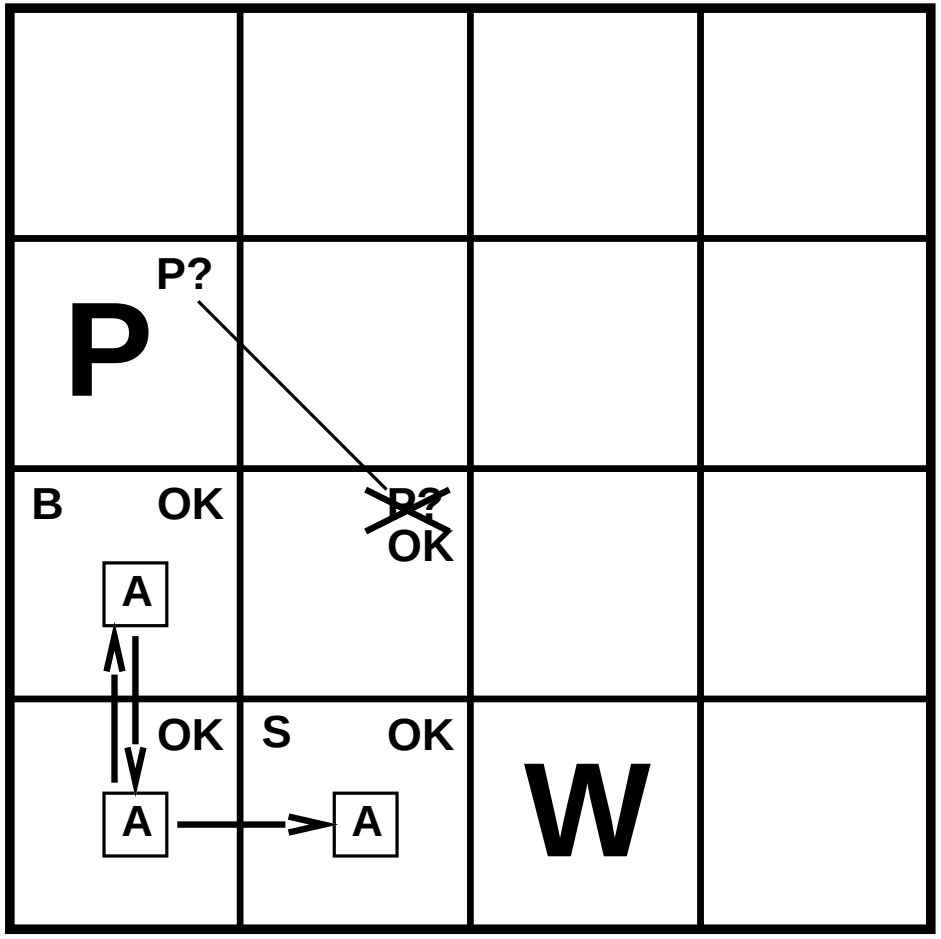
Explore the Wumpus World



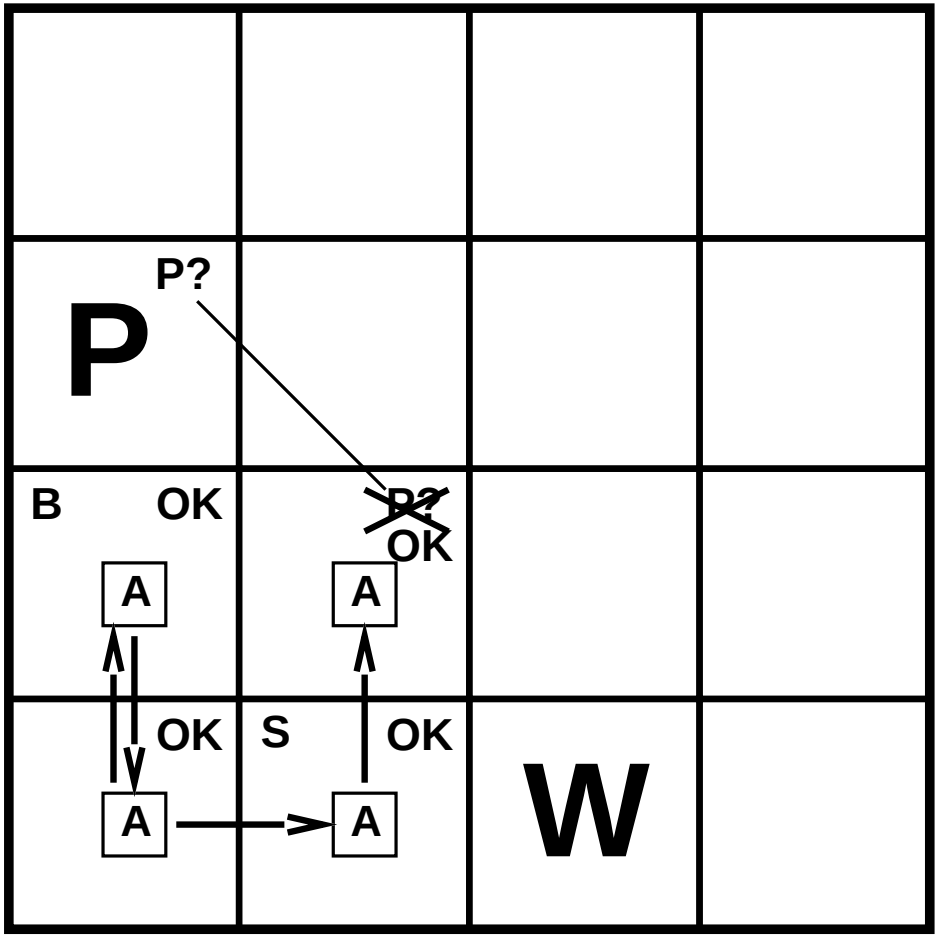
Explore the Wumpus World



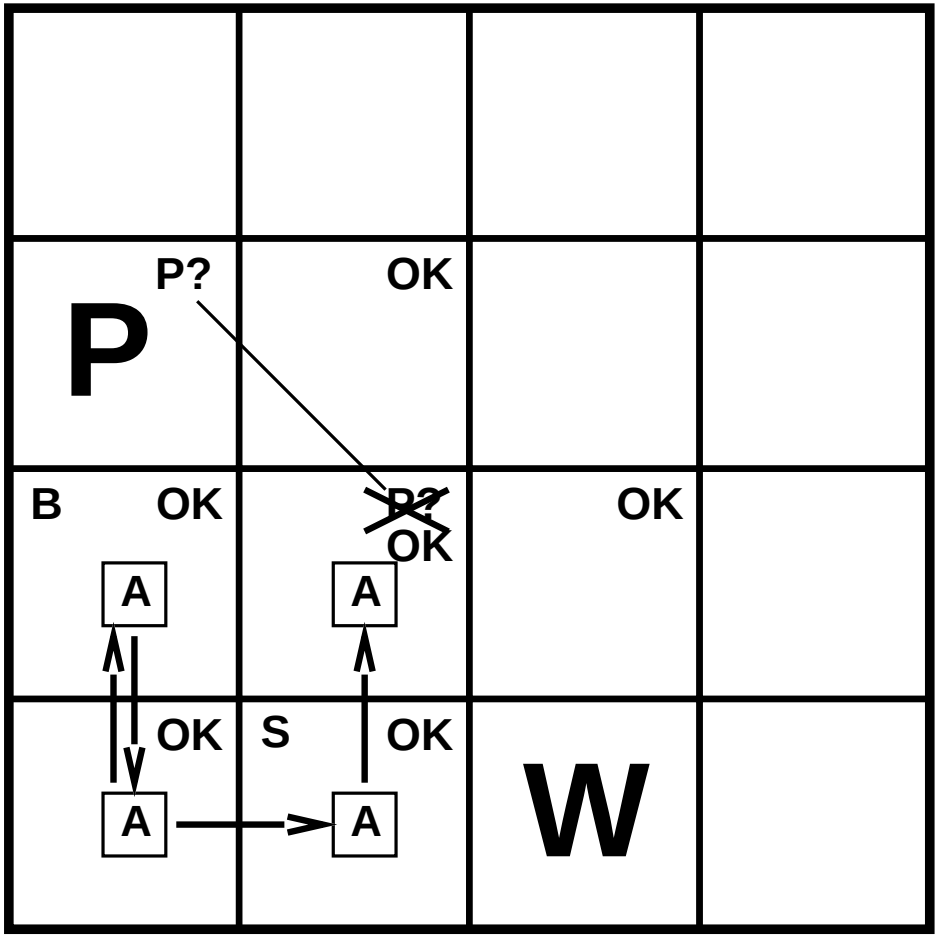
Explore the Wumpus World



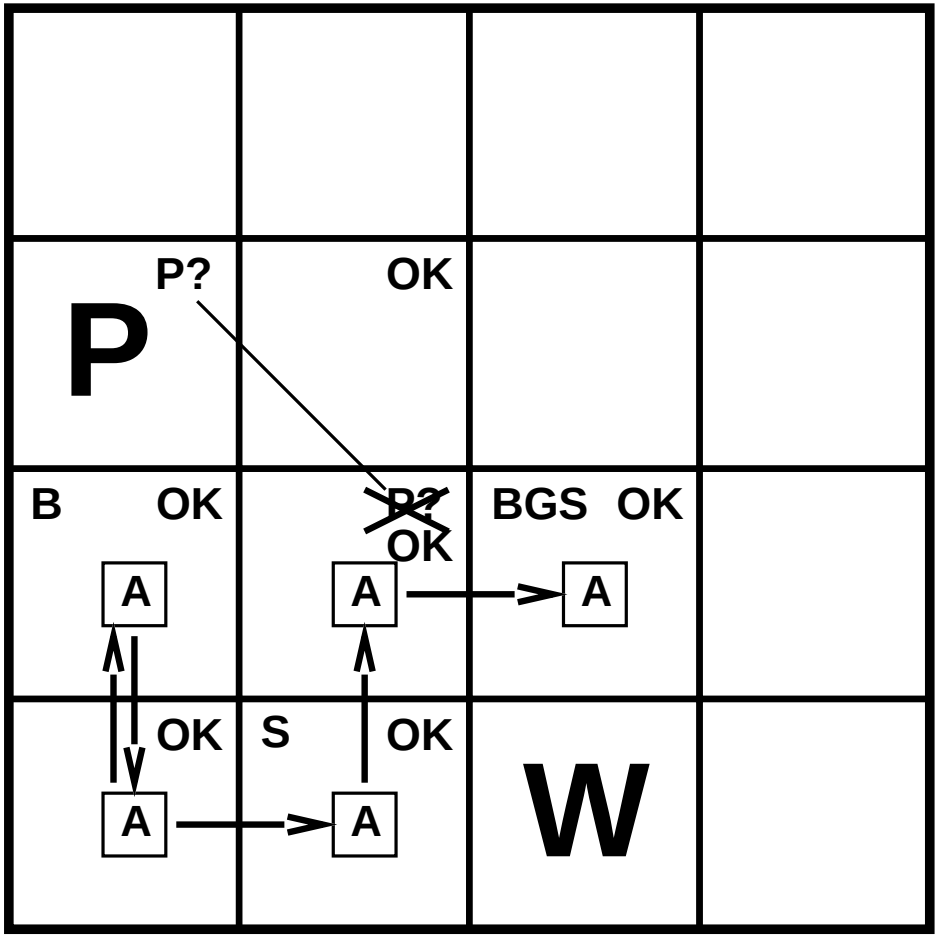
Explore the Wumpus World



Explore the Wumpus World



Explore the Wumpus World



Wumpus World

Perceptions are **told** / added to the knowledgebase.

Consequences as conditions for actions can then be **asked** from the knowledgebase, e.g.,

- tile x, y is provably safe if

  $$\neg W_{x,y} \wedge \neg P_{x,y}$$

  is entailed by the knowledgebase.

- if there are no provably safe tiles,
  choose among tiles x, y that may or may not be dangerous,
  i.e., where

  $$W_{x,y} \vee P_{x,y}$$

  is **not** entailed.

Beyond Wumpus World, have a look at General Game Playing (http://games.stanford.edu).

Wumpus World

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*,*breeze*,*glitter*]
  static: *KB*, a knowledge base, initially containing the "physics" of the wumpus world
    x, y, *orientation*, the agent's position (initially [1,1]) and orientation (initially *right*)
    visited, an array indicating which squares have been visited, initially *false*
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

  update x,y,*orientation*, *visited* based on *action*
  if *stench* **then** TELL($KB, S_{x,y}$) **else** TELL($KB, \neg\, S_{x,y}$)
  if *breeze* **then** TELL($KB, B_{x,y}$) **else** TELL($KB, \neg\, B_{x,y}$)
  if *glitter* **then** *action* $\leftarrow$ *grab*
  else if *plan* is nonempty then *action* $\leftarrow$ POP(*plan*)
  else if for some fringe square [i,j], ASK($KB, (\neg\, P_{i,j} \wedge \neg\, W_{i,j})$) is *true* **or**
      for some fringe square [i,j], ASK($KB, (P_{i,j} \vee W_{i,j})$) is *false* **then do**
    plan $\leftarrow$ A*-GRAPH-SEARCH(ROUTE-PROBLEM([x,y], *orientation*, [i,j],*visited*))
    action $\leftarrow$ POP(*plan*)
  else *action* $\leftarrow$ a randomly chosen move
  return *action*