Propositional Logic

Propositional Logic

The elements of the language:

Atoms: Two distinguished atoms T and F and the countably infinite set of those strings of characters that begin with a capital letter, for example, P, Q, R, . . ., P1, Q1, ON_A_B, etc.

Connectives: ∧, ∨,⟶, and, ¬, called "and", "or", "implies", and "not".

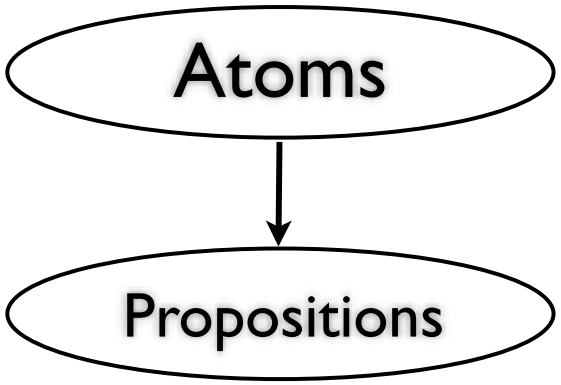
Syntax of well-formed formulas (wffs), also called ***sentences***:

- Any atom is a wff
- if ω1, ω2 are wffs, so are
- ω1 ∧ ω2 (conjunction)
- ω1 ∨ ω2 (disjunction)
- ω1⟶ω2 (implication)
- ¬ω1 (negation)

Parentheses will be used extra-linguistically grouping wffs into sub wffs according to recursive defs

Semantics

What do sentences *mean*?

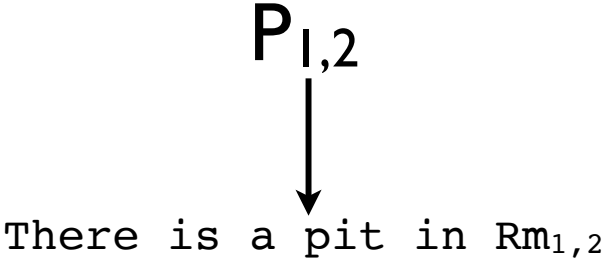


Semantics is about associating elements of a logical language with elements of a domain of discourse.

In the case of propositional logic, the domain of discourse is **propositions** about the world.

One associates atoms in the language with propositions.

An **interpretation** associates a proposition with each atom and a value (True or False)



If atom α is associated with proposition *P*, then we say that α has value True just in case *P* is true of the world; otherwise it has value False

The Truth Table Method

Truth tables can be used to compute the truth value of any **wff** given the truth values of the constituent atoms in the formula.

ω1	ω2	ω1 ∧ ω2	ω1 ∨ ω2	¬ω1	ω1→ω2
True	True	True	True	False	True
True	False	False	True	False	False
False	True	False	True	True	True
False	False	False	False	True	True

$[(P \rightarrow Q) \rightarrow R] \rightarrow P$

P is False
Q is False
R is True

Interpretation

If an agent describes its world using *n* features (corresponding to propositions) and these features are represented as **n** atoms in the agent's model of the world then there are 2^n ways the world can be as far as the agent can discern/express.

Satisfiability and Models

An interpretation **satisfies** a wff if the wff is assigned the value True under the interpretation.

An interpretation that satisfies a wff (set of wffs) is called a **model** of the wff (set of wffs).

Find an interpretation that is a model of: $P_{1,2} \vee W_{1,2} \rightarrow \neg OK_{1,2}$

A wff is said to be **inconsistent** or **unsatisfiable** if there are no interpretations that satisfy it. (Likewise for sets of sentences)

$P_{1,2} \wedge \neg P_{1,2}$ $\{P_{1,2} \vee W_{1,2}, P_{1,2} \vee W_{1,2}, \neg P_{1,2} \vee W_{1,2}, \neg P_{1,2} \vee \neg W_{1,2}\}$

Validity and Entailment

A wff is said to be **valid** if it has value True under all interpretations of its constituent atoms.

Are the following valid sentences? $\neg(P_{1,2} \wedge \neg P_{1,2})$ $\neg(P_{1,2} \wedge \neg W_{1,2})$

If a wff ω has value True under all those interpretations for which each of the wffs in a set Δ has value True, then we say that Δ **logically entails** ω and that ω logically follows from Δ and that ω is a logical consequence of Δ. We use the symbol $\models$ to denote **logical entailment** and write $\Delta \models \omega$

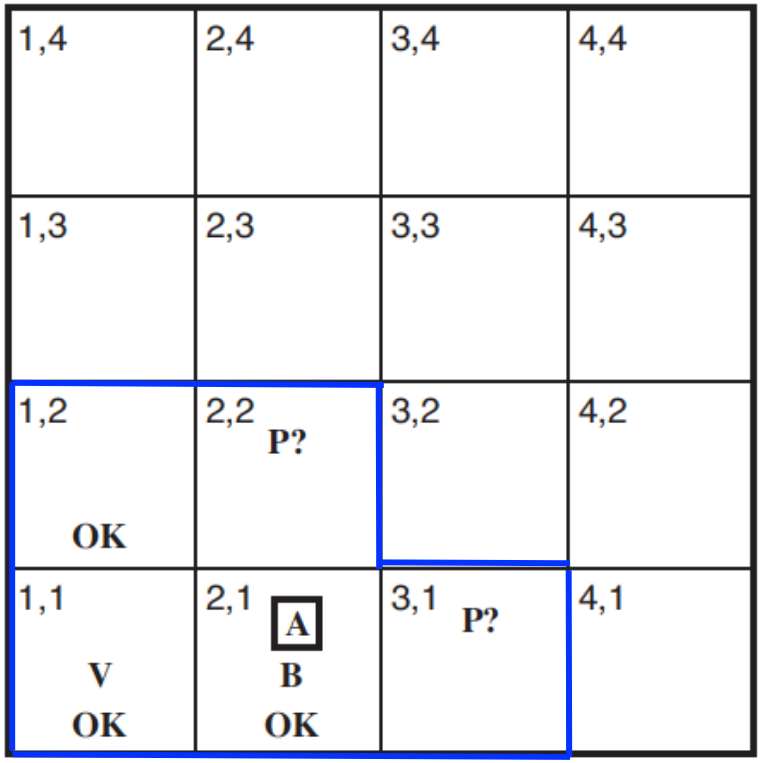
$\{P_{1,2}\} \vDash P_{1,2}$ $\{\} \vDash \neg(P_{1,2} \wedge \neg P_{1,2})$

$\{P_{1,2}, P_{1,2} \rightarrow W_{1,2} \vDash {}_{1,2}$

$F \vDash \omega$ where ω is any wff!

Require an efficient means of testing whether sentences are True in an interpretation and whether sentences are entailed by sets of sentences.

An Entailment Example



Lets restrict ourselves to the blue cells:
[1,1], [2,1], [3,1],[1,2],[2,2]

We want to reason about PITS in:
[1,2],[2,2],[3,1]

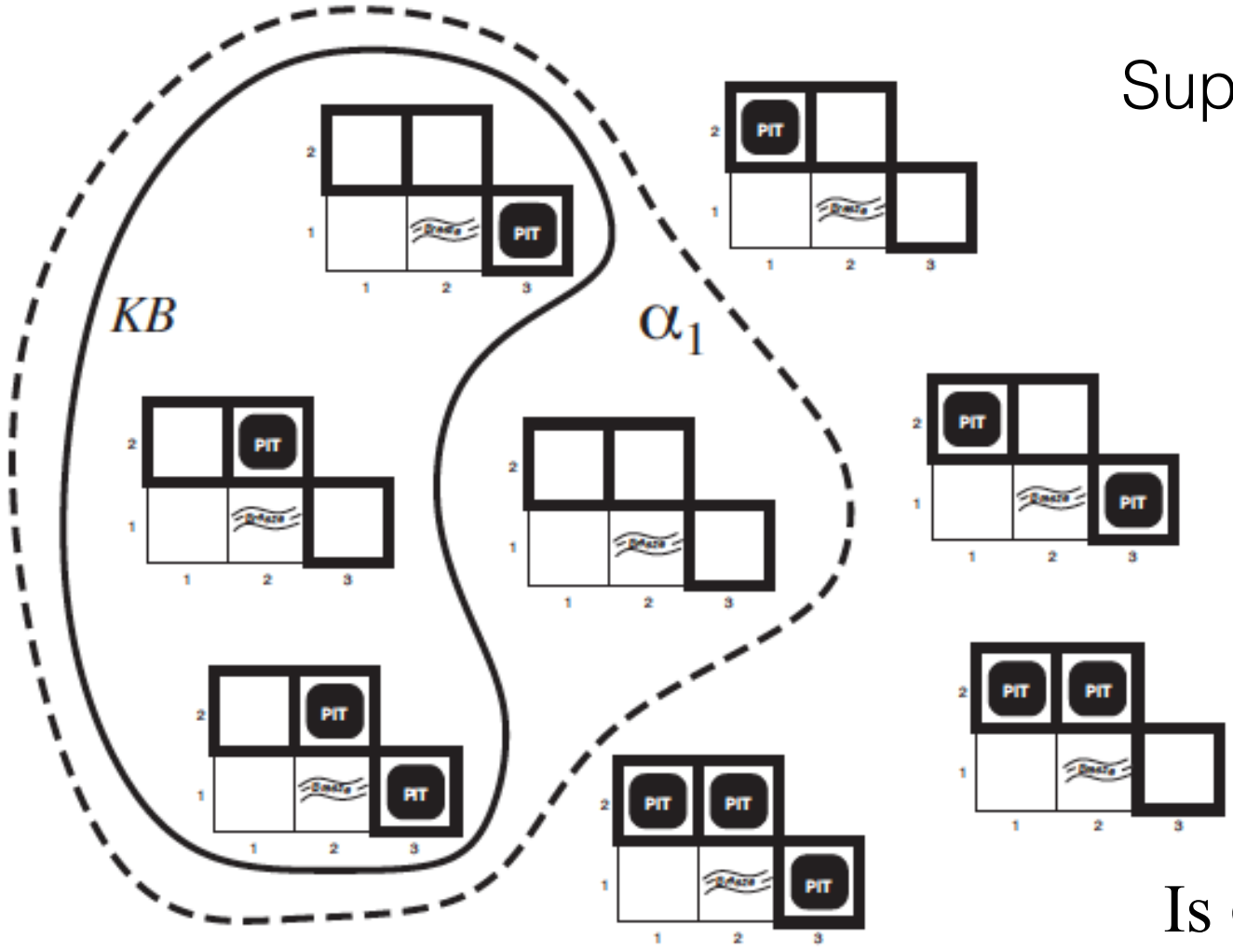
There are 8 possibilities: pit or no pit. Consequently, 8 possible models for the presence/non-presence of pits

But our percepts together with the rules of the game restrict us to three possible models satisfying the KB

$$\neg B_{1,1}, \neg S_{1,1}, OK_{1,1}, OK_{1,2}, OK_{2,1}, B_{2,1}, P_{2,2} \vee P_{3,1}$$

Rules of The game: $OK_{x,y} \rightarrow \neg P_{x,y}$ .... $OK_{x,y} \rightarrow \neg S_{x,y}$

Wumpus Possible Worlds



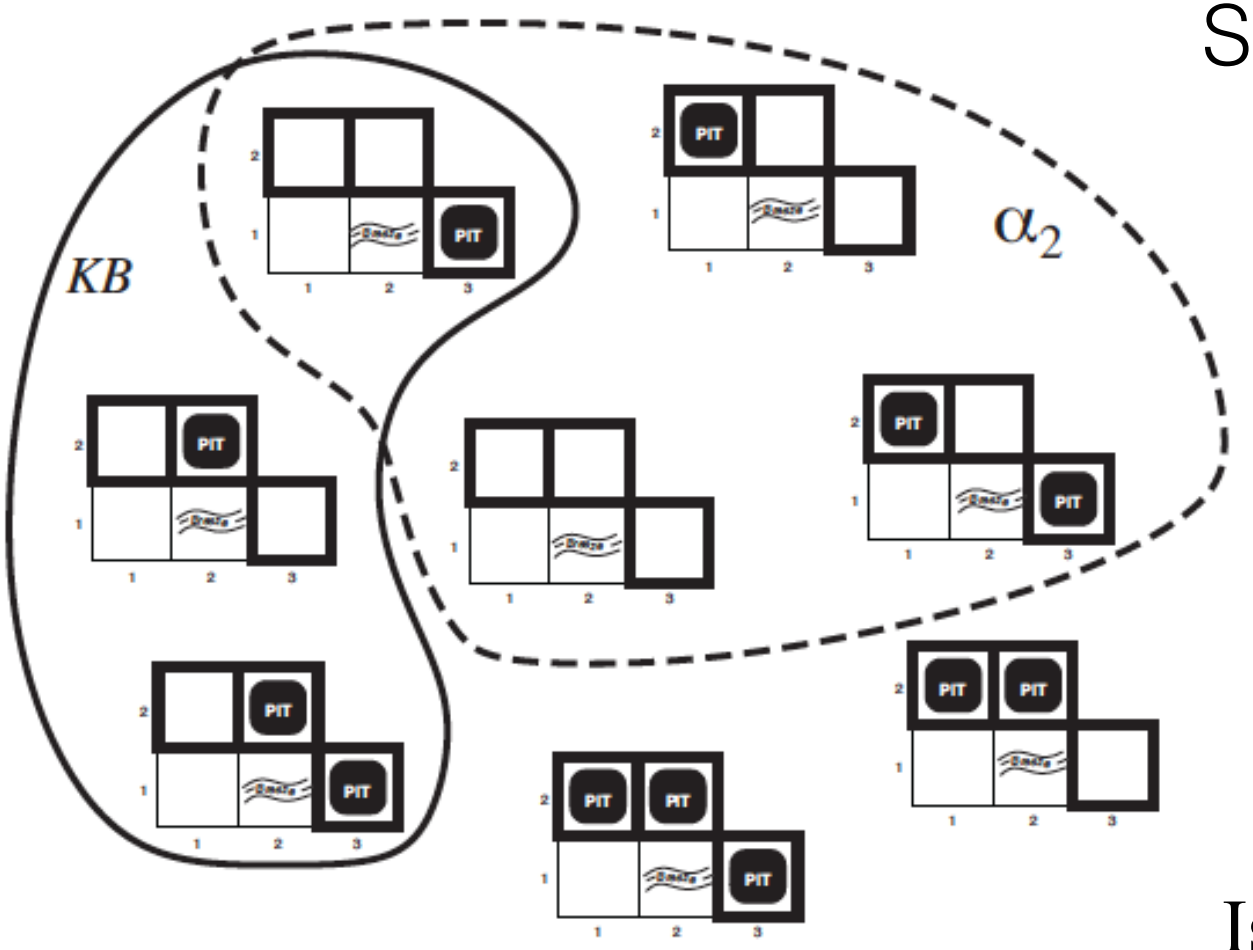
Suppose:

$$\alpha_1 = \neg P_{1,2}$$

Is α_1 entailed by KB?

YES! $KB \models \alpha_1$

Wumpus Possible Worlds



Suppose:

$$\alpha_2 = \neg P_{2,2}$$

Is α_2 entailed by KB?

NO!

$$KB \models \alpha_2$$

Truth Table Enumeration

- Enumerate all models
- Check that the query is is true in all models that satisfy the KB

Entailment checking by enumeration
Model checking approach

- Recursively build tree where each leaf is a model.
- Check that:
- Each model that makes KB true, makes query true.

function TT-ENTAILS?(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $symbols \leftarrow$ a list of the proposition symbols in KB and α
  return TT-CHECK-ALL($KB, \alpha, symbols, \{\}$)

---

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** *true* or *false*
  if EMPTY?($symbols$) **then**
    if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)
    else return *true* // *when KB is false, always return true*
  else do
    $P \leftarrow$ FIRST($symbols$)
    $rest \leftarrow$ REST($symbols$)
    return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
      and
      TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

Proof Theory

Straightforward model-checking approaches are not efficient since the number of models grows exponentially with the number of variables.

Can we find a more efficient “syntactic” means of of showing semantic consequence without the need to generate models?

We also have to “guarantee” that the syntactic approach is equivalent to the semantic approach.

For when I am presented with a false theorem, I do not need to examine or even to know the demonstration, since I shall discover its falsity a posteriori by means of an easy experiment, that is by calculation, costing no more than paper and ink, which will show the error no matter how small it is…

And if someone would doubt my results, I should say to him:

"Let us calculate, Sir", and thus by taking paper to pen and ink, we should soon settle the question

–Gottfried Wilhelm Leibniz [1677]

Rules of Inference and Proofs

Now that we have a feeling for the intuitions behind entailment and its potential, the next step is to find syntactic characterizations of the reasoning process (inference) to make this functionality feasible for use in intelligent agents. We require a **proof theory**.

Rules of inference permit us to produce additional wffs from others in a **sound** or **truth-preserving** manner.

Some Examples:

$$\frac{\omega 1,\ \omega 2}{\omega 1 \wedge \omega 2} \qquad \frac{\omega 1,\ \omega 1 \rightarrow \omega 2}{\omega 2}$$

Definition of a Proof

The sequence of wffs $\{\omega 1, \omega 2, \ldots, \omega n\}$ is called a **proof** (or **deduction**) of ω_n from a set of wffs Δ iff each ω_i in the sequence is either

- in Δ , or
- can be inferred from a wff (or wffs) earlier in the sequence by using one of the rules of inference (in the proof theory).

Proof of $Q \wedge R$ from Δ

$$\Delta = \{P, R, P \rightarrow Q\}$$

$$\{P, P \rightarrow Q, Q, R, Q \wedge R\}$$

If there is a proof of ω_n from Δ , we say that ω_n is a theorem of the set Δ . The following notation will be used for expressing that ω_n can be proved from Δ : $\Delta \vdash \omega_n$

(or $\Delta \vdash_{\mathscr{R}} \omega_n$, where $\mathscr{R}$ refers to a set of inference rules

$$\Delta \vdash Q \wedge R$$

Soundness and Completeness

If, for any set of wffs, Δ , and wff, ω, $\Delta \vdash_{\Re} \omega$ implies $\Delta \models \omega$, we say that the set of inference rules, $\Re$, is **sound**.

If, for any set of wffs, Δ , and wff, ω, it is the case that whenever $\Delta \models \omega$, there exists a proof of ω from Δ using the set of inference rules, $\Re$, we say that $\Re$ is **complete**.

Syntactic characterizations of Entailment

Soundness -- not too strong!
Completeness -- not too weak!

Some Important Meta-Theorems

The Deduction Theorem

if $\{\omega 1, \omega 2, \ldots, \omega n\} \models \omega$ then $(\omega 1 \wedge \omega 2 \wedge \ldots \omega n) \rightarrow \omega$ is valid and vice-versa.

Can transform a question of entailment into a question of validity

Reductio ad absurdum

If the set Δ has a model but $\Delta \cup \{\neg \omega\}$ does not, then $\Delta \models \omega$

Proof by Refutation: To prove that $\Delta \models \omega$, show that $\Delta \cup \{\neg \omega\}$ has no model.

Efficient Propositional Model Checking

DPPL

Clauses

A **literal** is with an atom (positive literal) or the negation of an atom (negative literal)

A **clause** is an expression of the form:

$$l_1 \vee l_2 \vee \ldots \vee l_k$$

where each l_i is a literal

A wff written as a *conjunction of clauses* is said to be in **conjunctive normal form** (**CNF**).

A wff written as a *disjunction of conjunctions of literals* is said to be in **disjunctive normal form** (**DNF**).

Any propositional formula can be converted into An equivalent CNF or DNF form

Converting to CNF or DNF form

1. Eliminate implication connectives by using the equivalent form with $\neg, \vee$.
2. Reduce the scope of $\rightarrow$ connectives by applying DeMorgan's laws and by eliminating double negations ($\neg\neg$) if they arise.
3. Convert to CNF(DNF) by using associative and distributive laws.

$$\neg(\omega_1 \vee \omega_2) \equiv \neg\omega_1 \wedge \neg\omega_2$$
$$\neg(\omega_1 \wedge \omega_2) \equiv \neg\omega_1 \vee \neg\omega_2$$

DeMorgan Laws

$$\omega_1 \wedge (\omega_2 \vee \omega_3) \equiv (\omega_1 \wedge \omega_2) \vee (\omega_1 \wedge \omega_3)$$
$$\omega_1 \vee (\omega_2 \wedge \omega_3) \equiv (\omega_1 \vee \omega_2) \wedge (\omega_1 \vee \omega_3)$$

Distributive Laws

$$(\omega_1 \wedge \omega_2) \wedge \omega_3 \equiv \omega_1 \wedge (\omega_2 \wedge \omega_3)$$
$$(\omega_1 \vee \omega_2) \vee \omega_3 \equiv \omega_1 \vee (\omega_2 \vee \omega_3)$$

Associative Laws

An Example

$$\neg(P \rightarrow Q) \vee (R \rightarrow P)$$

$\neg(\neg P \vee Q) \vee (\neg R \vee P)$	Eliminate implication connectives
$(P \wedge \neg Q) \vee (\neg R \vee P)$	Apply DeMorgan's Law
$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)$	Apply Distributive Law
$(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$	Factor (remove duplicates)

$$\neg(P \rightarrow Q) \vee (R \rightarrow P) \equiv (P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$$

CNF Form

Davis Putnam Algorithm

- The Davis-Putnam Algorithm (1960)
  - In a seminal paper, they described an effective satisfiability checking algorithm
  - Satisfiability by search
  - Takes as input a formula in **conjunctive normal form**
- The Davis, Putnam, Logeman, Loveland Algorithm (1962) **DPLL**
  - An extension of the DP algorithm with better space efficiency
- Essentially a recursive, depth-first enumeration of possible models with three improvements over **TT-ENTAILS**
  - Early Termination
  - Pure Symbol Heuristic
  - Unit Clause Heuristic
- Most modern SAT solvers are still based on ideas from **DPPL**

Some notation

A partial assignment is a mapping from a set of variables to truth values : $\varphi : V \rightarrow \{\text{true}, \text{false}\}$

An application of a partial assignment to a clause set *F* is denoted by: $\varphi * F$

It results in the clause set obtained from *F* by first removing All clauses satisfied by φ , and then removing all from the remaining clauses all literal occurrences which are falsified by φ

$\varphi : \{A : \text{true}, D : \text{false}\}$ $(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee D)$

$(\text{true} \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee \text{false})$

$(\neg B \vee \neg C) \wedge (C \vee \text{false})$

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

A partial assignment φ is a weak autarchy for *F* if: $\varphi * F \subseteq F$

If φ is a weak autarchy for F, then $\varphi * F$ is satisfiability equivalent to F

Early Termination

If A is true in an assignment then

$\varphi : \{A : \text{true}\}$ $(A \vee B \vee D) \wedge (A \vee \neg E \vee F) \wedge (A \vee G)$

$(\text{true} \vee B \vee D) \wedge (\text{true} \vee \neg E \vee F) \wedge (\text{true} \vee G)$

is true without knowing the assignment of other variables.

If A and G are false in an assignment then

$\varphi : \{A : \text{false}, G : \text{false}\}$ $(A \vee B \vee D) \wedge (A \vee \neg E \vee F) \wedge (A \vee G)$

$(A \vee B \vee D) \wedge (A \vee \neg E \vee F) \wedge (\text{false} \vee \text{false})$

$(A \vee B \vee D) \wedge (A \vee \neg E \vee F) \wedge (\text{false})$

is false without knowing the assignment of other variables.

Pure Symbol Heuristic

A “pure” symbol is a symbol that always appears with the same sign in all clauses

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

A is pure, B is pure and C is not

Assigning a pure symbol the value that makes it true with never make the original clause false

$\varphi : \{A : \text{true}\}$

$$(\text{True} \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee \text{True})$$

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

φ is a weak autarchy for F : $\varphi * F$ is satisfiability equivalent to F

φ is a weak autarchy for F : $\varphi * F$ is unsatisfiability equivalent to F

Unit Clause Heuristic

Unit clause in resolution: A clause with one literal

Unit clause in DPLL: also means clauses in which all literals but one are already assigned false by the model

$\varphi : \{A : \text{true}, B : \text{false}\}$ $(\neg A \vee B \vee C) \wedge (D \vee E) \wedge (\neg C \vee F) \wedge G$

$(\text{False} \vee \text{False} \vee C) \wedge (D \vee E) \wedge (\neg C \vee F) \wedge G$

For a unit clause to be true, it must have one assignment.

Unit Clause Heuristic: Assign all such symbols before branching on the remainder

Example

$\varphi : \{A : \text{true}, B : \text{false}\}$ $(\neg A \vee B \vee C) \wedge (D \vee E) \wedge (\neg C \vee \neg F) \wedge G$

$(\text{false} \vee \text{false} \vee C) \wedge (D \vee E) \wedge (\neg C \vee \neg F) \wedge G$

$C \wedge (D \vee E) \wedge (\neg C \vee \neg F) \wedge G$

$\varphi : \{A : \text{true}, B : \text{false}, G : \text{true}\}$

$C \wedge (D \vee E) \wedge (\neg C \vee \neg F) \wedge \text{true}$

$C \wedge (D \vee E) \wedge (\neg C \vee \neg F)$

$\varphi : \{A : \text{true}, B : \text{false}, G : \text{true}, C : \text{true}\}$

$C \wedge (D \vee E) \wedge (\neg C \vee \neg F)$

$\text{true} \wedge (D \vee E) \wedge (\text{false} \vee \neg F)$

$(D \vee E) \wedge \neg F$

$\varphi : \{A : \text{true}, B : \text{false}, G : \text{true}, C : \text{true}, F : \text{false}\}$

$(D \vee E) \wedge \text{true}$

$(D \vee E)$

The DPPL Algorithm

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*

Detects early termination for partially completed models

  $P, value \leftarrow$ 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($clauses, rest, model \cup \{P{=}false\}$))

Splitting Rule

Provides a skeleton of the search process.

Using DPPL for Inference

Want to know whether: $\Delta \models \alpha$

Want to turn this into a satisfiability problem!

<u>Deduction Theorem:</u> If $\Delta \models \alpha$ then $\models \Delta \rightarrow \alpha$

$\Delta \rightarrow \alpha$ is valid iff $\neg(\Delta \rightarrow \alpha) = \Delta \wedge \neg\alpha$ is unsatiafiable

Let β be $\Delta \wedge \neg\alpha$ in CNF form

If DPPL-Satisfiable?(β) is true then $\Delta \models \alpha$ is false

If DPPL-Satisfiable?(β) is false then $\Delta \models \alpha$ is true

Recent Extensions to DPPL

- *Component Analysis*
  - Find independent subsets of unassigned variables (components) and solve each component separately
- *Variable and Value Ordering*
  - degree heuristic - choose a variable appearing most frequently among remaining clauses
  - choose true or false as an assignment heuristically
- *Intelligent backtracking*
  - Also do conflict clause learning
- *Random restarts*
  - If little progress in extending an assignment, random restart
    - remember clauses assigned, change variable and value selection
- *Clever indexing techniques*
  - acquiring clause types rapidly…

Axiomatizing the Wumpus World

Physics of the Wumpus World:
Modeling is difficult with Propositional Logic

Schemas:

$(B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y}))$ Def. of breeze in pos [x,y]

$(S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}))$ Def. of stench in pos [x,y]

$(W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}))$ There is at least one wumpus!

..., etc. There is only one wumpus!

Logical Wumpus Agent

function HYBRID-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*,*breeze*,*glitter*,*bump*,*scream*]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
    t, a counter, initially 0, indicating time
    plan, an action sequence, initially empty

  TELL(KB, MAKE-PERCEPT-SENTENCE(*percept*, t))
  TELL the KB the temporal "physics" sentences for time t
  $safe \leftarrow \{[x, y] : \text{ASK}(KB, OK^t_{x,y}) = true\}$
  if ASK(KB, $Glitter^t$) $= true$ **then**
    $plan \leftarrow [Grab] + \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$
  if *plan* is empty **then**
    $unvisited \leftarrow \{[x, y] : \text{ASK}(KB, L^{t'}_{x,y}) = false \text{ for all } t' \leq t\}$
    $plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap safe, safe)$
  if *plan* is empty and ASK(KB, $HaveArrow^t$) $= true$ **then**
    $possible\_wumpus \leftarrow \{[x, y] : \text{ASK}(KB, \neg\, W_{x,y}) = false\}$
    $plan \leftarrow \text{PLAN-SHOT}(current, possible\_wumpus, safe)$
  if *plan* is empty **then** // no choice but to take a risk
    $not\_unsafe \leftarrow \{[x, y] : \text{ASK}(KB, \neg\, OK^t_{x,y}) = false\}$
    $plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap not\_unsafe, safe)$
  if *plan* is empty **then**
    $plan \leftarrow \text{PLAN-ROUTE}(current, \{[1, 1]\}, safe) + [Climb]$
  $action \leftarrow \text{POP}(plan)$
  TELL(KB, MAKE-ACTION-SENTENCE(*action*, t))
  $t \leftarrow t + 1$
  return *action*

Successor state axioms, etc

Try to construct plans based on goals with decreasing priority

function PLAN-ROUTE(*current*,*goals*,*allowed*) **returns** an action sequence
  inputs: *current*, the agent's current position
    goals, a set of squares; try to plan a route to one of them
    allowed, a set of squares that can form part of the route

  problem $\leftarrow$ ROUTE-PROBLEM(*current*, *goals*, *allowed*)
  return A*-GRAPH-SEARCH(*problem*)

$L^t_{x,y}$: Visited [x,y] at time t

ASK(KB, $\neg W_{x,y}$) $= false$

means that $KB \models \neg W_{x,y}$ is false

It does not mean that $KB \models W_{x,y}$ is true

ASK(KB, $\neg OK^t_{x,y}$) $= false$

means that $KB \models \neg OK^t_{x,y}$ is false

It does not mean that $KB \models OK^t_{x,y}$ is true

Local Search Algorithms for SAT

- Studying local search algorithms previously that combine both greediness and randomness
  - Hill climbing
  - Simulated Annealing
  - Stochastic Beam Search
- Local search can be applied directly to the SAT problem
  - Find an assignment that satisfies all clauses
  - Instances are full assignments
  - Children generated by flipping a variable's assignment
  - Evaluation function -
    - count the number of unsatisfied clauses
    - minimize
  - Can be many local minima
    - Use randomness to escape

WalkSAT

On each iteration: pick an unsatisfied clause and pick a symbol in it to flip

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

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

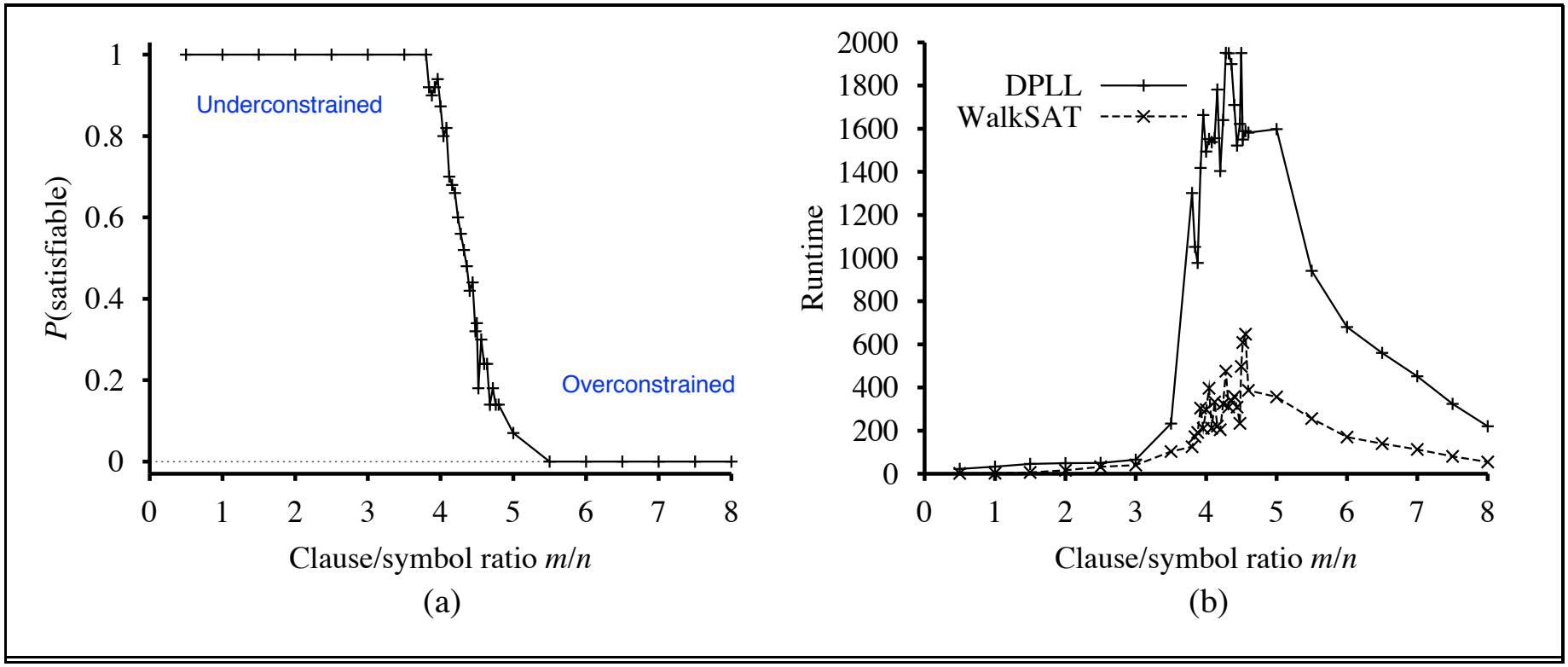
Two ways to choose the symbol to flip:

- *min-conflicts*: minimize the number of unsatisfied clauses
- *random walk*: pick the symbol randomly

WalkSAT: Completeness and Termination

- **WalkSAT** is sound
  - When the algorithm returns a model it does satisfy the input clauses.
- **WalkSAT** is not complete
  - When **WalkSAT** fails
    - either the sentence is unsatisfiable, or
    - the algorithm needs more time to find a solution
  - *max_flips* = infinity and p > 0
    - if a model exists, it will eventually find it (random walk)
    - if a model does not exist, the algorithm never terminates.
- SAT is **NP-Complete** so some problem instances will require exponential runtime.
  - Can we delineate the hard problem instances from the easy problem instances?

Landscape of Random SAT Problems



(a) Graph showing the probability that a random 3-CNF sentence with n = 50 symbols is satisfiable, as a function of the clause/symbol ratio m/n.

(b) Graph of the median run time (measured in number of recursive calls to DPLL, a good proxy) on random 3-CNF sentences.

The most difficult problems have a clause/symbol ratio of about 4.3.

$CNF_3(m,50)$

Sentences with 50 variables and 3 literals per clause

5 symbols/5clauses:

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

Resolution Theorem Proving

Resolution Theorem Proving

We considered the **unit clause heuristic [1960 Davis Putnam]** when studying **DPPL** and found that it is a satisfiability/truth preserving heuristic.

Robinson [1965], in a major breakthrough in automated theorem proving, based his technique on the **resolution inference rule** and also generalised it for the 1st-order case by introducing “on-demand” grounding using a **unification algorithm.**

Let’s begin with **Unit Resolution**

A **unit clause** is a disjunction consisting of a single literal

The **unit resolution rule** takes a **clause** and a **unit clause** and returns a new clause called the **resolvant**.

Resolution Rules

Unit Resolution Rule:

$$\frac{l_1 \vee \ldots \vee l_k, \quad m}{l_1 \vee \ldots \vee l_{i-1} \vee \ldots l_k}$$

The rule resolves on complementary literals: l_i, m

Example:

$$\frac{A \vee B \vee C \vee D \qquad \neg C}{A \vee B \vee D}$$

Example:

$$\frac{A \vee B \vee C \vee D \qquad \neg B}{A \vee C \vee D}$$

Resolution Rules

The unit resolution rule can be generalised:

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where l_i and m_j are complementary literals

Example:

$$\frac{A \vee B \vee C \vee D \qquad \neg E \vee \neg C \vee F}{A \vee B \vee D \vee \neg E \vee F}$$

Resolution

Note: The empty clause is **False**

$$\frac{P, \quad \neg P}{\bot}$$

Rule chaining is a special case of Resolution:

If $R \rightarrow P$ and $P \rightarrow Q$ then $R \rightarrow Q$

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

Soundness of Resolution

The Resolution Rule is Sound:

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

$$\{l_1 \vee \ldots \vee l_k, m_1 \vee \ldots \vee m_n\} \vdash_{\mathscr{R}}$$

$$\{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n\}$$

Preserves Truth and Satisfiability

Completeness of Resolution

Resolution as is, is not complete!

$$\frac{P \quad R}{P \vee R}$$?

$\{P, R\} \models P \vee R$ Is it is not the case that $\{P, R\} \vdash_{\mathscr{R}} P \vee R$

Resolution can not be used
directly to decide all logical entailments….

But…..

Resolution Refutation is Complete

Recall our meta-theorem:

Reductio ad absurdum

If the set Δ has a model but $\Delta \cup \{\neg\omega\}$ does not, then $\Delta \models \omega$

Proof by Refutation: To prove that $\Delta \models \omega$, show that $\Delta \cup \{\neg\omega\}$ has no model.

We can show that the negation of $P \vee R$ $\quad (\neg P \wedge \neg R)$
is inconsistent with $(P) \wedge (R)$

Resolve on P or R to generate a contradiction:

$$\frac{R, \quad \neg R}{\bot} \qquad \frac{P, \quad \neg P}{\bot}$$

Resolution Refutation Procedure

To prove an arbitrary wff, ω, from a set of wffs Δ , proceed as follows:

1. Convert the wffs in Δ to clause form -- a (conjunctive) set of clauses.
2. Convert the negation of the wff to be proved, ω , to clause form.
3. Combine the clauses from steps 1 and 2 into a single set, Γ.
4. Iteratively apply resolution to the clauses in Γ and add the results to Γ either until there are no more resolvents that can be added or until the empty clause is produced.

The empty clause will be produced by the refutation resolution procedure if $\Delta \models \omega$. We say that propositional resolution is **refutation complete**.

If Δ is a finite set of clauses and if $\Delta \not\models \omega$, then the resolution refutation procedure will terminate without producing the empty clause. We say that entailment is **decidable** for the propositional calculus by resolution refutation.

A Resolution Example

Suppose the agent is in [1,1] and there is no breeze. Show that there is no pit in [1,2]

$KB = \{(B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1}), \neg B_{1,1}\}$ Prove $\neg P_{1,2}$

Show that $KB \wedge \neg\neg P_{1,2}$ is inconsistent

$KB \wedge \neg\neg P_{1,2}$ in CNF form:

$(\neg P_{2,1} \vee B_{1,1})$
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$
$(\neg P_{1,2} \vee B_{1,1})$
$\neg B_{1,1}$
$P_{1,2}$

A Resolution Algorithm

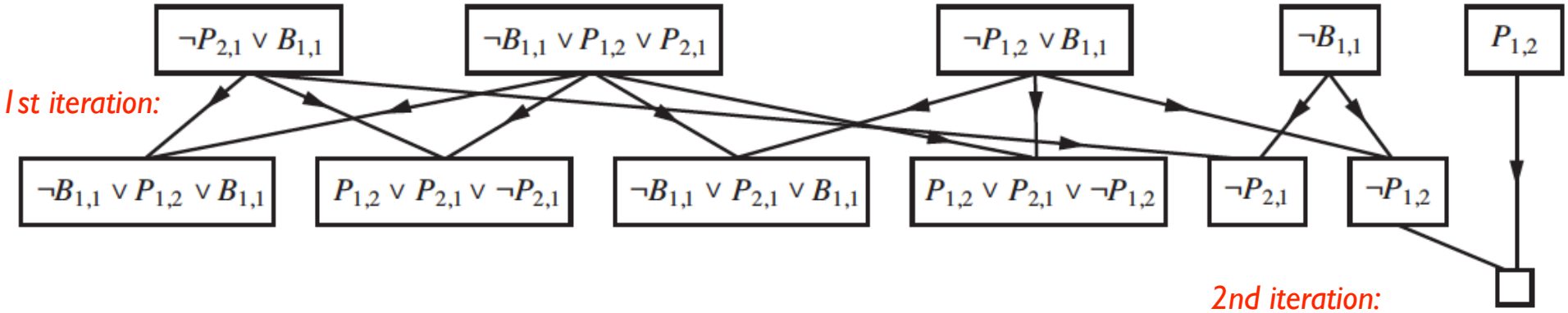
function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ 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*
    $clauses \leftarrow clauses \cup new$

No new clauses generated, no empty clause

Applying the Algorithm

Algorithm:



$$KB \wedge \neg \neg P_{1,2}$$

$(\neg P_{2,1} \vee B_{1,1})$
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$
$(\neg P_{1,2} \vee B_{1,1})$
$\neg B_{1,1}$
$P_{1,2}$

Refutation Tree: