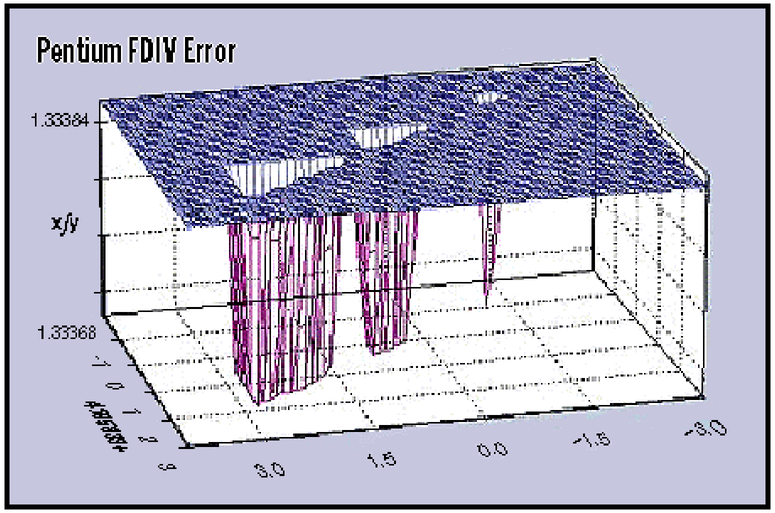
Intelligent Systems Programming

Lecture 4: Propositional Logic

IT University of Copenhagen

Verification and Expert Systems



Today's Program

- **Propositional Logic [12:00-12:50]**
  - Fundamental Concepts in Logic
  - Syntax and Semantics
  - Inference
    - Entailment
    - Logical equivalence
    - Inference rules
    - Formal proofs
- **Efficient Inference Algorithms [13:00-13:50]**
  - Resolution
  - Inference with Horn clauses
  - Efficient SAT checking
    - DPLL
    - WalkSat
  - Phase transition
- **Deloitte Hackathon [13:50-14:00]**
- **Exercises [14:00-16:00]**

Fundamental Concepts of Logic

The Purpose of Logics

- Logics are formal languages for representing information such that conclusions can be drawn
- Natural language is too complex and ambiguous

  "John saw the diamond through the window and stole it"

  Reading 1: John stole the diamond

  Reading 2: John stole the window

- Sentences in logics are assertions about a world that are either true or false

Logic: Syntax and Semantics

- Syntax defines the written form of legal sentences in the language
- Semantics define the truth-value of sentences in a world
- World is the setting or environment in which you derive the truth of sentences
- E.g., the language of inequalities
  - $x+2 \geq y$ is a sentence; $x2+y > \{\}$ is not a sentence
  - $x+2 \geq y$ is true in a world where $x = 7$, $y = 1$
  - $x+2 \geq y$ is false in a world where $x = 0$, $y = 6$

Entailment and Inference

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

$\models$ is a meta symbol, not a part of the syntax!

- **Knowledge base *KB* entails sentence α if and only if α is true in all worlds where *KB* is true**
  - E.g., *KB* containing *the-apple-is-red* and *the-apple-is-sweet* entails *the-apple-is-sweet*
  - E.g., *KB* containing "$y \geq 4$", "$y \leq 4$" entails $y = 4$

- Inference is to **decide** whether $KB \models \alpha$

Models

- A model is a formal description of a possible world used to decide truth-value of sentences

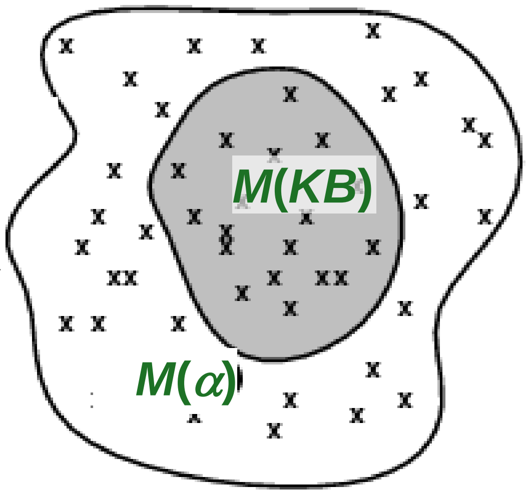
  Example:
  Possible world = state of nuclear power plant
  Model = state {*broken*, *hot*, *cold*} of pipe *A*, *B*, and *C*

- We say *m* is a model of a sentence *α* if *α* is true in *m*

- *M*(*α*) is the set of all models of *α*

Models

- $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$



Model Checking is to explicitly check this relationship

Inference Algorithms

- ***KB*** $\vdash_i$ ***α*** : sentence *α* can be derived from *KB* by procedure *i*
- Soundness: *i* is sound if *KB* $\vdash_i \alpha$ implies *KB* $\models \alpha$
  - Any sentence derived by *i* from *KB* is truth preserving
- Completeness: *i* is complete if *KB* $\models \alpha$ implies *KB* $\vdash_i \alpha$
  - All the sentences entailed by *KB* can be derived by procedure *i*
  - That is, the procedure will answer any question whose answer follows from what is known by the *KB*

Propositional Logic

Syntax

- Atomic sentences
  - Proposition symbols P, Q,… are sentences
  - The two constants *True* and *False* are sentences
- Complex sentences
  - If S_1 and S_2 are sentences then so are (in order of precedence)

• $\neg S_1$	*negation*	$\neg$ *not*	$\neg Q$, Q	***literals***
• $(S_1 \wedge S_2)$	*conjunction*	$\wedge$ *and*	S_1, S_2	*conjuncts*
• $(S_1 \vee S_2)$	*disjunction*	$\vee$ *or*	S_1, S_2	*disjuncts*
• $(S_1 \Rightarrow S_2)$	*implication*	$\Rightarrow$ *implies*	S_1	***premise***
			S_2	***conclusion***
• $(S_1 \Leftrightarrow S_2)$	*biimplication*	$\Leftrightarrow$ *if-and-only-if*		

Semantics (1)

- **Each model *m* assigns truth value *true* (1) or *false* (0) to each proposition symbol**

  E.g.

P	*Q*	*R*
false	*true*	*false*

- The constants *True* and *False* are always given the expected values

True	*False*
true	*false*

- This specifies the truth value of atomic sentences

Semantics (2)

- Rules for evaluating truth with respect to a model *m*:

$\neg S$	is *true*	iff	S is *false*		
$S_1 \wedge S_2$	is *true*	iff	S_1 is *true*	and	S_2 is *true*
$S_1 \vee S_2$	is *true*	iff	S_1 is *true*	or	S_2 is *true*
$S_1 \Rightarrow S_2$	is *true*	iff	S_1 is *false*	or	S_1, S_2 are *true*
$S_1 \Leftrightarrow S_2$	is *true*	iff	$S_1 \Rightarrow S_2$ is *true*	and	$S_2 \Rightarrow S_1$ is *true*

- Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P \Rightarrow (Q \wedge R) = \neg\, false \Rightarrow (true \wedge false) = true \Rightarrow false = false$$

Validity

A sentence is valid if it is true in all models

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, ...

Validity is connected to entailment via the

Deduction Theorem:

$KB \models \alpha$ if and only if ($KB \Rightarrow \alpha$) is valid

Satisfiability

A sentence is satisfiable if it is true in some model

e.g., $A \vee B$

A sentence is unsatisfiable if it is true in no models

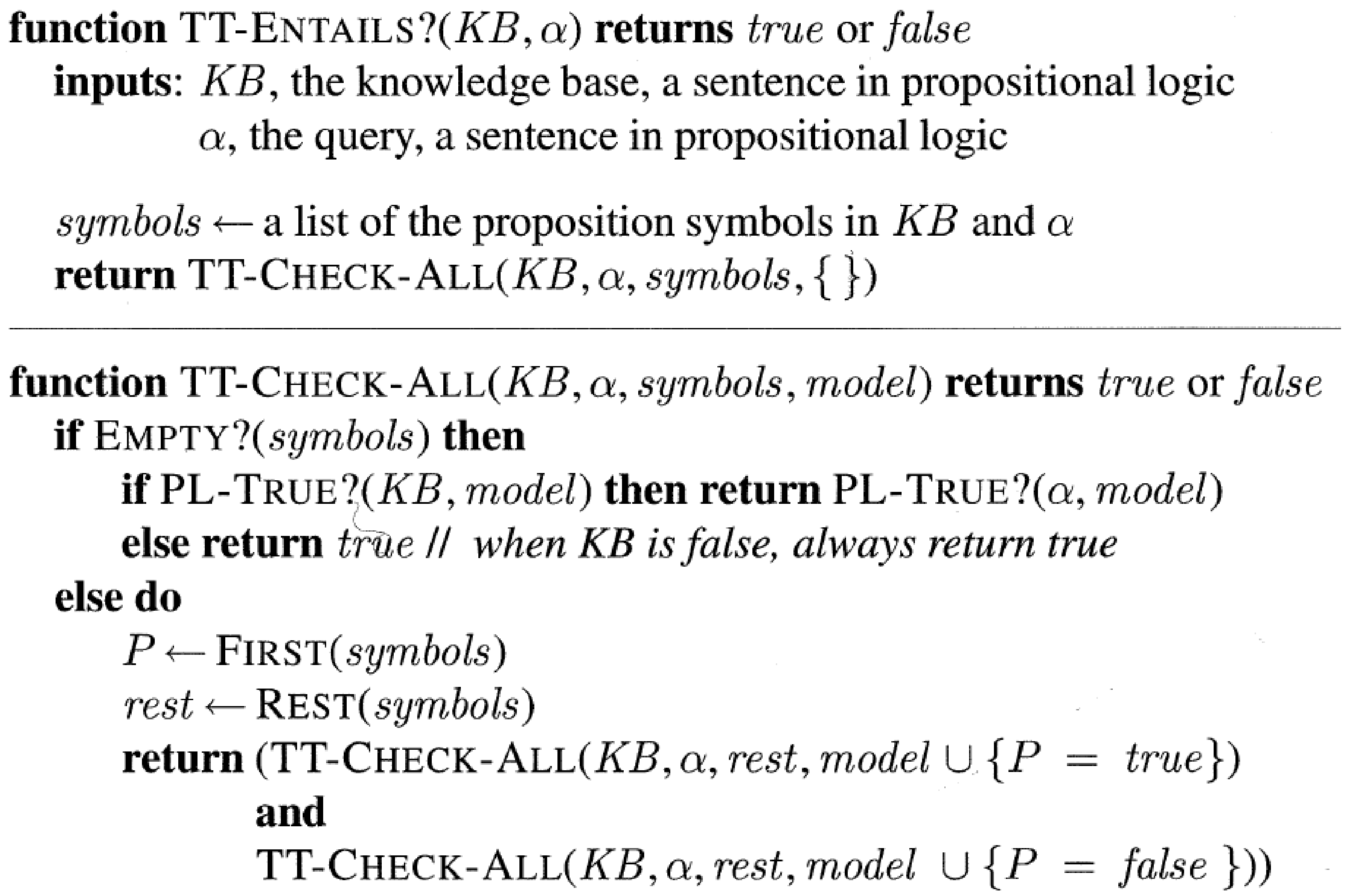
e.g., $A \wedge \neg A$

Satisfiability is connected to entailment via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

Inference by Enumeration

- Inference: decide whether $KB \models \alpha$



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

    $symbols \leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
    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($KB, \alpha, rest, model \cup \{P = false\}$))
```

Properties of TT-ENTAILS

- DFS enumeration and check of all models
- Sound? yes checks if α is true when *KB* is true
- Complete? yes checks all models
- For *n* symbols
  - time complexity is $O(2^n)$
  - space complexity is $O(n)$

Logical Equivalence

- Two sentences are logically equivalent iff they are true in the same models:

$$\alpha \equiv \beta \text{ iff } M(\alpha) = M(\beta) \text{ iff } \alpha \models \beta \text{ and } \beta \models \alpha$$

- Standard equivalences:
  - $\alpha \wedge \neg\alpha \equiv \mathit{False}$
  - $\alpha \vee \neg\alpha \equiv \mathit{True}$
  - $\alpha \wedge \mathit{True} \equiv \alpha$
  - $\alpha \vee \mathit{False} \equiv \alpha$
  - $\alpha \wedge \mathit{False} \equiv \mathit{False}$
  - $\alpha \vee \mathit{True} \equiv \mathit{True}$
  - $\alpha \wedge \alpha \equiv \alpha$
  - $\alpha \vee \alpha \equiv \alpha$

How would you prove these equivalences?

More Standard Equivalences

$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

Sentence Simplification

- **Determine logical equivalence between sentences using standard rules**

Example

$$
\begin{aligned}
& (a \vee (b \Rightarrow a)) & \\
\equiv \quad & (a \vee (\neg b \vee a)) & \text{impl. elim.} \\
\equiv \quad & \ldots & \\
\equiv \quad & b \Rightarrow a &
\end{aligned}
$$

- Formal proof!

Inference Rules

numerator (premises) $\models$ ***denominator (conclusion)***

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Modus Ponens

$$\frac{\alpha \wedge \beta}{\alpha}$$

And-Elimination

$$\frac{\alpha \Rightarrow \beta}{\neg\alpha \vee \beta} \qquad \frac{\neg\alpha \vee \beta}{\alpha \Rightarrow \beta}$$

All Equivalence rules

Inference Proof

- Search for inference rules to chain “goal” with axioms

$$\begin{tikzcd}
 & \text{Axioms (facts)} \arrow[rr, dash] \arrow[dl, dash] & & \dfrac{}{\alpha \wedge \gamma} \arrow[dl] \\
\dfrac{}{\alpha \Rightarrow \beta} \arrow[dr] & & \dfrac{\alpha \wedge \gamma}{\alpha} \arrow[dl] & \\
 & \dfrac{\alpha \Rightarrow \beta, \quad \alpha}{\beta} & & \\
\text{Goal} \arrow[ur, dash] & & &
\end{tikzcd}$$

Only complete approach if set of inference rules is complete!

Efficient Inference Algorithms

Conjunctive Normal Form (CNF)

Definition

- A literal is a symbol or a negated symbol
- A clause is a disjunction of literals
- CNF is a conjunction of clauses

E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

Translate Arbitrary Sentence to CNF

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$
3. Move $\neg$ inwards using de Morgan's rules and double-negation
4. Apply distribution law ($\land$ over $\lor$) and flatten
5. Eliminate symbol duplicates in clauses (factoring)

Resolution: A Complete Inference Rule

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

where ℓ_i and m_j are complementary literals ($\ell_i \equiv \neg m_j$).

E.g., $$\frac{P \vee Q, \qquad \neg Q \vee R}{P \vee R}$$

Resolution Algorithm

- Remember: $KB \models \alpha$ iff $KB \wedge \neg\alpha$ is unsatisfiable
- Keep doing resolution on clauses until
  - fixpoint reached with no empty clause ()
    return *false*
  - () (= False) derived
    return *true*

$$\frac{Q, \quad \neg Q}{()}$$

Resolution Algorithm

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \land \neg\alpha$
  $new \leftarrow \{\}$
  **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*
     $clauses \leftarrow clauses \cup new$
```

Properties of the Resolution Algorithm

- **Sound**, yes
- **Complexity**
  - Time, Space: $O(2^n)$, due to blow-up of number of clauses
- **Complete,** yes
  Ground resolution theorem
  if a CNF with set of clauses S is unsatisfiable, then the resolution closure of those clauses, $RC(S)$, contains the empty clause $()$.

Proof of ground resolution theorem

We must show: **if *S* is unsatisfiable then () in *RC*(*S*)**

Prove log. equiv. sentence: **if () not in *RC*(*S*) then *S* satisfiable**

Satisfying assignment to propositions P_1 to P_k

For $i = 1$ to k

- $P_i = \mathit{False}$ if there is a clause $(\mathit{False} \vee \ldots \vee \mathit{False} \vee \neg P_i)$
- $P_i = \mathit{True}$ otherwise

Why is this assignment satisfying?

Horn Form Clauses

- Horn clause: clause with *at most* one positive literal
- Definit clause: *exactly* one positive literal

$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow C$

$\Rightarrow F$

head

body

fact

- Modus Ponens (for Definit clause)

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- **Inference possible in linear time!**

Forward chaining inference

KB:

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

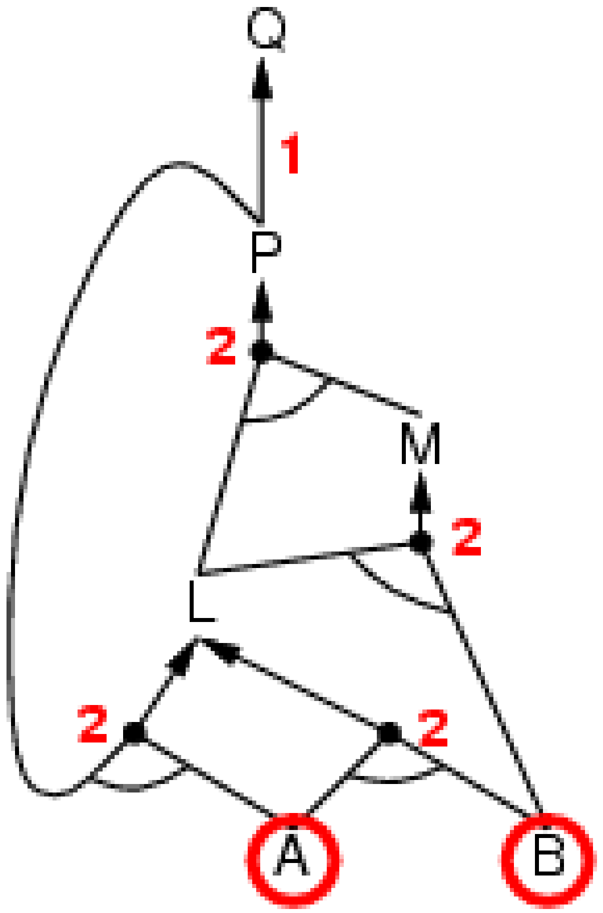
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

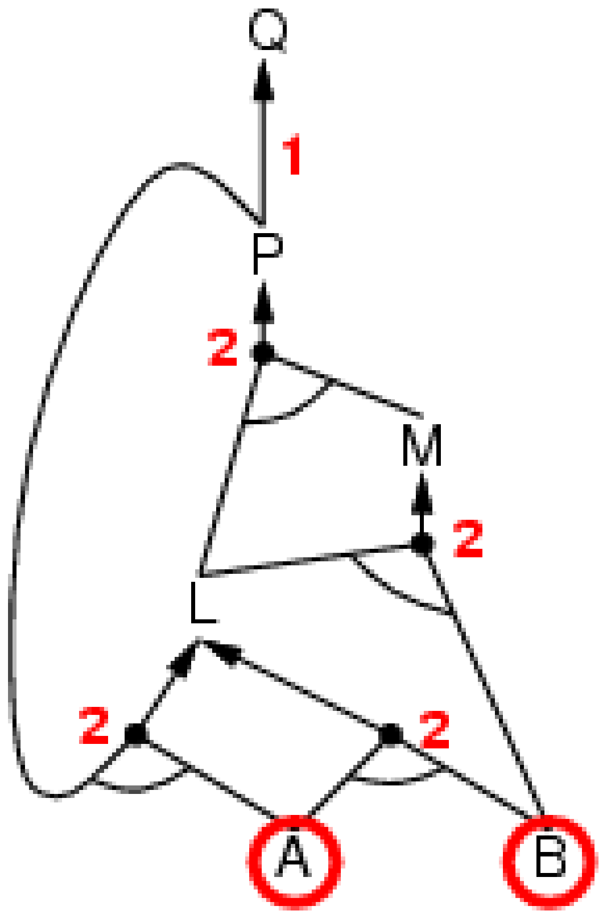
A

B

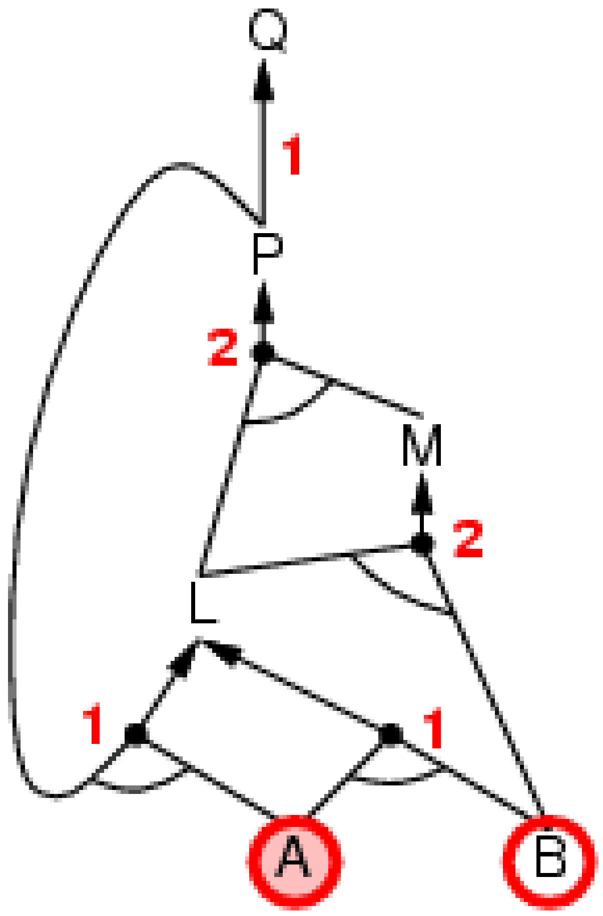
$KB \vDash Q$?



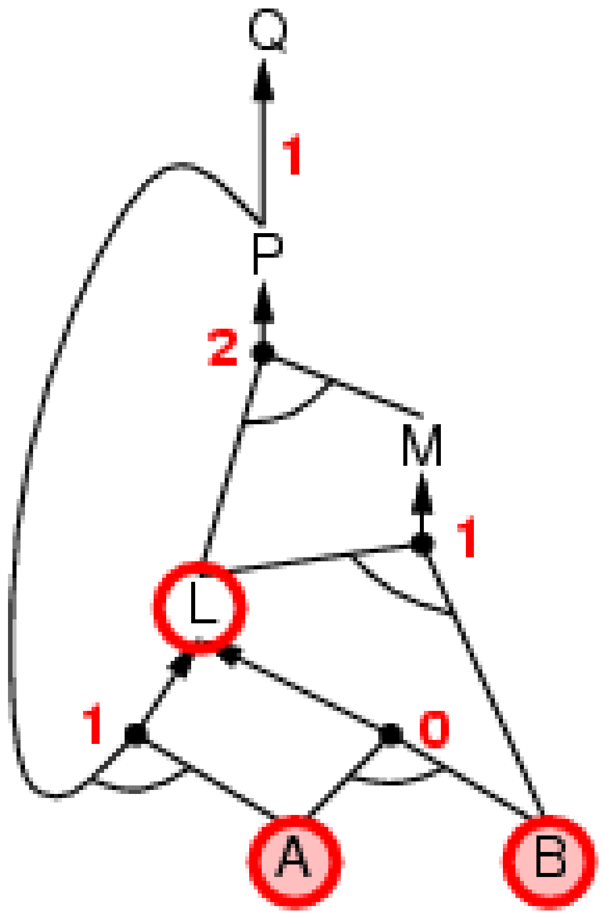
Forward chaining example



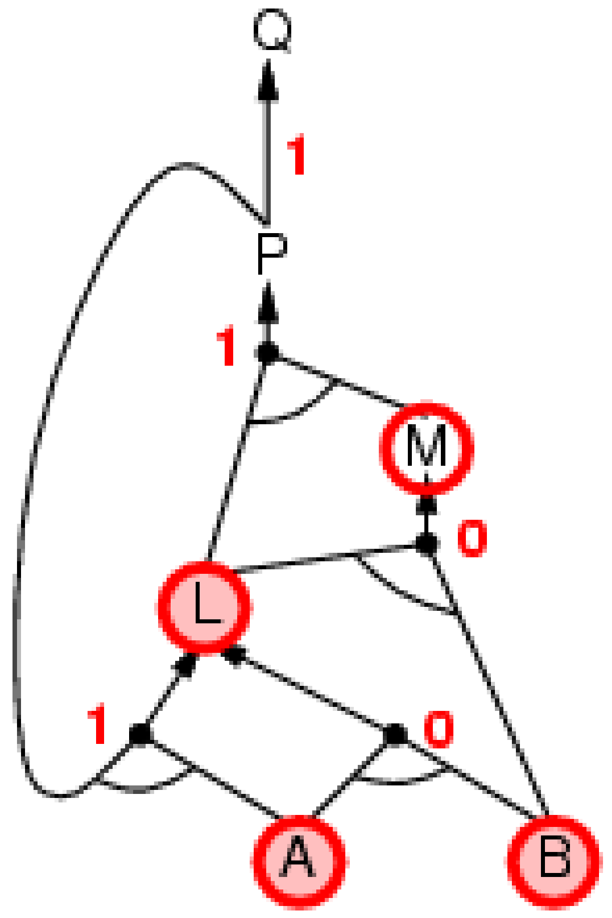
Forward chaining example



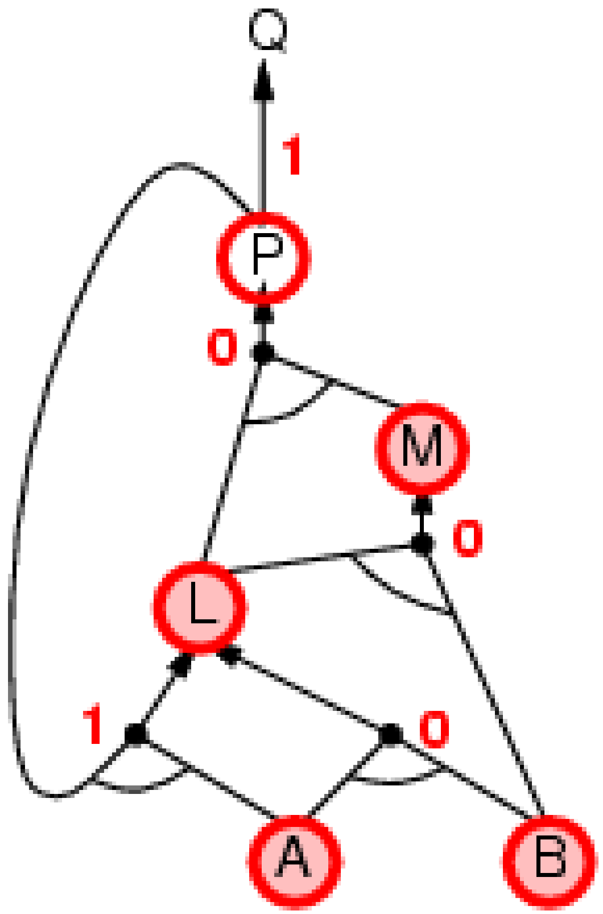
Forward chaining example



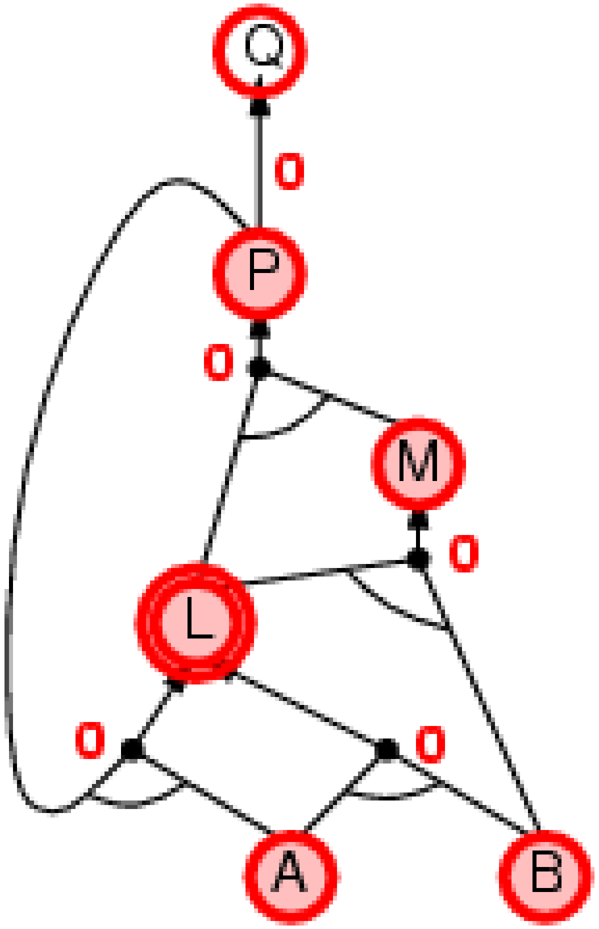
Forward chaining example



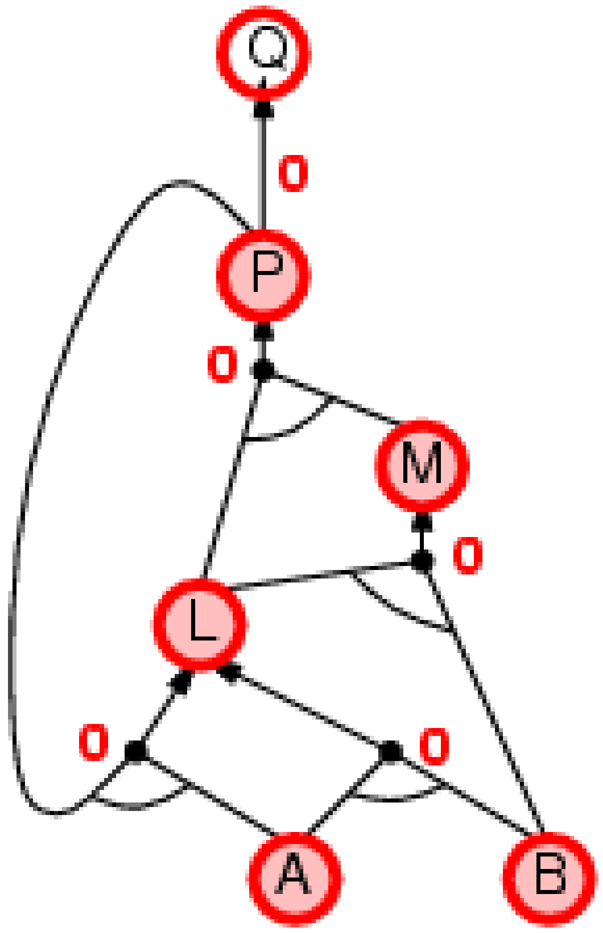
Forward chaining example



Forward chaining example



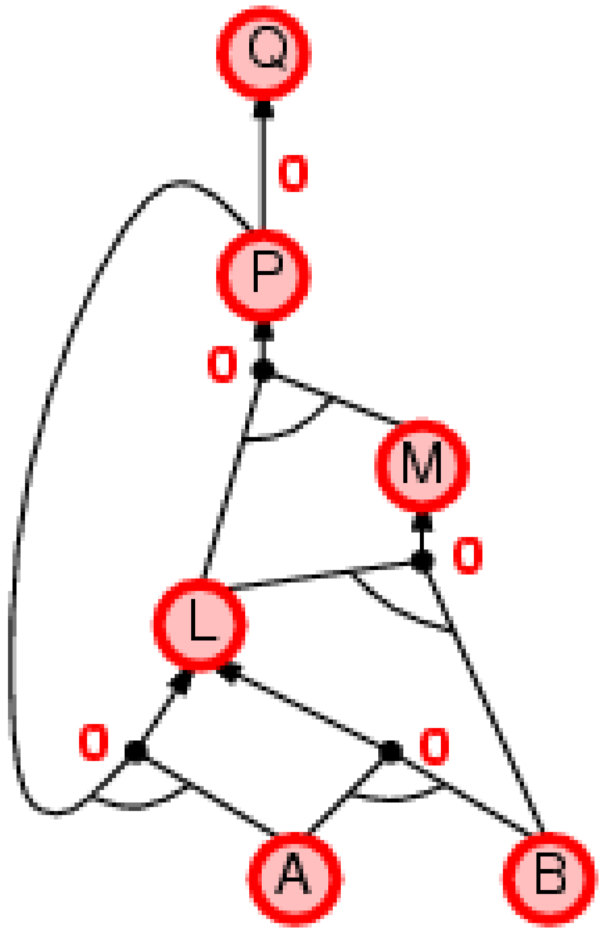
Forward chaining example



Forward chaining example

KB ⊨ Q

Yes!



Properties of Forward Chaining

- **Sound**, yes since Modus Ponens is sound
- **Complete**, yes
- **Space and time**: $O(n)$, where n is the total number of clause literals

Efficient SAT-Checking

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)

Incomplete local search algorithms

- `WalkSAT` algorithm

The DPLL Algorithm

Determine if a CNF sentence is satisfiable

Improvements over **truth table enumeration**:

1. **Early termination**

   A clause is true if any literal is true

   A sentence is false if any clause is false

2. **Pure symbol heuristic**

   Pure symbol: always appears with the same "sign" in all clauses

   e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure

   Make a pure symbol literal true

3. **Unit propagation**

   Unit clause: only one literal in the clause

   The only literal in a unit clause must be true

   e.g., $(False \vee \neg B)$: B must be false

The DPLL Algorithm

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

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

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

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

The `WalkSAT` algorithm

Determine if a CNF sentence is satisfiable

Algorithm

- Start with a random complete assignment
- In each iteration:
  - Pick random false clause
  - With probability *p*
    - flip random literal in clause
  - Else
    - flip literal that makes most clauses true
- Stop when reaching given max number of iterations

Phase Transition

- Consider random 3-CNF sentences. e.g.,

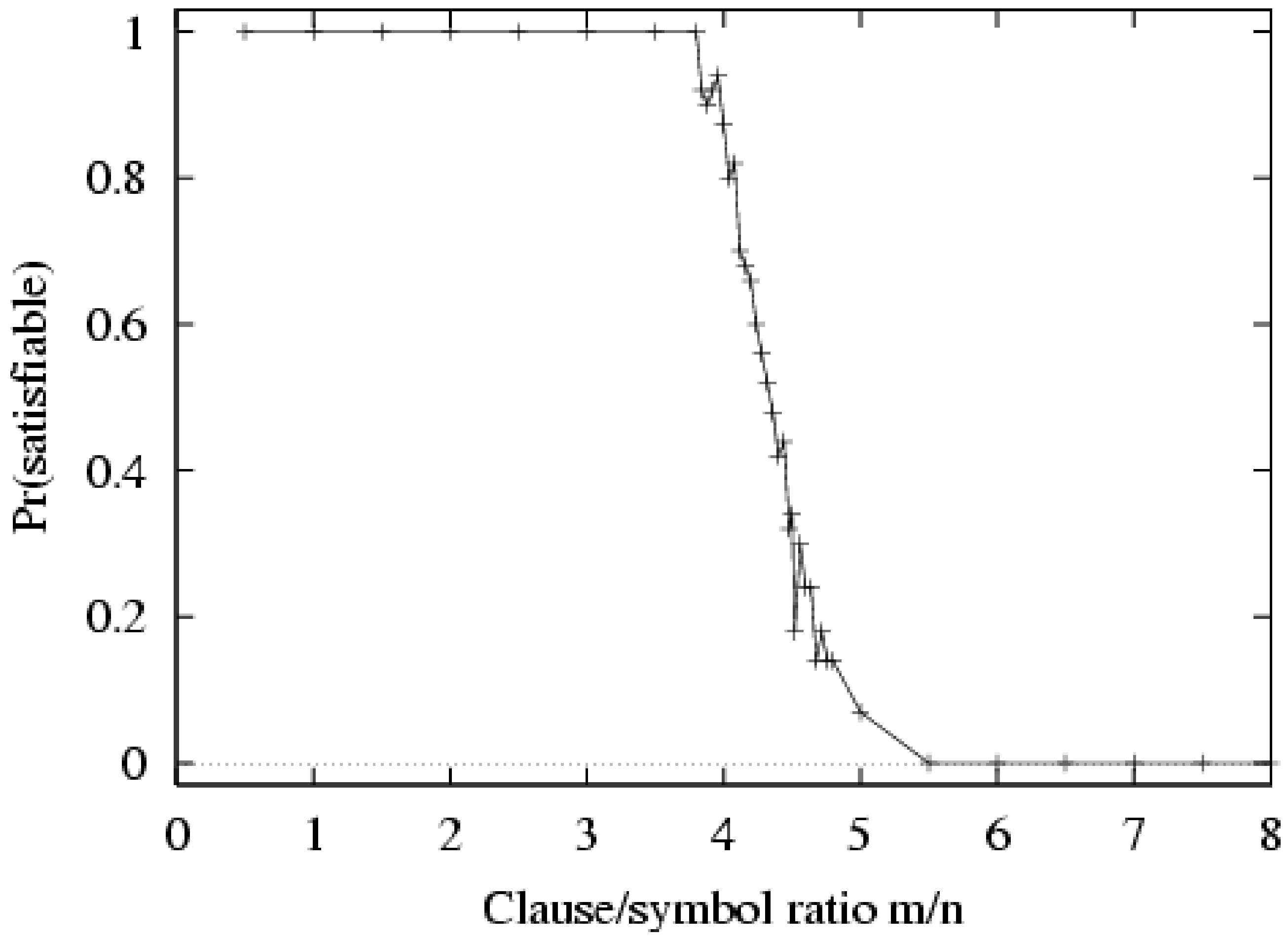
  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

  m = number of clauses

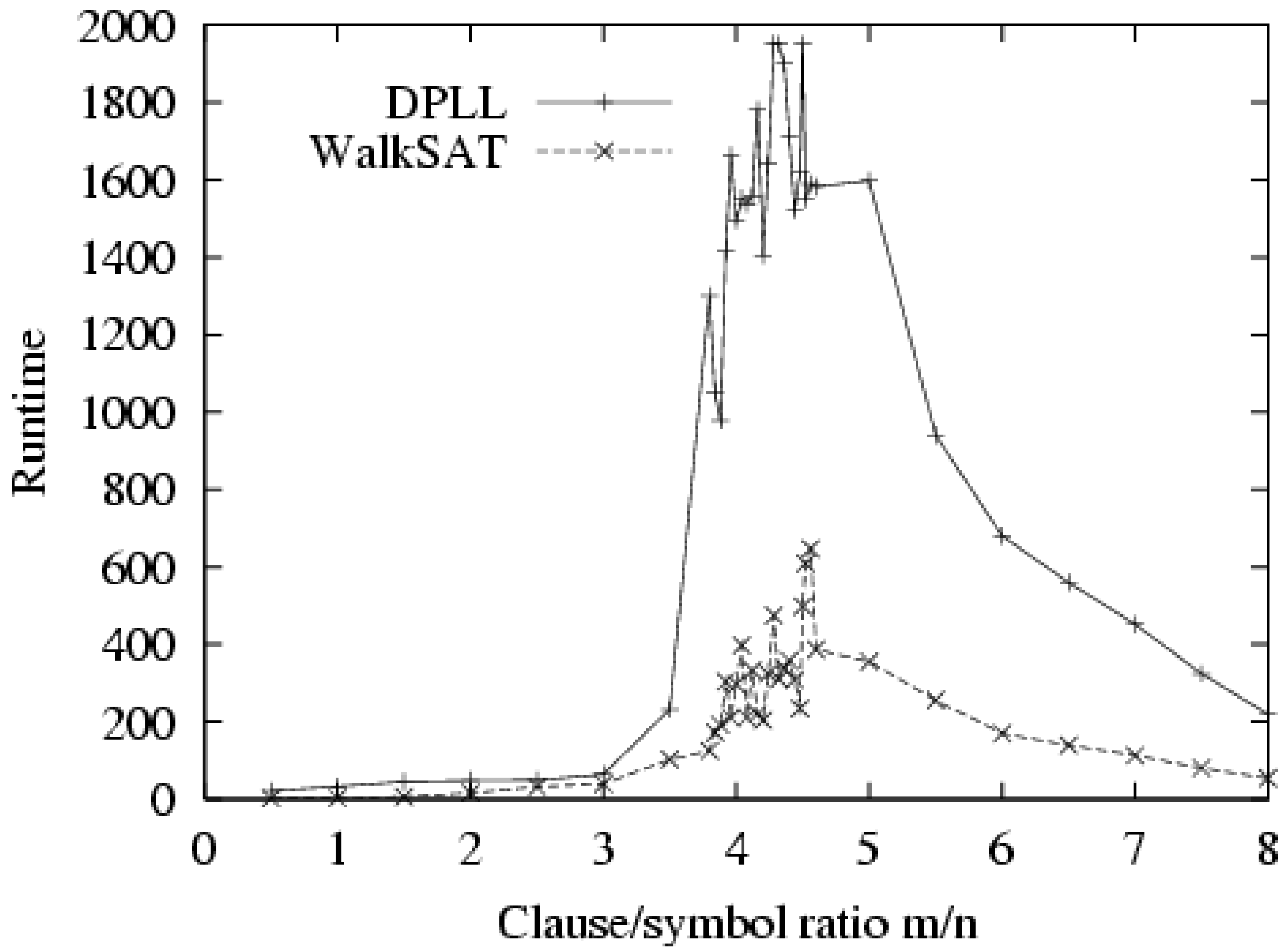
  n = number of symbols

  – Hard problems seem to cluster near *m*/*n* = 4.3 (phase transition)

Phase Transition



Phase Transition


Runtime
DPLL
WalkSAT
Clause/symbol ratio m/n