Propositional Logic Programming and

Knowledge Representation with Horn Clauses

Logic Programming

- Important Formalism for Knowledge Representation and Reasoning (KRR)
- Has its roots in Theorem Proving, based on the Resolution Calculus (Robinson)
- Express knowledge in terms of facts and rules
- **Algorithm = Logic + Control** (Kowalski)
- PROLOG emerged a general purpose programming language
- Modern languages: enhanced with features such as constraint solving, non-monotonic negation for KRR
- Here: Consider core language, plus some extensions

Positive Propositional Logic Programs

$$
\begin{aligned}
shut\_down &:- overheat \\
shut\_down &:- leak \\
leak &:- valve\_closed, pressure\_loss \\
valve\_closed &:- signal\_1 \\
pressure\_loss &:- signal\_2 \\
overheat &:- signal\_3 \\
signal\_1 &:- \\
signal\_2 &:-
\end{aligned}
$$

- This program captures (simplified) knowledge about a steam engine equipped with three signal gauges.
- Informally, the rules tell that the system has to be shut down if it is in a dangerous state.
- Such states are connected to causes and signals by respective rules.

Logic Program Syntax

- A *Horn clause* is a rule of the form

$$A_0 \leftarrow A_1, \ldots, A_m \qquad (m \geq 0),$$

where each A_i is a propositional atom.

- The parts on the left and on the right of "$\leftarrow$" are called the *head* and the *body* of the rule, respectively.

- A rule r of the form $A_0 \leftarrow$, i.e., whose body is empty, is called a *fact*.

- A *logic program* is a finite set of Horn clauses.

Logic Program Semantics

- An atom A *is true* w.r.t. program P (denoted $P \models A$), if A is a classical consequence of P.

$$
\begin{aligned}
\mathit{shut\_down} &:- \mathit{overheat} \\
\mathit{shut\_down} &:- \mathit{leak} \\
\mathit{leak} &:- \mathit{valve\_closed}, \mathit{pressure\_loss} \\
\mathit{valve\_closed} &:- \mathit{signal\_1} \\
\mathit{pressure\_loss} &:- \mathit{signal\_2} \\
\mathit{overheat} &:- \mathit{signal\_3} \\
\mathit{signal\_1} &:- \\
\mathit{signal\_2} &:-
\end{aligned}
$$

- $P \models \mathit{signal\_1}$, $P \models \mathit{signal\_2}$, $P \models \mathit{valve\_closed}$, $\ldots$ $P \models \mathit{leak}$, $\ldots$

Relationship to the SAT Problem

- Each program P can viewed as a classical CNF $\phi(P)$
- Each rule r corresponds to a clause $\phi(r)$:

$$A_0 \leftarrow A_1, \ldots, A_m \qquad \rightleftharpoons \qquad A_0 \vee \neg A_1 \cdots \neg A_m$$

- $\phi(P) = \bigwedge_{r \in P} \phi(r)$

Theorem. $P \models A$ holds if and only if $\phi(P) \wedge \neg A$ is unsatisfiable

Remark: in Logic Programming, a "query" A is often written as $\leftarrow A$.

Logic Programs: Semantics

- The *Herbrand Base B_P* of program P is the set of all atoms occurring in P
- A *Herbrand interpretation* of P is any subset $I \subseteq B_P$

  Intuitively, the atoms in I are true and all others are false.
- A *Herbrand model* of P is any Herbrand interpretation I which satisfies every rule $A_0 \leftarrow A_1, \ldots, A_m$ in P, i.e., $A_0 \in I$ whenever $\{A_1, \ldots, A_m\} \subseteq I$
- The semantics of P is given by the *least Herbrand model* of P, denoted $LM(P)$, i.e., the unique Herbrand model M of P such that each different Herbrand model I of P satisfies $I \not\subseteq M$.

Example /2

shut_down :– *overheat*

shut_down :– *leak*

leak :– *valve_closed*, *pressure_loss*

valve_closed :– *signal_1*

pressure_loss :– *signal_2*

overheat :– *signal_3*

signal_1 :–

signal_2 :–

- M_1 = { <all atoms> } = B_P
- M_2 = { *signal_1*, *signal_2*, *valve_closed*, *pressure_loss*, *leak shut_down*, *leak*, *overheat* }
- M_3 = { *signal_1*, *signal_2*, *valve_closed*, *pressure_loss*, *shut_down*, *leak* }

Operational Characterization

- We can compute $lm(P)$ by fixpoint iteration of the *immediate consequence* operator

$$T_P : 2^{B_P} \longrightarrow 2^{B_P}$$

defined by

$$T_P(I) = \{A_0 \in B_P \mid P \text{ contains a rule } A_0 \text{ :- } A_1, \ldots, A_m \text{ such that } \{A_1, \ldots, A_m\} \subseteq I \text{ } \textit{holds} \}.$$

- Intuition: all facts provable by rules in P from I in one step.
- Notice: The operator T_P is *monotone*, i.e., $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$

Fixpoint Results

Well-known results in Logic Programming:

- **Theorem.** T_P has a *least fixpoint* T_P^∞, which is the limit of the sequence $\langle T_P^i \rangle_{i \geq 0}$ defined by

$$\begin{aligned} T_P^0 &= \emptyset, \\ T_P^{i+1} &= T_P(T_P^i), i \geq 0. \end{aligned}$$

- **Theorem.** $T_P^\infty = \{A \in B_P \mid P \models A\}$

- **Theorem.** $T_P^\infty = LM(P)$

Example (continued)

$$
\begin{aligned}
\textit{shut\_down} &:- \textit{overheat}\\
\textit{shut\_down} &:- \textit{leak}\\
\textit{leak} &:- \textit{valve\_closed}, \textit{pressure\_loss}\\
\textit{valve\_closed} &:- \textit{signal\_1}\\
\textit{pressure\_loss} &:- \textit{signal\_2}\\
\textit{overheat} &:- \textit{signal\_3}\\
\textit{signal\_1} &:- \quad \textit{signal\_2} :- .
\end{aligned}
$$

$$
\begin{aligned}
T_P^0 &= \emptyset,\\
T_P^1 &= \{\textit{signal\_1}, \textit{signal\_2}\},\\
T_P^2 &= T_P^1 \cup \{\textit{valve\_closed}, \textit{pressure\_loss}\},\\
T_P^3 &= T_P^2 \cup \{\textit{leak}\},\\
T_P^4 &= T_P^\infty = T_P^3 \cup \{\textit{shutdown}\}.
\end{aligned}
$$

Thus, the least fixpoint is reached in four steps

Logic Programs: Inference if negative Information

- Conclude negative information under *Negation as Failure*:

- **Definition.** $T \models \neg A$ if $T \not\models A$

Example: $P \models \neg$*overheat*, because $P \not\models$ *overheat*

- Evaluation inference: For each atom A,

  - $P \models A \Leftrightarrow A \in LM(P)$
  - $P \models \neg A \Leftrightarrow A \notin LM(P)$

- This constructively implements the *Closed World Assumption*

Complexity of Propositional Logic Programs

- Existence of $lm(P)$ is trivial (it always exists)
- Reasoning: Given a program P and an atom A, decide whether $A \in lm(P)$

Theorem. Deciding whether $A \in lm(P)$ is **P**-complete.

Proof.

- **Membership:** Computing T_P^∞ is feasible in polynomial time, and then we only need to check whether $A \in T_P^\infty$.
- **Hardness:** Encoding of a deterministic Turing Machine (DTM).

  Given a DTM T, an input string I and a number of steps N (where N is a polynomial in $|I|$), construct in logspace a program $P = P(T, I, N)$ and an atom A such that $P \models A$ iff T accepts input I within N steps

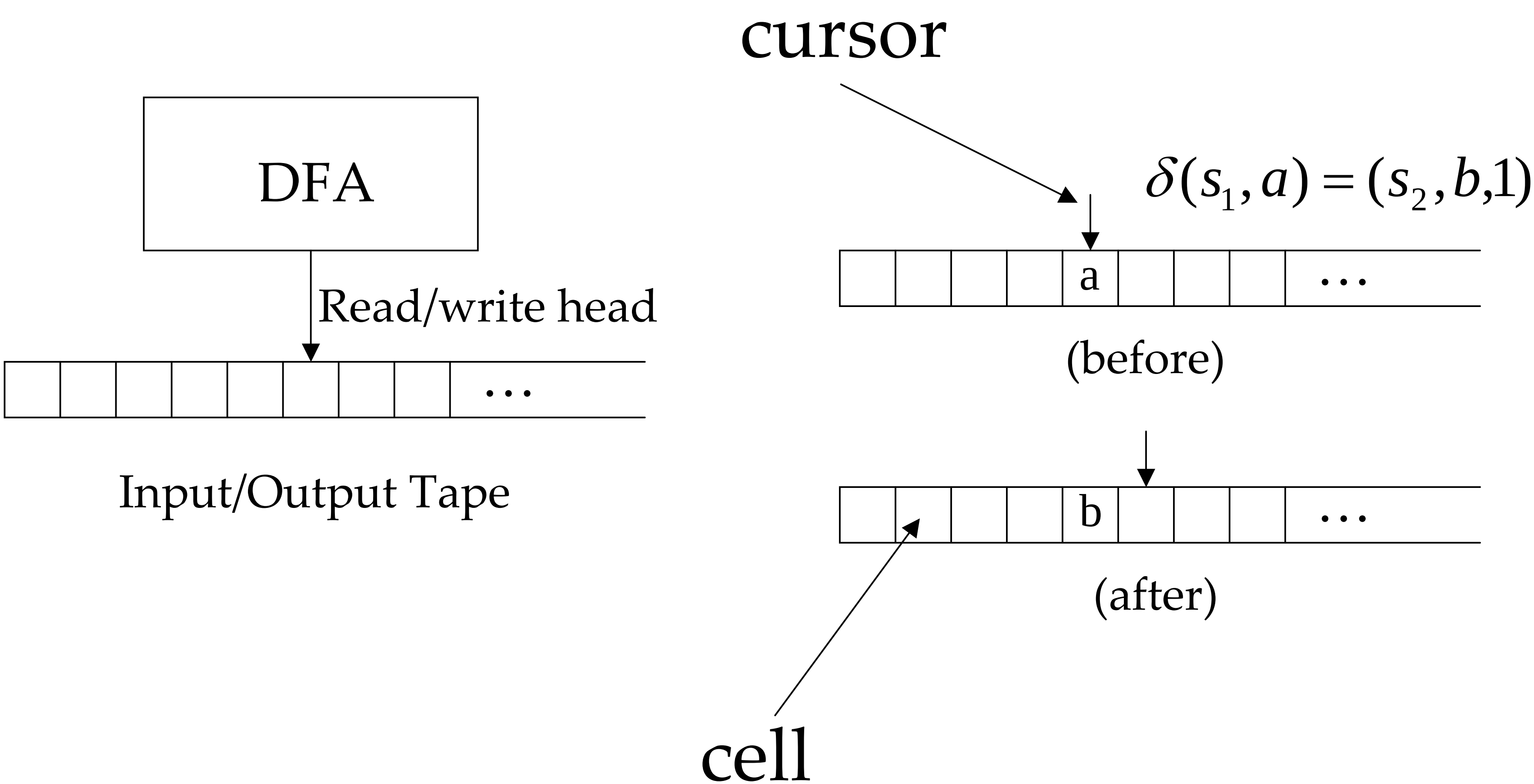
P-hardness:

Deterministic Turing Machine (DTM) $T = (S, \Sigma, \delta, s_0)$

where: $\sqcup \in \Sigma$, s_0, accept $\in S$, $\delta: S \times \Sigma \to (S \times \Sigma \times \{-1,0,1\})$

- T divided into *cells*, *cursor* move along the tape
- An input string I is written on the tape: the first $|I|$ cells $c_0,\ldots,c_{|I|-1}$ of the tape, all other cells contain $\sqcup$
- T takes successive *steps* of computation according to δ.

$$\delta(s,\sigma) = (s', \sigma', d) \ (d = -1 \text{ or } 0 \text{ or } 1)$$



DFA
Read/write head
…
Input/Output Tape
cursor
$\delta(s_1, a) = (s_2, b, 1)$
a
…
(before)
b
…
(after)
cell

T: DTM

Transition function $t=\langle s,\sigma, s',\sigma',d\rangle$ expresses the following if-then-rule:

If at some time instant τ the DTM is in state s, the cursor points to cell number π, and this cell contains symbol σ

Then at instant $\tau +1$ the DTM is in state s', cell number π contains symbol σ', and the cursor points to cell number $\pi+d$

- Possible to describe the computation of a DTM ***T*** on input string ***I*** from its initial configuration at time instance 0 to the configuration at instant ***N*** by a (horn) propositional logic program ***L(T,I,N)***

- The goal: encode the PTIME Turing computation of ***T*** on input ***I*** with a horn logic program ***L*** and an atom ***G***, s.t. $L \models G$ iff ***T*** accepts ***I*** in at most ***N*** steps

Propositional atoms

(there are many, but only polynomially many…)

- ***symbol***$_\alpha[\tau,\pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\alpha \in \Sigma$. Intuitive meaning: at instant τ of the computation, cell number π contains symbol α

- ***cursor*** $[\tau,\pi]$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$. Intuitive meaning: at instant τ of the cursor points to cell number π

- ***state***$_s[\tau]$ for $0 \leq \tau \leq N$ and $s \in S$. Intuitive meaning: at instant τ the DTM T is in state s

- ***accept*** Intuitive meaning: T has accepted.

- Initialization facts:

$symbol_{\sigma}[0,\pi] \leftarrow$ for $0 \leq \pi \leq |I|$, where $I_{\pi} = \sigma$

$symbol_{\sqcup}[0,\pi] \leftarrow$ for $|I| \leq \pi \leq N$

$cursor[0,0] \leftarrow$

$state_{s_0}[0] \leftarrow$

- Transition rules: $t=\langle s,\sigma, s',\sigma',d\rangle$ $0 \le \tau,\pi \le N$

$symbol_{\sigma'}[\tau+1,\pi] \leftarrow state_s[\tau], symbol_{\sigma}[\tau,\pi], cursor[\tau,\pi]$

$cursor\,[\tau+1,\pi+d] \leftarrow state_s[\tau], symbol_{\sigma}[\tau,\pi], cursor[\tau,\pi]$

$state_{s'}[0] \leftarrow state_s[\tau], symbol_{\sigma}[\tau,\pi], cursor[\tau,\pi]$

- Inertia rules: $0 \le \pi \neq \pi' \le N$

$symbol_{\sigma}[\tau+1,\pi] \leftarrow symbol_{\sigma}[\tau,\pi], cursor[\tau,\pi']$

- Accept rules: $0 \le \tau \le N$

$accept \leftarrow state_{\mathrm{accept}}[\tau]$

Our encoding precisely simulates the behaviour machine T on input I up to N steps.

(This can be formally shown by induction on the time steps.)

Therefore:

L(T,I,N) $\models$ *accept* if and only if the DTM T accepts the input string I within N steps.

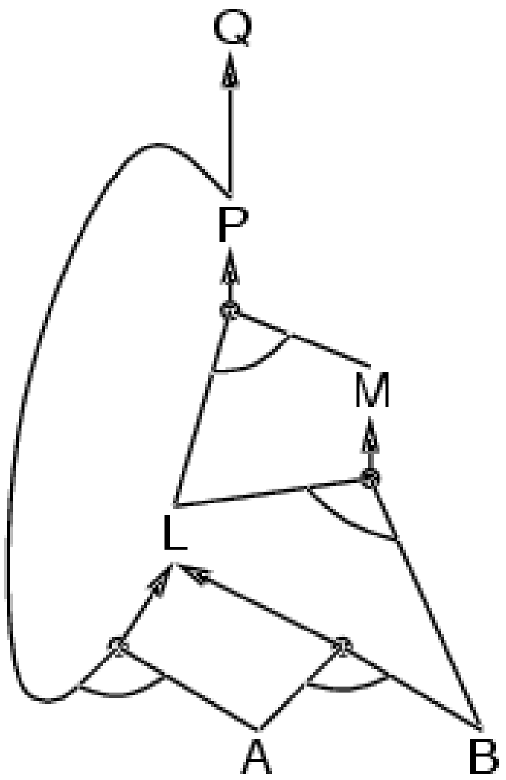
The construction is feasible in Logspace.

➔ Horn clause inference is P-complete

Forward chaining

- Idea: fire any rule whose premises are satisfied in the *KB*
  - add its conclusion to the *KB*, until query is found

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

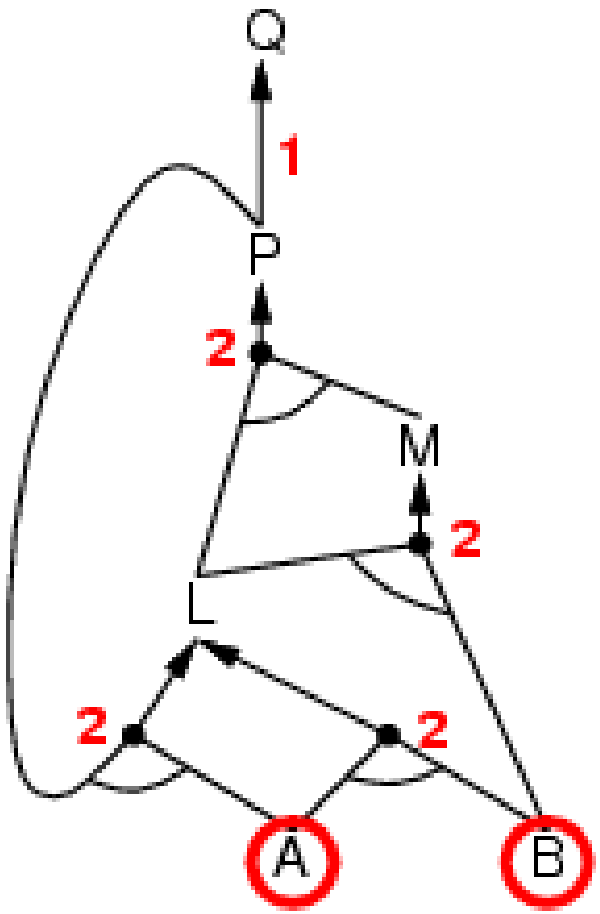


Forward chaining algorithm (Minoux)

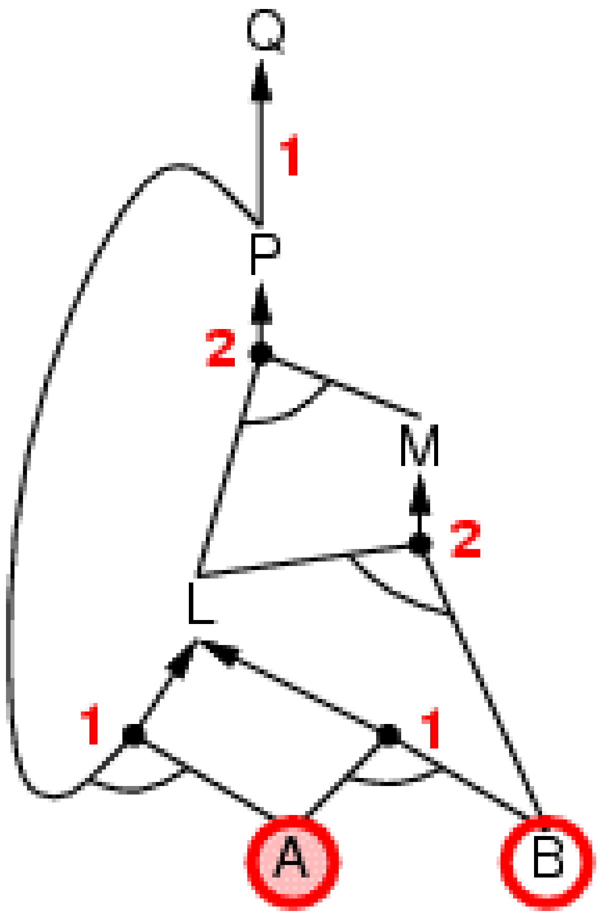
```
function PL-FC-ENTAILS?(KB, q) returns true or false
   local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table, indexed by symbol, each entry initially false
                    agenda, a list of symbols, initially the symbols known to be true

   while agenda is not empty do
       p ← POP(agenda)
       unless inferred[p] do
           inferred[p] ← true
           for each Horn clause c in whose premise p appears do
               decrement count[c]
               if count[c] = 0 then do
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

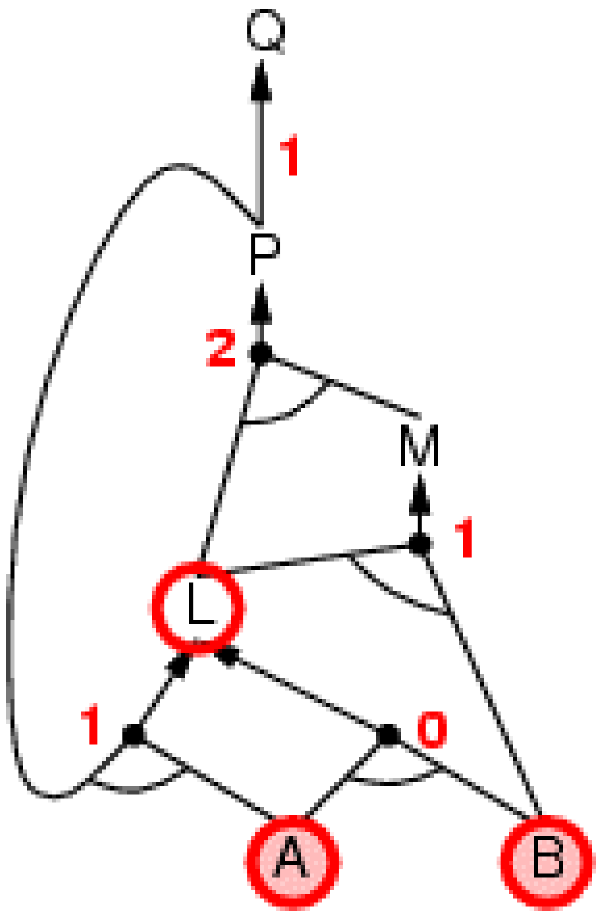
Forward chaining example



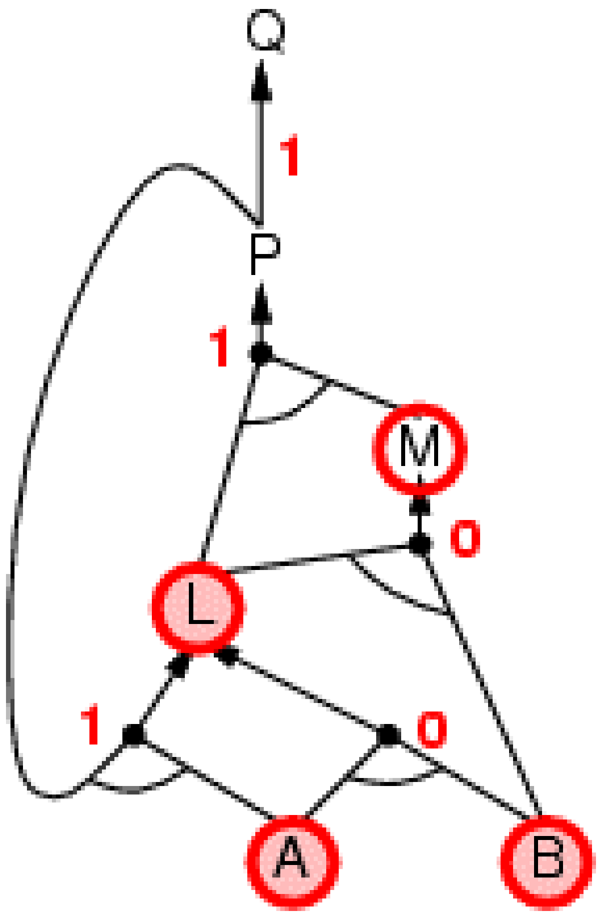
Forward chaining example



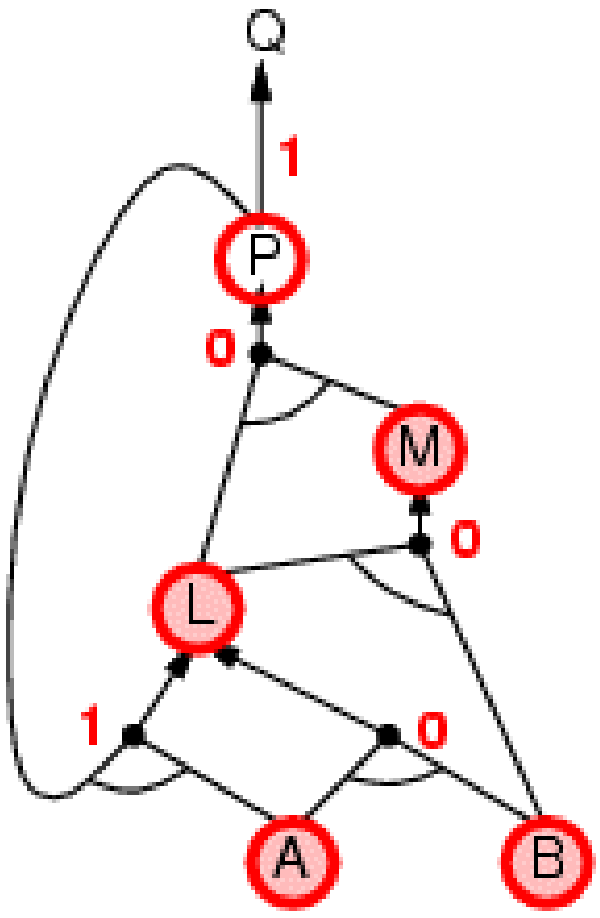
Forward chaining example



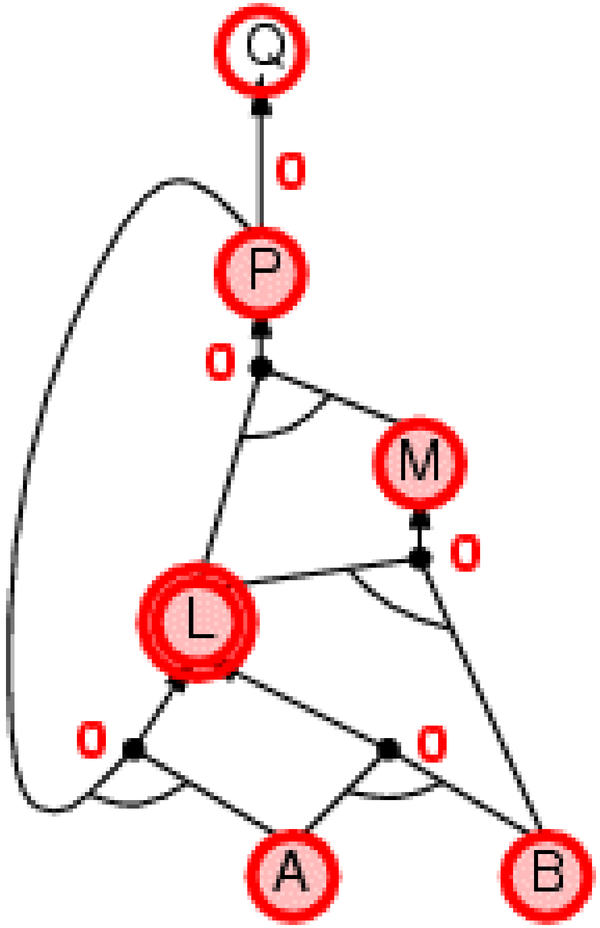
Forward chaining example



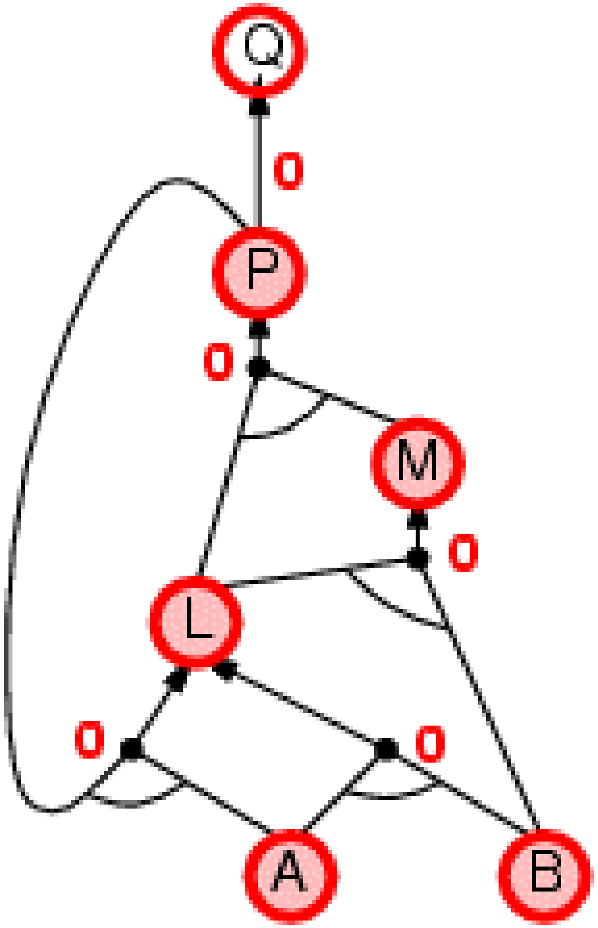
Forward chaining example



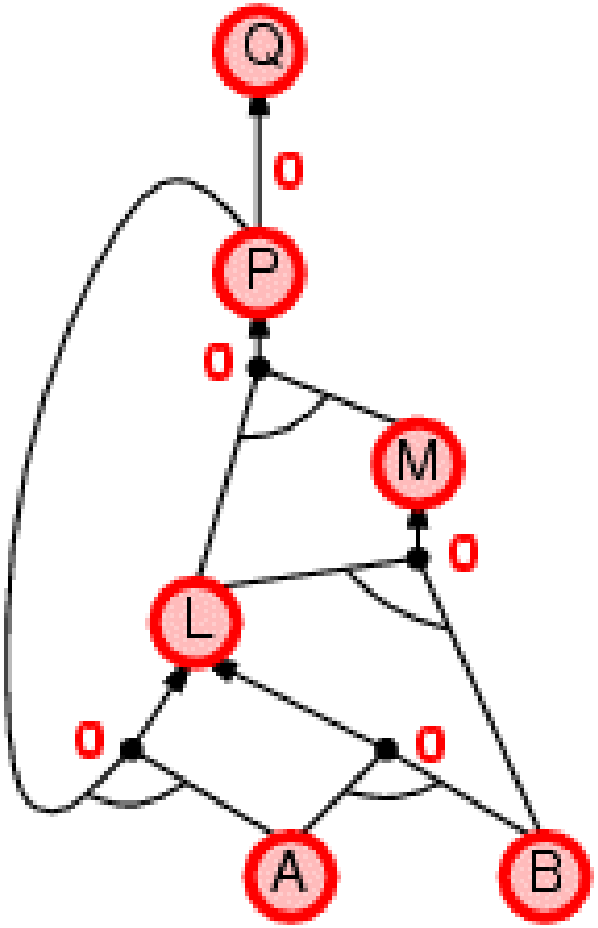
Forward chaining example



Forward chaining example



Forward chaining example



This algorithm can be implemented to run in Linear time on a Random Access Machine.

It suffices to use appropriate data structures (arrays)

Read the Minoux Paper

➔ Propositional Horn inference is feasible in Linear Time