PROPOSITIONAL INference, PROPOSITIONAL AGENTS

Chapter 7.5–7.7
Outline

◇ Inference rules and theorem proving
  – forward chaining
  – backward chaining
  – resolution
◇ Efficient model checking algorithms
◇ Boolean circuit agents
Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules
- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as “actions” in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking
  truth table enumeration (always exponential in $n$)
  improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
  heuristic search in model space (sound but incomplete)
    e.g., min-conflicts-like hill-climbing algorithms
Forward and backward chaining

Horn Form (restricted)

KB = conjunction of Horn clauses

Horn clause =

◊ proposition symbol; or
◊ (conjunction of symbols) \( \Rightarrow \) symbol

E.g., \( C \land (B \Rightarrow A) \land (C \land D \Rightarrow B) \)

Modus Ponens (for Horn Form): complete for Horn KBs

\[
\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}{\beta}
\]

Can be used with forward chaining or backward chaining. These algorithms are very natural and run in linear time.
Forward chaining

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

$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
$A$
$B$
Forward chaining algorithm

function PL-FC-ENTAILS?(\(KB, q\)) returns true or false

inputs: \(KB\), the knowledge base, a set of propositional Horn clauses
\(q\), the query, a proposition symbol

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 in \(KB\)

while agenda is not empty do
  \(p \leftarrow \text{POP(agenda)}\)
  unless inferred[\(p\)] do
    inferred[\(p\)] \(\leftarrow\) true
    for each Horn clause \(c\) in whose premise \(p\) appears do
      decrement count[\(c\)]
      if count[\(c\)] = 0 then do
        if \(\text{HEAD}[c] = q\) then return true
        \(\text{PUSH(HEAD}[c], \text{agenda})\)
  return false
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example

Chapter 7.5-7.7
Forward chaining example
Forward chaining example
Forward chaining example
Proof of completeness

FC derives every atomic sentence that is entailed by $KB$

1. FC reaches a **fixed point** where no new atomic sentences are derived

2. Consider the final state as a model $m$, assigning true/false to symbols

3. Every clause in the original $KB$ is true in $m$
   
   **Proof:** Suppose a clause $a_1 \land \ldots \land a_k \Rightarrow b$ is false in $m$
   
   Then $a_1 \land \ldots \land a_k$ is true in $m$ and $b$ is false in $m$
   
   Therefore the algorithm has not reached a fixed point!

4. Hence $m$ is a model of $KB$

5. If $KB \models q$, $q$ is true in **every** model of $KB$, including $m$

   **General idea:** construct any model of $KB$ by sound inference, check $\alpha$
Backward chaining

Idea: work backwards from the query $q$:

- to prove $q$ by BC,
  - check if $q$ is known already, or
  - prove by BC all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
  1) has already been proved true, or
  2) has already failed
Backward chaining example

Diagram showing a backward chaining process with nodes labeled Q, P, M, L, A, and B.
Backward chaining example

Chapter 7.5-7.7
Backward chaining example

Diagram showing nodes and arrows.
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

Diagram showing relationships between variables Q, P, M, L, A, and B.
Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
   e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
   e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be much less than linear in size of KB
Resolution

Conjunctive Normal Form (CNF—universal)

conjunction of \textit{disjunctions} of \textit{literals}

\textit{clauses}

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

Resolution inference rule (for CNF):

\[
\begin{array}{c}
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n \\
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\end{array}
\]

where \(\ell_i\) and \(m_j\) are complementary literals. E.g.,

\[
P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2}
\]

\[
P_{1,3}
\]

Resolution is sound and complete for propositional logic
Conversion to CNF

\[ B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}) \]

1. Eliminate \( \Leftrightarrow \), replacing \( \alpha \Leftrightarrow \beta \) with \( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \).
   
   \[
   (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})
   \]

2. Eliminate \( \Rightarrow \), replacing \( \alpha \Rightarrow \beta \) with \( \neg \alpha \lor \beta \).
   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})
   \]

3. Move \( \neg \) inwards using de Morgan’s rules and double-negation:
   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})
   \]

4. Apply distributivity law (\( \lor \) over \( \land \)) and flatten:
   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
   \]
Resolution algorithm

Proof by contradiction, i.e., show $KB \land \neg \alpha$ unsatisfiable

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 $C_i, C_j$ 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
  $clauses \leftarrow clauses \cup new$
Resolution example

\[ KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \quad \alpha = \neg P_{1,2} \]
DPLL: backtracking++

Backtracking applied to SAT problems:
- variables are proposition symbols, clauses are constraints

Several key improvements:

1. **Early termination**: stop if all clauses true or any clause false
   e.g., \( \{A = true\} \) satisfies \((A \lor B) \land (A \lor C)\)

2. **Pure symbols**: symbol has same sign in all as-yet-unsatisfied clauses
   e.g., \( A \) and \( B \) are pure in \((A \lor \neg B) \land (\neg B \lor \neg C) \land (C \lor A)\)
   \[ \Rightarrow \] assign symbol to make literals true

3. **Unit clauses**: clause has exactly one as-yet-unfalsified literal
   e.g., if \( \{A = true\} \) already, \((\neg A \lor \neg B)\) is a unit clause
   \[ \Rightarrow \] assign symbol to make clause true (cf. forward chaining, MRV)
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 true

    P, value ← Find-Pure-Symbol(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])

    P, value ← Find-Unit-Clause(clauses, model)
    if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])

    P ← First(symbols); rest ← Rest(symbols)
    return DPLL(clauses, rest, [P = true|model]) or
          DPLL(clauses, rest, [P = false|model])

Highly optimized implementation + caching unsolvable subassignments
⇒ modern solvers handle tens of millions of clauses
⇒ practical for large hardware and medium software verification
Propositions and time

Suppose the wumpus-world agent wants to keep track of its location

A sentence such as $L_{1,1} \land \text{FacingRight} \land \text{Forward} \Rightarrow L_{2,1}$ doesn’t work: after one inference step, $L_{1,1}$ and $L_{2,1}$ are in KB!!

Changeable aspects of world need separate symbols for each time step

e.g., $L_{1,1}^1$ means “Agent is at [1,1] at time step 1”, and

$\ L_{1,1}^1 \land \text{FacingRight}^1 \land \text{Forward}^1 \Rightarrow L_{2,1}^2$

Reflex rules: for every $t$, we have, e.g., $\text{Glitter}^t \Rightarrow \text{Grab}^t$

⚠️ Need copies of all axioms involving temporal symbols for every time step (might be infinitely many!)
Tracking changes in the world

State estimation is the general task of keeping track of environment state given a stream of percepts.

For logic-based systems: maintain a representation of the set of all logically possible world states, given axioms and percepts.

Basic trick: successor-state axioms define truth of proposition at $t+1$ from propositions at $t$.

E.g., $Alive^t \Leftrightarrow \neg Scream^t \land Alive^{t-1}$

$L_{1,1}^t \Leftrightarrow (L_{1,1}^{t-1} \land (\neg Forward^{t-1} \lor Bump^t))$

$\lor (L_{1,2}^{t-1} \land (FacingDown^{t-1} \land Forward^{t-1}))$

$\lor (L_{2,1}^{t-1} \land (FacingLeft^{t-1} \land Forward^{t-1}))$
Boolean circuit agents

- Breeze
- Stench
- Glitter
- Bump
- Scream
- Forward
- TurnLeft
- TurnRight
- Grab
- Shoot
- Alive

\[ \text{Chapter 7.5-7.7} \]
Boolean circuit agents contd.

- Breeze
- Stench
- Glitter
- Bump
- Scream
- FacingLeft
- FacingDown
- L₁,₁
- L₂,₁
- L₁,₂
- Forward
- TurnLeft
- TurnRight
- Grab
- Shoot

\[ \neg \]
\[ \land \]
\[ \lor \]
Summary

Inference methods work by **theorem proving** or **model checking**

Forward, backward chaining are linear-time, complete for Horn clauses

Resolution is complete for propositional logic

DPLL is an efficient, complete model checker;
WalkSAT is incomplete but often very fast in practice

Circuit-based agents provide a simple way to handle time
but are usually less complete than inference-based agents