Deliberative Agents
Knowledge Representation I

Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Bioinformatics and Computational Biology Program
Center for Computational Intelligence, Learning, & Discovery
Iowa State University
honavar@cs.iastate.edu
www.cs.iastate.edu/~honavar/
www.cild.iastate.edu/
www.bcb.iastate.edu/
www.igert.iastate.edu

Intelligent behavior requires knowledge about the world

Deliberative Agents

- Intelligent behavior requires knowledge about the world
Deliberative Agents

- Intelligent behavior requires knowledge about the world
- **Procedural**, e.g., functions
  - Using knowledge = executing the procedure
- **Declarative**, e.g., facts
  - Using knowledge = performing inference

- Deliberative agents
  - Can represent and reason with knowledge
  - Exhibit *logical rationality*

Knowledge representation (KR) is a *surrogate*

A *declarative* knowledge representation
- Maps **facts that are true in the world** into sentences
- Reasoning is performed by manipulating **sentences** according to *sound* rules of inference
- The results of inference are **sentences** that correspond to **facts** that are true in the world
- This correspondence provides *semantic grounding* for the representation
- Allows agents to **substitute thinking for acting** in the world
  - **Known facts**: The coffee is hot; coffee is a liquid; a hot liquid will burn your tongue;
  - **Inferred fact**: Coffee will burn your tongue
Semantic Grounding

KR is a set of ontological commitments

- What does an agent care about?
  - Entities –
    - coffee, liquid, tongue
  - Properties –
    - being hot, being able to burn
  - Relationships
    - Coffee is a liquid
- KR involves abstraction, simplification
  - A representation is like a cartoon
KR involves a set of epistemological commitments

- **What can we know?**
  - Propositional logic
    - Is a proposition *true* or *false*?
  - Probability theory
    - What is the *probability* that a given proposition true?
  - Decision theory
    - Which choice among a set of candidate choices is the most *rational*?
- Logic of Knowledge
  - What does John *know* that Jim does not?

KR is a theory of intelligent reasoning

- **How can knowledge be encoded?**
  - Syntax
- **What does the encoded knowledge mean?**
  - Semantics (entailment)
  - Inferences that are sanctioned by the semantics
- **What can we infer from what we know?**
  - Inferences that can be performed (by rules of inference)
  - Soundness, completeness, efficiency
- **How can we manage inference?**
  - What should we infer from among the things we can infer?
KR formalisms

KR formalisms provide provision for describing
• Individuals
• Sets of individuals (classes)
• Properties of individuals
• Properties of classes
• Relationships between individuals
• Relationships between classes
• Actions and their effects
• Locations and events in space and time
• Uncertainty

KR formalisms

• Logical
  – e.g., Propositional Logic, First order logic, Description logic)
• Probabilistic
  – e.g., Bayesian networks
• Grammatical
  – e.g., Context free grammars
• Structured
  – e.g., frames – as in object-oriented programming
• Decision theoretic
• ...
KR is a medium for efficient computation

• Reasoning = computation
• KR involves tradeoffs between
  – Expressivity and tractability (decidability, efficiency)
  – General purpose reasoning versus special-purpose, domain-specific inference
  – Declarative versus procedural knowledge

KR is a medium of expression and communication

• If we assume shared
  – ontological commitments
  – KR formalism (syntax, semantics, reasoning)
• Then KR is a medium for
  – Expression
    • How general is it?
    • How precise is it?
    • Is it expressive enough?
  – Communication
    • Can we talk or think in the language?
    • What can we communicate with ease?
    • What things are difficult to communicate?
Logical Agents I – Propositional Logic

• Knowledge-based agents
• Logic in general - models and entailment
• Propositional (Boolean) logic
  – Syntax
  – Semantics
  – Equivalence, Validity, Satisfiability, Decidability
• Inference rules
  – Soundness, completeness
  – Resolution
• Inference procedures for automated theorem proving
  – Soundness, completeness, efficiency

Knowledge-Based Agents

• Knowledge-based agents represent and use knowledge about the world
Knowledge Base

- Knowledge base = set of sentences about the world in a declarative formal language
- Basic operations on the knowledge base
  - Tell
  - Ask
- Additional operations
  - Untell (in non monotonic logic)

A simple knowledge-based agent

```java
function KB-Agent(percept) returns an action
    static: KB, a knowledge base
    t, a counter, initially 0, indicating time
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

Knowledge based agents
- Are not arbitrary programs
- Are amenable to knowledge level description
  - what the agent knows
Logic as a Knowledge Representation Formalism

Logic is a declarative language to:
• Assert sentences representing facts that hold in a world W (these sentences are given the value true)
• Deduce the true/false values to sentences representing other aspects of W

Examples of Logics

• Propositional logic
  \( A \land B \Rightarrow C \)
• First-order predicate logic
  \( \forall x \exists y \text{ Mother}(y,x) \)
• Logic of Knowledge
  \( K(\text{John}, \text{Father}(\text{Zeus}, \text{Cronus})) \)
• Logic of Belief
  \( B(\text{John}, \text{Father}(\text{Zeus}, \text{Cronus})) \)

Note: You can believe things that are false, but you cannot know things that are false.
Components of Logical KR

- A logical formalism
  - Syntax for well-formed-formulae (wff)
  - Vocabulary of logical symbols (and, or, not, implies etc.)
  - Interpretation semantics for logical symbols
    - e.g. A or B holds in the world whenever A holds in the world or B holds in the world
- An ontology
  - Vocabulary of non logical symbols
    - objects, properties (e.g., A above), etc.
    - definitions of non-primitive symbols (e.g., iff)
    - axioms restricting the interpretation of primitive symbols (more on this later)
- Proof theory – sound rules of inference

Propositional Logic

- Atomic sentences - propositions
  - Propositions can be true or false
- Statements can be combined using logical connectives. Examples:
  - C = “It’s cold outside”
    - C is a proposition
  - O = “It’s October”
    - O is a proposition
  - If O then C
    - if it’s October then it’s cold outside
Propositional Logic: Syntax, logical symbols

The proposition symbols $P_1$, $P_2$ etc are sentences.

If $S$ is a sentence, $\neg S$ is a sentence (negation).

If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction).

If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction).

If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication).

We use extra-linguistic symbols (e.g., parentheses) to group sentences to avoid ambiguity.

Propositional Logic - Semantics

A Proposition

• does not have intrinsic meaning
• gets its meaning from correspondence with statements about the world (interpretation)

- e.g., proposition $B$ denotes the fact that battery is charged
• is True or False in a chosen interpretation
• Connectives do no intrinsic meaning – need interpretation
Propositional Logic – Model Theoretic Semantics

• Consider a logic with only two propositions:
  – *Rich, Poor*
  – denoting *Tom is rich* and *Tom is poor* respectively

• A *model* $M$ is a subset of the set $A$ of *atomic sentences* in the language

• By a model $M$ we mean the state of affairs in which
  – every atomic sentence that is in $M$ is *true* and
  – every atomic sentence that is not in $M$ is *false*

Example

\[ A = \{ \text{Rich, Poor} \} \]
\[ M_0 = \emptyset \]
\[ M_1 = \{ \text{Rich} \} \]
\[ M_2 = \{ \text{Poor} \} \]
\[ M_3 = \{ \text{Rich, Poor} \} \]

Propositional Logic: Semantics of Logical Symbols

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \Rightarrow Q$</th>
<th>$P \Leftrightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Note that $(P \Rightarrow Q) \Leftrightarrow (\neg P \lor Q)$
Model Theoretic Semantics

\[ A = \{ \text{Rich, Poor} \} \]
\[ M_0 = \{ \} \quad M_1 = \{ \text{Rich} \} \quad M_2 = \{ \text{Poor} \} \quad M_3 = \{ \text{Rich, Poor} \} \]

- Rich is True in \( M_1, M_3 \)
- Rich ∨ Poor is True in \( M_1, M_2, M_3 \)
- Rich ∧ Poor is True in \( M_3 \)  
  Hmm !!
- Rich ⇒ ¬Poor is True in \( M_0, M_1, M_2 \)

Proof Theory: Entailment

- We say that \( p \) entails \( q \)
  (written as \( p \models q \)) if \( q \) holds in every model in which \( p \) holds

\[ \mu_q = \text{set of models in which } q \text{ holds} \]
\[ \mu_p = \text{set of models in which } p \text{ holds} \]
\[ p \models q \text{ if it is the case that } \mu_p \subseteq \mu_q \]
Validity and satisfiability

- A sentence is **valid** if it is true in all models,
  - e.g., True, A ∨ ¬A, A ⇒ A, (A ∨ (A ⇒ B)) ⇒ B
- A sentence is **satisfiable** if it is true in some model
  - e.g., A ∨ B, C
- A sentence is **unsatisfiable** if it is true in no models
  - e.g., A ∧ ¬A
- Satisfiability is connected to inference via the following:
  - KB \models \alpha if and only if (KB ∧ ¬\alpha) is unsatisfiable
  - Useful for proof by contradiction
Logical Rationality

• An propositional logic based agent $A$ with a knowledge base $KB_A$ is justified in inferring $q$ if it is the case that

$$KB_A \models q$$

• How can the agent $A$ decide whether in fact $KB_A \models q$ ?
  – Model checking
    • Enumerate all the models in which $KB_A$ holds
    • Enumerate all the models in which $q$ holds
    • Check whether $KB_A \subseteq \mu_q$
  – Inference algorithm based on inference rules

Inference Rule

• An inference rule $\{\xi, \psi\} \vdash \varphi$ consists of
  – 2 sentence patterns $\xi$ and $\psi$ called the premises and
  – one sentence pattern $\varphi$ called the conclusion

• If $\xi$ and $\psi$ match two sentences of KB then
  – the corresponding $\varphi$ can be inferred according to the rule

• Given a set of inference rules $I$ and a knowledge base $KB$ inference is the process of successively applying inference rules from $I$ to $KB$, each rule application adding its conclusion to $KB$
Inference rules

- **Modus ponens**
  
  \[ p \Rightarrow q \]
  
  \[ p \]
  
  \[ q \]

  *Modus ponens* derives only inferences sanctioned by entailment

  *Modus ponens* is sound

- **Loony tunes**

  \[ \text{friday} \]
  
  \[ p \Rightarrow q \]
  
  \[ q \]
  
  \[ q \]
  
  \[ p \]

  *Loony tunes* can derive inferences that are not sanctioned by entailment

  *Loony tunes* is not sound

---

**Example: Inference using Modus Ponens**

\[ \{ p \Rightarrow q , p \} \vdash q \]

\[ \{ \xi , \psi \} \vdash \varphi \]

KB:

- Battery-OK, Bulbs-OK \( \Rightarrow \) Headlights-Work
- Battery-OK, Starter-OK, ¬Empty-Gas-Tank \( \Rightarrow \) Engine-Starts
- Engine-Starts, ¬Flat-Tire, Headlights-Work \( \Rightarrow \) Car-OK

Battery-OK, Bulbs-OK, Starter-OK, ¬Empty-Gas-Tank, ¬Flat-Tire

Ask

Car-OK?
Soundness and Completeness of an inference rule \( |- \)

- We write \( p \vdash q \) to denote that that \( p \) can be inferred from \( q \) using the inference rule \( |- \).

An inference rule \( |- \) is said to be

- **Sound** if whenever \( p \vdash q \), it is also the case that \( p \models q \).
- **Complete** if whenever \( p \models q \), it is also the case that \( p \vdash q \).

We can show that **modus ponens** is sound, but **not complete** unless the KB is **Horn** i.e., the KB can be written as a collection of sentences of the form

\[
 a_1 \land a_2 \land \ldots \land a_n \Rightarrow b
\]

where each \( a_i \) and \( b \) are atomic sentences.
Unsound inference rules are not necessarily useless!

Abduction (Charles Peirce) is not sound, but useful in diagnostic reasoning or hypothesis generation

\[
p \Rightarrow q
\]

\[
q \quad p
\]

**BlockedArtery** \(\Rightarrow\) **HeartAttack**

**HeartAttack**

\[\text{BlockedArtery} \]

Logical equivalence

- Two sentences are *logically equivalent* iff true in same set of models or \(\alpha \equiv \beta\) iff \(\alpha \models \beta\) and \(\beta \models \alpha\).

\[
(\alpha \land \beta) \equiv (\beta \land \alpha) \quad \text{commutativity of } \land
\]

\[
(\alpha \lor \beta) \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor
\]

\[
((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land
\]

\[
((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor
\]

\[
\neg(\neg \alpha) \equiv \alpha \quad \text{double-negation elimination}
\]

\[
(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition}
\]

\[
(\alpha \Rightarrow \beta) \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination}
\]

\[
(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}
\]

\[
\neg(\neg \alpha \land \beta) \equiv (\neg \alpha \lor \neg \beta) \quad \text{de Morgan}
\]

\[
\neg(\neg \alpha \lor \beta) \equiv (\neg \alpha \land \neg \beta) \quad \text{de Morgan}
\]

\[
(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor
\]

\[
(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\]
Searching for proofs

• Finding proofs can be cast as a search problem

• Search can be
  – forward (forward chaining) to derive goal from KB
  – or backward (backward chaining) from the goal

• Searching for proofs
  – is not more efficient than enumerating models in the worst case
  – Can be more efficient in practice with the use of suitable heuristics

• Standard propositional logic is monotonic
  – the set of entailed sentences can only increase as inferred facts are added to KB

\[
\text{for any sentences } \alpha \text{ and } \beta : \quad \text{if } KB \models \alpha \text{ then } KB \land \beta \models \alpha
\]

Soundness and Completeness of an inference algorithm

• An inference algorithm starts with the KB and applies applicable inference rules until the desired conclusion is reached

• An inference algorithm is sound if it uses a sound inference rule

• An inference algorithm is complete if it uses a complete inference rule together with a complete search procedure.
Inference by model checking is sound and complete for propositional Knowledge Bases

- Depth-first enumeration of all models is sound and complete
  
  ```
  function TT-ENTAILS?(KB, α) returns true or false
  symbols ← a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, symbols, [])
  
  function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
    else return true
  else do
    P ← FIRST(symbols); rest ← REST(symbols)
    return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model) and
    TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
  
  For n symbols, worst case time complexity is \(O(2^n)\), space complexity is \(O(n)\).
- In practice, much more efficient inference possible

Sound Inference rules in PL

- Modus Ponens
  
  \[ \alpha \Rightarrow \beta, \alpha \quad \Rightarrow \quad \beta \]

- And-elimination: from a conjunction any conjunct can be inferred:
  
  \[ \alpha \land \beta \quad \Rightarrow \quad \alpha \]

- All logical equivalences can be used as inference rules
  
  \[ \alpha \leftrightarrow \beta \quad \Rightarrow \quad (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \]

- An inference procedure involves repeated application of applicable inference rules. An inference procedure is sound if it uses only sound inference rules.

Vasant Honavar, 2009
Proof

- The proof of a sentence \( \alpha \) from a set of sentences \( KB \) is the derivation of \( \alpha \) obtained through a series of applications of sound inference rules to \( KB \).
- Modus Ponens is sound for Propositional Logic.
- Modus Ponens is complete for Horn KB:
  - Whenever \( KB \models q \) repeated application of Modus ponens is guaranteed to infer \( q \) if the KB consists of only Horn Clauses, i.e., only sentences of the form:
    \[ p_1 \land p_2 \land \ldots \land p_n \Rightarrow q \]
  - Complexity of inference is polynomial in the size of the Horn KB.
- Modus Ponens is not complete for arbitrary Propositional KB.

Proof

- The proof of a sentence \( \alpha \) from a set of sentences \( KB \) is the derivation of \( \alpha \) obtained through a series of applications of sound inference rules to \( KB \).
- \( KB \models \alpha \) if and only if \( \{ KB, \neg \alpha \} \) is unsatisfiable.
- Proving \( \alpha \) from \( KB \) is equivalent to deriving a contradiction from \( KB \) augmented with the negation of \( \alpha \).
- Proof typically requires transformation of sentences into a normal form.
Iowa State University  
Department of Computer Science  
Artificial Intelligence Research Laboratory

**Inference with Horn KB**

- Each Horn clause has only one positive literal
- Inference can be done with forward or backward chaining
- Entailment decidable in time linear in the size of propositional KB
- Prolog

Vasant Honavar, 2009.

---

**Forward chaining**

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

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B & 
\end{align*}
\]
Forward chaining algorithm

- Forward chaining is sound and complete for Horn KB

```python
function PC-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
```

Vasant Honavar, 2009.
Forward chaining example
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 $m$
   \[ a_1 \land \ldots \land a_k \Rightarrow b \]
4. Hence $m$ is a model of $KB$
5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
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
Backward chaining example
Backward chaining example

![Diagram of backward chaining example]
Backward chaining example
Backward chaining example

Backward chaining example
Forward vs. backward chaining

- FC is **data-driven**, 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?
- Run time of BC can be, in practice, **much less** than linear in size of **KB**
Towards a sound and complete inference rule for propositional KB

\[ \{ p \Rightarrow q, p \}\{q\} \quad \text{Modus ponens is sound} \]

\( p \) does not have to be an atomic sentence

\[
\begin{align*}
  a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_i \land a_{i+1} \cdot \cdot \cdot a_n \Rightarrow b \\
  T \Rightarrow a_i \\
  \frac{a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n \Rightarrow b}{a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n \Rightarrow b}
\end{align*}
\]

\( q \) may be contingent on other facts

\[
\begin{align*}
  a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_i \land a_{i+1} \cdot \cdot \cdot a_n \Rightarrow b \\
  d_1 \land d_2 \cdot \cdot \cdot d_m \Rightarrow c \\
  \frac{a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n \land d_1 \land d_2 \cdot \cdot \cdot d_m \Rightarrow b}{a_1 \land a_2 \land a_3 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n \land d_1 \land d_2 \cdot \cdot \cdot d_m \Rightarrow b}
\end{align*}
\]

Assume \( a_i = c \)

---

Resolution principle

\( b, c \) do not have to be an atomic sentences

\[
\begin{align*}
  a_1 \land a_2 \land \cdot \cdot \cdot a_{i-1} \land a_i \land a_{i+1} \land \cdot \cdot \cdot a_n \Rightarrow b_1 \lor b_2 \lor \cdot \cdot \cdot \lor b_k \\
  d_1 \land d_2 \cdot \cdot \cdot d_m \Rightarrow c \quad (\text{assume } a_i = c)
\end{align*}
\]

\[
\begin{align*}
  (a_1 \land a_2 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n) \land (d_1 \land d_2 \cdot \cdot \cdot d_m) \Rightarrow b_1 \lor b_2 \cdot \cdot \cdot \lor b_k
\end{align*}
\]

As before, this rule can be shown to be sound.

\[
\begin{align*}
  a_1 \land a_2 \land \cdot \cdot \cdot a_{i-1} \land a_i \land a_{i+1} \land \cdot \cdot \cdot a_n \Rightarrow b_1 \lor b_2 \lor \cdot \cdot \cdot \lor b_k \\
  d_1 \land d_2 \cdot \cdot \cdot d_m \Rightarrow c_1 \lor c_2 \lor \cdot \cdot \cdot c_{j-1} \lor c_j \lor c_{j+1} \lor \cdot \cdot \cdot c_l
\end{align*}
\]

\[
\begin{align*}
  (a_1 \land a_2 \cdot \cdot \cdot a_{i-1} \land a_{i+1} \cdot \cdot \cdot a_n) \land (d_1 \land \cdot \cdot \cdot d_m) \Rightarrow (b_1 \lor b_2 \cdot \cdot \cdot \lor b_k) \lor (c_1 \lor c_2 \lor \cdot \cdot \cdot c_{j-1} \lor c_{j+1} \lor \cdot \cdot \cdot c_l)
\end{align*}
\]

(assume \( c_j = a_i \))

Resolution is sound and complete for propositional KB
• Resolution is sound and complete for propositional $KB$
• *Formal Proof Omitted*

Applying resolution

• Transform $KB$ into an *equivalent* Conjunctive normal form (CNF)
  – Each sentence in $KB$ is a disjunction of literals or their negations using known logical equivalences
  – $KB$ is a conjunction of disjunctions
Transformation to Clause Form (CNF)

Example:

\[(A \lor \neg B) \Rightarrow (C \land D)\]

1. Eliminate \(\Rightarrow\)
\[
\neg(A \lor \neg B) \lor (C \land D)
\]

2. Reduce scope of \(\neg\)
\[
(\neg A \land B) \lor (C \land D)
\]

3. Distribute \(\lor\) over \(\land\)
\[
(\neg A \lor (C \land D)) \land (B \lor (C \land D))
\]
\[
(\neg A \lor C) \land (\neg A \lor D) \land (B \lor C) \land (B \lor D)
\]

Set of clauses:
\[
\{\neg A \lor C, \neg A \lor D, B \lor C, B \lor D\}
\]

Applying resolution

Example

\(Engine-Starts \land \neg Flat-Tire \land Headlights-Work \Rightarrow Car-OK\)

is transformed to

\(\neg Engine-Starts \lor Flat-Tire \lor \neg Headlights-Work \lor Car-OK\)

\(Battery-OK \land Bulbs-OK\)

is transformed to

\(\{Battery-OK, Bulbs-OK\}\)
Example

Given:
- Engine-Starts v Flat-Tire v ¬ Headlights-Work v Car-OK
  ¬ Battery-OK v ¬ Bulbs-OK v Headlights-Work

Inferred:
- Engine-Starts v Flat-Tire v Car-OK ¬ Battery-OK v ¬ Bulbs-OK

Resolution by Refutation Algorithm

Add negation of goal to KB, derive empty clause (contradiction)

RESOLUTION-REFUTATION (KB, α)

clauses ← set of clauses obtained from KB and ¬ α
new ← {}
Repeat:
  For each C, C’ in clauses do
    res ← RESOLVE(C, C’)
    If res contains the empty clause then return yes
    new ← new U res
    If new ⊆ clauses then return no
  clauses ← clauses U new
Exercise: Prove *Car-OK* using resolution by refutation

**KB:**

\[
\begin{align*}
&\text{Battery-OK} \land \text{Bulbs-OK} \Rightarrow \text{Headlights-Work} \\
&\text{Battery-OK} \land \text{Starter-OK} \land \neg \text{Empty-Gas-Tank} \Rightarrow \text{Engine-Starts} \\
&\text{Engine-Starts} \land \neg \text{Flat-Tire} \land \text{Headlights-Work} \Rightarrow \text{Car-OK} \\
&\text{Battery-OK} \land \text{Bulbs-OK} \\
&\text{Starter-OK} \land \neg \text{Empty-Gas-Tank} \land \neg \text{Flat-Tire}
\end{align*}
\]

Ask

\[\text{Car-OK}\]
The Davis-Putnam-Logemann-Loveland (DPLL) algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

- Improvements over truth table enumeration:
  - Early termination
  - Pure symbol heuristic
  - Unit clause heuristic

DPLL algorithm

Improvements over truth table enumeration:

- Early termination
  - A clause is true if any literal is true. A sentence is false if any clause is false.
  - E.g. $(\neg B \lor P \lor Q) \land (\neg P \lor B) \land (\neg Q \lor B)$
DPLL algorithm

Improvements over truth table enumeration:

– Pure symbol heuristic
  • Pure symbol always appears with the same "sign" in all clauses
  • e.g., in the three clauses (A ∨ ¬B), (¬B ∨ ¬C), (C ∨ A),
    • A and B are pure, C is impure
    • Assign a pure symbol so that their literals are true

– Unit clause heuristic
  Unit clause: only one literal in the clause or only one literal which has not yet received a value. The only literal in a unit clause must be true. First do this assignments before continuing with the rest (unit propagation!)
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, |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|)

Vasant Honavar, 2009.

The WalkSAT algorithm

• Incomplete, local search algorithm
• Evaluation function:
  – The min-conflict heuristic of minimizing the number of unsatisfied clauses
• Steps are taken in the space of complete assignments, flipping the truth value of one variable at a time
• Balance between greediness and randomness.
  – To avoid local minima

Vasant Honavar, 2009.
The WalkSAT algorithm

\begin{function}
\textbf{WalkSAT}(clauses, p, max-flips) returns a satisfying model or failure
\begin{algorithmic}
\Function {WalkSAT} {clauses, p, max-flips}
\State \textbf{inputs:} clauses, a set of clauses in propositional logic
\hspace{1em} p, the probability of choosing to do a “random walk” move
\hspace{1em} max-flips, number of flips allowed before giving up
\State \text{model} \leftarrow \text{a random assignment of true/false to the symbols in clauses}
\For {i = 1 to max-flips}
\If {model satisfies clauses} \textbf{return} model
\State \text{clause} \leftarrow \text{a randomly selected clause from clauses that is false in model}
\State \text{with probability } p \text{ flip the value in model of a randomly selected symbol from clause}
\Else \text{flip whichever symbol in clause maximizes the number of satisfied clauses}
\EndIf
\EndFor
\Return \text{failure}
\EndFunction
\end{algorithmic}
\end{function}

Under-constrained problems are easy: e.g. n-queens in CSP

In SAT: e.g.,
\[(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)\]

Increase in complexity by keeping the number of symbols fixed and increasing the number of clauses (add constraints)

- Hard problems seem to cluster near \(m/n = 4.3\) (critical point)
Hard satisfiability problems

Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$
Circuit-based implementation

- Intelligence without representation? (Brooks)
- Circuit-based agents
  - Reflex agents with internal state
  - Implemented using sequential circuits (logic gates plus registers)
  - Circuits evaluated in dataflow fashion
  - Inference linear in circuit size
  - Circuit size may be exponentially larger than the inference based agent’s KB in some environments
- There are tradeoffs between inference-based and circuit-based agents
- Best of both worlds –
  - Inference based with routinely used inferences compiled into circuits