CS477 Formal Software Dev Methods

Elsa L Gunter
2112 SC, UIUC
tgunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures
by Mahesh Vishwanathan, and by Gul Agha

January 31, 2018
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
Model Checking for Propositions

- Problem: Would like an efficient way to answer for a given proposition $P$:
  - Does a given valuation satisfy $P$?
    - Valuation gives specific values for variables in $P$
Problem: Would like an efficient way to answer for a given proposition $P$:

- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?

Note: A general algorithm to answer the last can be used to answer the second and vice versa.

Difficulty: Answering if $P$ is satisfiable is NP-complete.

Algorithms exist with good performance in general practice. BDDs are one such.
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?

Note: A general algorithm to answer the last can be used to answer the second and vice versa.

Difficulty: Answering if $P$ is satisfiable is NP-complete

Algorithms exist with good performance in general practice

BDDs are one such
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?
  - $P$ is true in all valuations

Note: A general algorithm to answer the last can be used to answer the second and vice versa.
Problem: Would like an efficient way to answer for a given proposition $P$:

- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?
  - $P$ is true in all valuations

Note: A general algorithm to answer the last can be used to answer the second and vice versa.
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?
  - $P$ is true in all valuations

**Note:** A general algorithm to answer the last can be used to answer the second and vice versa.

Difficulty: Answering if $P$ is satisfiable is NP-complete
Problem: Would like an efficient way to answer for a given proposition $P$:

- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?
  - $P$ is true in all valuations

Note: A general algorithm to answer the last can be used to answer the second and vice versa.

Difficulty: Answering if $P$ is satisfiable is NP-complete

Algorithms exist with good performance in general practice
Problem: Would like an efficient way to answer for a given proposition $P$:
- Does a given valuation satisfy $P$?
  - Valuation gives specific values for variables in $P$
- Is $P$ satisfiable?
  - Does there exist a valuation that makes $P$ true?
- Is $P$ a tautology?
  - $P$ is true in all valuations

Note: A general algorithm to answer the last can be used to answer the second and vice versa.

Difficulty: Answering if $P$ is satisfiable is NP-complete

Algorithms exist with good performance in general practice

BDDs are one such
Binary decision tree is a (rooted, directed) edge and vertex labeled tree with two types of vertices – internal nodes, and leaves – such that:

- Leaves are labeled by true or false.
Binary decision tree is a (rooted, directed) edge and vertex labeled tree with two types of vertices – *internal nodes*, and *leaves* – such that:

- Leaves are labeled by *true* or *false*.
- Leaves have *no* out edges.
Binary decision tree is a (rooted, directed) edge and vertex labeled tree with two types of vertices – internal nodes, and leaves – such that:

- Leaves are labeled by true or false.
- Leaves have no out edges
- Internal nodes are labeled by atomic propositions (variables)
Binary decision tree is a (rooted, directed) edge and vertex labeled tree with two types of vertices – internal nodes, and leaves – such that:

- Leaves are labeled by true or false.
- Leaves have no out edges.
- Internal nodes are labeled by atomic propositions (variables).
- Internal nodes have exactly two out edges.
Binary Decision Trees

- **Binary decision tree** is a (rooted, directed) edge and vertex labeled tree with two types of vertices – *internal nodes*, and *leaves* – such that:
  - Leaves are labeled by *true* or *false*.
  - Leaves have no out edges.
  - Internal nodes are labeled by *atomic propositions* (variables).
  - Internal nodes have exactly two out edges.
  - Left edges labeled *false* and right edges labeled *true*. 

Elsa L Gunter
CS477 Formal Software Dev Methods
January 31, 2018 3 / 17
Binary Decision Trees

- **Binary decision tree** is a (rooted, directed) edge and vertex labeled tree with two types of vertices – **internal nodes**, and **leaves** – such that:
  - Leaves are labeled by **true** or **false**.
  - Leaves have **no** out edges
  - Internal nodes are labeled by **atomic propositions** (variables)
  - Internal nodes have **exactly two** out edges
  - Left edges labeled **false** and right edges labeled **true**.
    - Think 0 and 1
Binary decision tree is a (rooted, directed) edge and vertex labeled tree with two types of vertices – *internal nodes*, and *leaves* – such that:

- Leaves are labeled by *true* or *false*.
- Leaves have no out edges.
- Internal nodes are labeled by *atomic propositions* (variables).
- Internal nodes have exactly two out edges.
- Left edges labeled *false* and right edges labeled *true*.
  - Think 0 and 1
- For each path (branch) in the tree, each atomic proposition may label at most one vertex of that path.
Binary decision trees can record the set of all models (and non-models) of a proposition.

- Path records a valuation: out edge label gives value for variable labeling an internal node.

Each valuation matches exactly one branch. More than one valuation may (will) match a given branch.
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
  - Leaf label says whether a valuation assigning those values to those variables

Is a model (true, therefore accept the valuation)
Or not a model (false, therefore reject the valuation)

Each valuation matches exactly one branch
More than one valuation may (will) match a given branch
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
  - Leaf label says whether a valuation assigning those values to those variables
    - Is a model (true, the tree accepts the valuation)
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
  - Leaf label says whether a valuation assigning those values to those variables
    - Is a model (true, the tree accepts the valuation)
    - Or not a model (false, the tree rejects the valuation)
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
  - Leaf label says whether a valuation assigning those values to those variables
    - Is a model (true, the tree accepts the valuation)
    - Or not a model (false, the tree rejects the valuation)
  - Each valuation matches exactly one branch
Binary Decision Trees

- Binary decision trees can record the set of all models (and non-models) of a proposition
  - Path records a valuation: out edge label gives value for variable labeling an internal node
  - Any variable not on path can have any value
  - Leaf label says whether a valuation assigning those values to those variables
    - Is a model (true, the tree accepts the valuation)
    - Or not a model (false, the tree rejects the valuation)
- Each valuation matches exactly one branch
- More than one valuation may (will) match a given branch
Example:

\[(A \land B) \lor (\neg C)\]
Example: Different Variable Ordering - Different Tree

\((A \land B) \lor (\neg C)\)
Example: Many Logically Equivalent Trees

\[(A \land B) \lor (\neg C)\]
Still have constants \( \{T, F\} \)

Still have countable set \( AP \) of propositional variables a.k.a. atomic propositions

Only one ternary connective: the conditional \texttt{if\_then\_else}\_

- First argument only a variable
- Second and third arguments propositions
- Example

\[
\text{if } C \text{ then if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F \text{ else } T
\]

Represents the last tree above
Semantics for Conditional Propositional Logic

- Define when a valuation \( v \) satisfies a conditional proposition by

\[
\begin{align*}
\forall t \in \{T, F\}, \quad v \models t & \iff v \models \text{if } A \text{ then } P_t \text{ else } P_f \\
& \iff v(A) = \text{true} \land v \models P_t \text{ or } \quad v(A) = \text{false} \land v \models P_f
\end{align*}
\]

- Example: let \( v = \{A \leftrightarrow \text{true}, B \leftrightarrow \text{true}, C \leftrightarrow \text{true}\} \)

\[
\begin{align*}
v \models \text{if } C \text{ then if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F \text{ else } T
\end{align*}
\]
Define when a valuation \( \nu \) satisfies a conditional proposition by

\[
\nu \models True \\
\nu \not\models False \\
\nu \models if\ A\ then\ P_t\ else\ P_f\ \text{iff} \\
\nu(A) = true\ \text{and}\ \nu \models P_t\ \text{or} \\
\nu(A) = false\ \text{and}\ \nu \models P_f
\]

Example: let \( \nu = \{ A \mapsto true, B \mapsto true, C \mapsto true \} \)

\[
\nu \models if\ C\ then\ if\ B\ then\ if\ A\ then\ True\ else\ False\ else\ False\ else\ True
\]

since \( \nu(C) = true\ \text{and} \)

\[
\nu \models if\ B\ then\ if\ A\ then\ True\ else\ False\ else\ False
\]
Semantics for Conditional Propositional Logic

- Define when a valuation $\nu$ satisfies a conditional proposition by

  $\nu \models \text{if } A \text{ then } P_t \text{ else } P_f$ iff

  $\nu(A) = \text{true}$ and $\nu \models P_t$ or

  $\nu(A) = \text{false}$ and $\nu \models P_f$

- Example: let $\nu = \{A \mapsto \text{true}, B \mapsto \text{true}, C \mapsto \text{true}\}$

  $\nu \models \text{if } C \text{ then if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F \text{ else } T$ since

  $\nu(C) = \text{true}$ and

  $\nu \models \text{if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F$ since

  $\nu(B) = \text{true}$ and

  $\nu \models \text{if } A \text{ then } T \text{ else } F$
Semantics for Conditional Propositional Logic

Define when a valuation \( v \) satisfies a conditional proposition by

\[
v \models \begin{cases} 
T & \text{if } A \text{ then } P_t \text{ else } P_f \\
F & \text{if } A \text{ then } P_t \text{ else } P_f 
\end{cases}
\]

iff

\[
v(A) = \text{true and } v \models P_t \text{ or } v(A) = \text{false and } v \models P_f
\]

Example: let \( v = \{ A \mapsto \text{true}, B \mapsto \text{true}, C \mapsto \text{true} \} \)

\[
v \models \begin{cases} 
\text{true} & \text{if } C \text{ then if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F \text{ else } T \\
F & \text{if } C \text{ then if } B \text{ then if } A \text{ then } T \text{ else } F \text{ else } F \text{ else } T 
\end{cases}
\]

since

\[
v(C) = \text{true and}
\]

\[
v \models \begin{cases} 
\text{true} & \text{if } B \text{ then if } A \text{ then } T \text{ else } F \\
F & \text{if } B \text{ then if } A \text{ then } T \text{ else } F 
\end{cases}
\]

since

\[
v(B) = \text{true and}
\]

\[
v \models \begin{cases} 
\text{true} & \text{if } A \text{ then } T \text{ else } F \\
F & \text{if } A \text{ then } T \text{ else } F 
\end{cases}
\]

since

\[
v(A) = \text{true and } v \models T
\]
Translating Original Propositions into if_then_else

- Start with proposition $P_0$ with variables $v_1, \ldots v_n$
- $P[c/v]$ is the proposition resulting from replacing all occurrences of variable $v$ with constant $c$
- Let $\overline{P}$ be the result of evaluating every subexpression of $P$ containing no variables
- Let $P_1 = if \ v_1 \ then \ P_0[T/v_1] \ else \ P_0[F/v_1]$
- Let $P_i = if \ v_i \ then \ P_{i-1}[T/v_i] \ else \ P_{i-1}[F/v_i]$
- $P_n$ is logically equivalent to $P$, but only uses if_then_else_.
  - Valuation satisfies $P$ if and only if it satisfies $P_n$
  - $P_n$ depends on the order of variables $v_1, \ldots v_n$
  - $P_n$ directly corresponds to a binary decision tree
Example:

\[ P = (A \land B) \lor (\neg C), \text{ variables } \{A, B, C\}, \ A < B < C \]

\[ P_0 = (A \land B) \lor (\neg C) \]
$P = (A \land B) \lor (\neg C)$, variables $\{A, B, C\}$, $A < B < C$

$P_0 = (A \land B) \lor (\neg C)$

$P_1 = \text{if } A \text{ then } (T \land B) \lor (\neg C) \text{ else } (F \land B) \lor (\neg C)$
Example:

\[ P = (A \land B) \lor (\neg C), \text{ variables } \{A, B, C\}, \ A < B < C \]

\[ P_0 = (A \land B) \lor (\neg C) \]
\[ P_1 = \text{if } A \text{ then } (T \land B) \lor (\neg C) \text{ else } (F \land B) \lor (\neg C) \]
\[ P'_2 = \text{if } B \text{ then } (\text{if } A \text{ then } (T \land T) \lor (\neg C) \text{ else } (F \land T) \lor (\neg C)) \text{ else } (\text{if } A \text{ then } (T \land F) \lor (\neg C) \text{ else } (F \land F) \lor (\neg C)) \]
Example:

\[ P = (A \land B) \lor (\neg C), \text{ variables } \{A, B, C\}, A < B < C \]

\[ P_0 = (A \land B) \lor (\neg C) \]
\[ P_1 = \text{if } A \text{ then } (T \land B) \lor (\neg C) \text{ else } (F \land B) \lor (\neg C) \]
\[ P'_2 = \text{if } B \text{ then } (\text{if } A \text{ then } (T \land T) \lor (\neg C) \text{ else } (F \land T) \lor (\neg C)) \]
\[ \hspace{1cm} \text{else } (\text{if } A \text{ then } (T \land F) \lor (\neg C) \text{ else } (F \land F) \lor (\neg C)) \]
\[ P_2 = \text{if } B \text{ then } (\text{if } A \text{ then } T \lor (\neg C) \text{ else } F \lor (\neg C)) \]
\[ \hspace{1cm} \text{else } (\text{if } A \text{ then } F \lor (\neg C) \text{ else } F \lor (\neg C)) \]
Example:

\[ P = (A \land B) \lor (\neg C), \text{ variables } \{ A, B, C \}, A < B < C \]

\[ P_0 = (A \land B) \lor (\neg C) \]
\[ P_1 = \text{if } A \text{ then } (T \land B) \lor (\neg C) \text{ else } (F \land B) \lor (\neg C) \]
\[ P'_2 = \text{if } B \text{ then } \begin{align*}
\text{if } A \text{ then } (T \land T) \lor (\neg C) \text{ else } (F \land T) \lor (\neg C) \\
\text{else } \begin{align*}
\text{if } A \text{ then } (T \land F) \lor (\neg C) \text{ else } (F \land F) \lor (\neg C)
\end{align*}
\end{align*} \]
\[ P_2 = \text{if } B \text{ then } \begin{align*}
\text{if } A \text{ then } T \lor (\neg C) \text{ else } F \lor (\neg C) \\
\text{else } \begin{align*}
\text{if } A \text{ then } F \lor (\neg C) \text{ else } F \lor (\neg C)
\end{align*}
\end{align*} \]
\[ P'_3 = \text{if } C \text{ then } \begin{align*}
\text{if } B \text{ then } \begin{align*}
\text{if } A \text{ then } T \lor (\neg T) \text{ else } F \lor (\neg T) \\
\text{else } \begin{align*}
\text{if } A \text{ then } F \lor (\neg T) \text{ else } F \lor (\neg T)
\end{align*}
\end{align*} \]
\[ \text{else } \begin{align*}
\text{if } B \text{ then } \begin{align*}
\text{if } A \text{ then } T \lor (\neg F) \text{ else } F \lor (\neg F) \\
\text{else } \begin{align*}
\text{if } A \text{ then } F \lor (\neg F) \text{ else } F \lor (\neg F)
\end{align*}
\end{align*}
\end{align*} \]
Example:

\[ P = (A \land B) \lor (\neg C), \text{ variables } \{A, B, C\}, A < B < C \]

\[ P_0 = (A \land B) \lor (\neg C) \]
\[ P_1 = \text{ if } A \text{ then } (T \land B) \lor (\neg C) \text{ else } (F \land B) \lor (\neg C) \]
\[ P_2 = \text{ if } B \text{ then } (\text{ if } A \text{ then } (T \land T) \lor (\neg C) \text{ else } (F \land T) \lor (\neg C)) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } (T \land F) \lor (\neg C) \text{ else } (F \land F) \lor (\neg C)) \]
\[ P_2 = \text{ if } B \text{ then } (\text{ if } A \text{ then } T \lor (\neg C) \text{ else } F \lor (\neg C)) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } F \lor (\neg C) \text{ else } F \lor (\neg C)) \]
\[ P_3 = \text{ if } C \text{ then } (\text{ if } B \text{ then } (\text{ if } A \text{ then } T \lor (\neg T) \text{ else } F \lor (\neg T)) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } F \lor (\neg T) \text{ else } F \lor (\neg T)) \]
\[ \quad \text{ else } (\text{ if } B \text{ then } (\text{ if } A \text{ then } T \lor (\neg F) \text{ else } F \lor (\neg F)) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } F \lor (\neg F) \text{ else } F \lor (\neg F)) \]
\[ P_3 = \text{ if } C \text{ then } (\text{ if } B \text{ then } (\text{ if } A \text{ then } T \text{ else } F) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } F \text{ else } F) \]
\[ \quad \text{ else } (\text{ if } B \text{ then } (\text{ if } A \text{ then } T \text{ else } T) \]
\[ \quad \text{ else } (\text{ if } A \text{ then } T \text{ else } T) \]
Example, cont.

\[ P_3 = \text{if } C \text{ then (if } B \text{ then (if } A \text{ then } T \text{ else } F) \text{ else (if } A \text{ then } F \text{ else } F)) \]

\[ \text{else (if } B \text{ then (if } A \text{ then } T \text{ else } T) \text{ else (if } A \text{ then } T \text{ else } T)) \]

\[ P_3 \] corresponds to second binary decision tree given earlier

- Any proposition in strict \texttt{if\_then\_else\_} form corresponds directly to a binary decision tree that accepts exactly the valuations that satisfy (model) the proposition.
Binary decision trees may contain (much) redundancy

- Binary Decision Diagram (BDD): Replace trees by (rooted) directed acyclic graphs
- Require all other conditions still hold
- Generalization of binary decision trees
- Allows for sharing of common subtrees.
- Accepts / rejects valuations as with binary decision trees.
Problem: given proposition may correspond to many different BDDs

How to create a (compact) canonical BDD for a proposition such that two different propositions are logically equivalent if and only if they have the same (isomorphic) canonical BDD

Start: order propositional variables $v_i < v_j$.

Bryant showed you can obtain such a canonical BDD by requiring
- Variables should appear in order on each path for root to leaf
- No distinct duplicate (isomorphic) subtrees (including leaves)
Achieving Canonical Form

- Start with an Ordered BDD (all edges in correct order)
- Repeat following until none apply
- **Remove duplicate leaves:** Eliminate all but one leaf with a given label and redirect all edges to the eliminated leaves to the remaining one
- **Remove duplicate nonterminals:** If node $n$ and $m$ have the same variable label, their left edges point to the same node and their right edges point to the same node, remove one and redirect edges that pointed to it to the other
- **Remove redundant tests:** If both out edges of node $n$ point to node $m$, eliminate $n$ and redirect all edges coming into $n$ to $m$
- Bryant gave procedure to do the above that terminates in linear time