SS 2017
Software Verification
CTL model checking, BDDs

Prof. Dr. Holger Schlingloff \(^1,^2\)
Dr. Esteban Pavese \(^1\)

\(^1\) Institut für Informatik der Humboldt Universität
\(^2\) Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Recap

- What is a tableau?
- How is it interpreted?
- Propositional closing rules?
- LTL tableaus?
- Closing rules?
- Model generation from an LTL tableau?
- Connection to LTL model checking?
CTL model checking

- For each LTS/model there is exactly one computation tree
  - CTL model checking works directly on the model (no need to extract computation sequences)
- For all subformulas of a formula and all states of a given model, mark whether the state satisfies the subformula
  - iteration on formulas according to their inductive definition
  - if \( p \) is an atomic proposition, then \( p^M = I(p) \)
  - \( \bot^M = \{\} \)
  - \( (\phi \rightarrow \psi)^M = (M-\phi^M + \psi^M) \)
  - \( (EX\phi)^M = \{w \mid \exists w' (wRw' \land w' \in \phi^M )\} \)
  - \( E(\phi U^+ \psi)^M = \{w \mid \text{there is a path } \alpha \text{ from } w \text{ and a } w' \text{ on } \alpha \text{ such that } (w<w' \land w' \in \psi^M ) \land \forall w'' (w<w''<w' \rightarrow w'' \in \phi^M )\} \)
  - \( A(\phi U^+ \psi)^M = \{w \mid \text{for all paths } \alpha \text{ from } w \text{ there is a } w' \text{ on } \alpha \text{ such that } (w<w' \land w' \in \psi^M ) \land \forall w'' (w<w''<w' \rightarrow w'' \in \phi^M )\} \)
  - \( (AX\phi)^M = \{w \mid \exists w' (wRw' \land \forall w' (wRw' \rightarrow w' \in \phi^M ))\} \)
Actual Calculation

• How to calculate \((\text{EX} \ \psi)^M\) from \(\psi^M\)?
  ▪ Inverse image construction
  ▪ How to calculate \((\text{AX} \ \psi)^M\) from \(\psi^M\)?

• How to calculate \(E(\varphi U^+ \psi)^M\) or \(A(\varphi U^+ \psi)^M\) from \(\varphi^M\) and \(\psi^M\)?

\[
E(\psi_2 U^+ \psi_1) \leftrightarrow \text{EX}(\psi_1 \lor \psi_2 \land E(\psi_2 U^+ \psi_1))
\]
\[
A(\psi_2 U^+ \psi_1) \leftrightarrow \text{AX}(\psi_1 \lor \psi_2 \land A(\psi_2 U^+ \psi_1))
\]
Inverse reachability calculation

```
function reach (Pointset Target): Pointset =
    Source := {}; Search := Target;
while Search ≠ {} do
    Search := pred (Search) \ Source;
    Source := Source \cup Search
endo;
return Source;
function pred (Point w): Pointset = return \{w' | (w', w) ∈ I(≺)\};
```
procedure CTL_check (Model \((U, I, w_0)\), Formula \(\varphi\)) =
  if \(w_0 \in \text{eval}(\varphi)\)
  then print(“\(\varphi\) is satisfied at \(w_0\) in \((U, I)\)”)
  else print(“\(\varphi\) not satisfied at \(w_0\) in \((U, I)\)”);

function eval (Formula \(\varphi\)): Pointset =
  case \(\varphi\) of
    \(p\) : return \(I(p)\);
    \(\bot\) : return \{}\;
    (\(\psi_1 \rightarrow \psi_2\)) : return \(U \setminus \text{eval}(\psi_1) \cup \text{eval}(\psi_2)\);
    \(E(\psi_2 U^+ \psi_1)\) : \(E_1 := \text{eval}(\psi_1); \ E_2 := \text{eval}(\psi_2); \ E := {}\); repeat until stabilization
      \(E := E \cup \{w | (\text{succ}(w) \cap (E_1 \cup (E_2 \cap E))) \neq {}\}\); return \(E\);
    \(A(\psi_2 U^+ \psi_1)\) : \(E_1 := \text{eval}(\psi_1); \ E_2 := \text{eval}(\psi_2); \ E := {}\); repeat until stabilization
      \(E := E \cup \{w | \{\} \neq \text{succ}(w) \subseteq E_1 \cup (E_2 \cap E)\}\); return \(E\);
  function succ (Point \(w\)): Pointset = return \(\{w' | (w, w') \in I(\prec)\}\);
Symbolic Representation

- Model checking algorithm deals with sets of states and with relations (sets of pairs of states)
- Need an efficient representation
Binary Encoding of Domains

- Any variable on a finite domain D can be replaced by $\log(D)$ binary variables
  - Similar to encoding of data types by compilers
  - E.g. var v: {0..15} can be replaced by var v1,v2,v3,v4: boolean
    (0=0000, 1=0001, 2=0010, 3=0011, ..., 15=1111)

- State space
  - Still in the order of original domain!
  - E.g. three int8-variables can have $2^{24}=10^8$ states
  - E.g. array of length 10 with 10-bit values $\rightarrow 10^{30}$ states

- Representation of large sets of states?
 Representation of Sets

Example: Consider the domain $D \triangleq \{0..15\}$

- symbolic: $S \triangleq \{x \mid x \mod 2 = 0 \lor x > 12\}$
- enumeration: $S = \{0, 2, 4, 6, 8, 10, 12, 13, 14, 15\}$
- bitstring: $S = (1010101010101111)$
- binary: $S = \{0000, 0010, 0100, 0110, 1000, 1010, 1100, 1101, 1110, 1111\}$
- propositional formula: $S = \{\vec{v} \mid v_4 = 0 \lor v_1 = 1 \land v_2 = 1\}$
- logical spectrum: $S = \text{lte}(v_1, \text{lte}(v_2, \top, \text{lte}(v_4, \perp, \top))), \text{lte}(v_4, \perp, \top))$
Ordered Tree Form

- Normal form for propositional formulas
- Uses only the connective $\text{Ite}$
  \[
  \text{Ite}(\varphi, \psi_1, \psi_2) \triangleq ((\varphi \rightarrow \psi_1) \land (\neg \varphi \rightarrow \psi_2))
  \]
- Linear ordering on the set of propositions
  - e.g., most significant bit first
- Shannon expansion
  \[
  \varphi \leftrightarrow \text{Ite}(v, \varphi\{v := \top\}, \varphi\{v := \bot\})
  \]
Truth table and tree form formula

Reduction: Replace $\text{Ite}(v, \psi, \psi)$ by $\psi$
Abbreviations

\[ S = \text{lte}(v_1, \text{lte}(v_2, \top, \text{lte}(v_4, \bot, \top)), \text{lte}(v_4, \bot, \top)) \]

- Introduce abbreviations
  \[ S' = \text{lte}(v_1, \text{lte}(v_2, \top, \delta), \delta), \text{ where } \delta \triangleq \text{lte}(v_4, \bot, \top) \]

- maximally abbreviated
  \[ S = \text{lte}(v_1, \delta_1, \delta_2), \text{ where } \delta_1 \triangleq \text{lte}(v_2, \top, \delta_2) \text{ and } \delta_2 \triangleq \text{lte}(v_4, \bot, \top) \]
Binary Decision Trees (BDTs)

- Binary decision tree

- Elimination of isomorphic subtrees (abbreviations)
Binary Decision Diagrams (BDDs)

- Elimination of redundant nodes (redundant subformulas) \( \text{Ite} (v, \psi, \psi) \) by \( \psi \)
Calculation of BDDs

function PL2BDD (Formula $\varphi$) : (Nodeset, Int)
    /* Calculates the BDD of $\varphi$
        as a set of nodes and a pointer to the topmost node */
    Nodeset table := \{\}; /* Table of BDD nodes ($\delta, i, \delta_1, \delta_2$) */
    Int max := 1; /* Index of maximal table entry */
    Int result := BDD($\varphi,1$); /* Index of topmost BDD node */
    return (table, result);

function BDD (Formula $\varphi$, Int $i$) : Int
    /* $\varphi$ is the current subformula, $i$ is the current BDD variable */
    /* Return value is a pointer to the maximal BDD node */
    if $i > n$ then return eval($\varphi$) /* $\varphi$ is a boolean constant */
    else $\delta_1 := BDD(\varphi\{v_i := \bot\}, i + 1)$; $\delta_2 := BDD(\varphi\{v_i := \top\}, i + 1)$;
        if $\delta_1 = \delta_2$ then return $\delta_1$
        elsif $\exists \delta : (\delta, i, \delta_1, \delta_2) \in$ table then return $\delta$
        else max := max + 1; table := table $\cup \{(max, i, \delta_1, \delta_2)\}$; return max;
Operations on BDDs

- Negation: easy (exchange T and F)
- Falsum: trivial
- and, or: Shannon expansion
  - \((\varphi \text{ OP } \psi) = x \land (\varphi\{x:=T\} \text{ OP } \psi\{x:=T\})\)
    \[\lor \neg x \land (\varphi\{x:=\bot\} \text{ OP } \psi\{x:=\bot\})\]
  - \((\varphi \land \psi) = (x \land (\varphi\{x:=T\} \land \psi\{x:=T\})) \lor (\neg x \land (\varphi\{x:=\bot\} \land \psi\{x:=\bot\}))\)
- BDD realization?
BDD-implies

```c
function BDD_imp (Bdd \( \varphi \), \( \psi \)) : Bdd =
    /* Calculates the BDD of \((\varphi \rightarrow \psi)\) from the BDDs of \(\varphi\) and \(\psi\) */
    if \( \varphi = 0 \) or \( \psi = 1 \) then return 1
    elsif \( \varphi = 1 \) then return \( \psi \)
    elsif \( \psi = 0 \) and (\( \varphi, i, \varphi_1, \varphi_2 \)) \( \in \) table\( \varphi \)
        then return new_node(\( i, \) BDD_imp(\( \varphi_1, 0 \)), BDD_imp(\( \varphi_2, 0 \))
    else (\( \varphi, i, \varphi_1, \varphi_2 \)) \( \in \) table\( \varphi \) and (\( \psi, j, \psi_1, \psi_2 \)) \( \in \) table\( \psi \)
        if \( i = j \) then return new_node(\( i, \) BDD_imp(\( \varphi_1, \psi_1 \)), BDD_imp(\( \varphi_2, \psi_2 \))
        elsif \( i < j \) then return new_node(\( i, \) BDD_imp(\( \varphi_1, \psi \)), BDD_imp(\( \varphi_2, \psi \))
        elsif \( i > j \) then return new_node(\( j, \) BDD_imp(\( \varphi, \psi_1 \)), BDD_imp(\( \varphi, \psi_2 \));

function new_node (Bddvar \( i \), Bdd \( \delta_1, \delta_2 \)) : Bdd =
    /* Returns a pointer to a new or existing BDD node */
    /* \( i \) is the number of a BDD variable, \( \delta_1, \delta_2 \) pointers to BDD nodes */
    if \( \delta_1 = \delta_2 \) then return \( \delta_1 \)
    elsif \( \exists \delta : (\delta, i, \delta_1, \delta_2) \in \) table then return \( \delta \)
    else max := max + 1; table := table \( \cup \) \{(max, i, \delta_1, \delta_2)\}; return max;
```
The Influence of Variable Ordering

\[ v_1 = 0 \rightarrow ((v'_1 = 1) \land (v'_2 = v_2) \land (v'_3 \neq v_3)) \]

- Heuristics: keep dependent variables close together!