Lecture 3: Satisfiability

Huan Zhang
huan@huan-zhang.com
Readings

• Chapter 7
• Appendix C

Outline
• Review on the proofs of inductive invariance properties
• Propositional Satisfiability problem
• Normal forms
• DPLL algorithm (next lecture)
Dijkstra’s mutual exclusion Algorithm [‘74]

N processes: 0, 1, ..., N-1
state of each process j is a single integer variable \( x[j] \in \{0, 1, 2, K-1\} \), where \( K > N \)
The “update” action is defined differently for P0 vs. others

\( P_0 \)
if \( x[0] = x[N-1] \) then \( x[0] := x[0] + 1 \) mod \( K \)

\( P_j, j > 0 \)
if \( x[j] \neq x[j -1] \) then \( x[j] := x[j-1] \)

\( p_i \) has TOKEN if and only if the blue conditional is true
Dijkstra’s mutual exclusion Algorithm ['74]

N processes: 0, 1, ..., N-1
state of each process j is a single integer variable x[j] ∈ \{0, 1, 2, K-1\}, where K > N
The “update” action is defined differently for P0 vs. others

\( P_0 \) if \( x[0] = x[N-1] \) then \( x[0] := x[0] + 1 \mod K \)

\( P_j, j > 0 \) if \( x[j] \neq x[j -1] \) then \( x[j] := x[j-1] \)

\( p_i \) has TOKEN if and only if the blue conditional is true
A language for specifying automata (IOA)

automaton DijkstraTR(N:Nat, K:Nat), where K > N

type ID: enumeration [0,...,N-1]
type Val: enumeration [0,...,K-1]

actions
  update(i:ID)

variables
  x:[ID -> Val] initially for all i:ID x[i] = 0

transitions
  update(i:ID)
    pre i = 0 \& x[i] = x[N-1]
    eff x[i] := (x[i] + 1) % K

  update(i:ID)
    pre i > 0 \& x[i] \neq x[i-1]
    eff x[i] := x[i-1]

Automaton \( \mathcal{A} = \langle X, \Theta, A, D \rangle \)
Reachable states and invariants

A state $u$ is **reachable** if there exists an execution $\alpha$ such that $\alpha.lstate = u$

$\text{Reach}_A(\Theta)$: set of states reachable from $\Theta$ by automaton $A$

An **invariant** is a set of states $I$ such that $\text{Reach}_A \subseteq I$
Theorem 7.1. Given a automaton \( \mathcal{A} = \langle X, \Theta, A, D \rangle \) and a set of states \( I \subseteq \text{val}(X) \) if:

- (Start condition) for any \( x \in \Theta \) implies \( x \in I \), and
- (Transition closure) for any \( x \rightarrow_a x' \) and \( x \in I \) implies \( x' \in I \)

then \( I \) is an (inductive) invariant of \( \mathcal{A} \). That is \( \text{Reach}_\mathcal{A}(\Theta) \subseteq I \).
Proving invariants by induction for Dijkstra

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, D \rangle$ and a set of states $I \subseteq \text{val}(X)$ if:

• (Start condition) for any $x \in \Theta$ implies $x \in I$, and

• (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then $I$ is an (inductive) invariant of $\mathcal{A}$. That is $\text{Reach}_\mathcal{A}(\Theta) \subseteq I$.

$I_1$: “Exactly one process has the token”.

(Start condition): Fix a $x \in \Theta$. $x \models \forall i \ x[i] = 0$ therefore $x \models I_1$

(Transition closure): Fix a $x \rightarrow_a x'$ such that $x \in I$.

Two cases to consider.

1. If $a = \text{update}(0)$ then
   a) since $x \models \text{Pre}(\text{update}(0))$ it follows that $x[0] = x[N - 1]$
   b) since $x \models I_1$ it follows that $\forall i > 0 \ x[i] = x[i - 1]$
   c) $x'[0] \neq x'[N - 1]$ by applying (a) and $\text{Eff}(\text{update}(0))$ to $x$
   d) $x'[1] \neq x[0]$ by applying (b) and $\text{Eff}(\text{update}(0))$ to $x$
   e) $\forall i > 1 \ x'[i] = x'[i - 1]$ by applying (b) and $\text{Eff}(\text{update}(0))$ to $x$

Therefore $x' \models I_1$.

2. If $a = \text{update}(i)$, $i > 0$ then fix arbitrary $i > 0$ ... (do it as an exercise)

From above **Theorem** it follows that $I_1$ is an invariant of DijkstraTR
Proving invariants by induction for Dijkstra

Theorem 7.1. Given an automaton \( \mathcal{A} = \langle X, \Theta, A, D \rangle \) and a set of states \( I \subseteq \text{val}(X) \) if:

- (Start condition) for any \( x \in \Theta \) implies \( x \in I \), and
- (Transition closure) for any \( x \rightarrow_a x' \) and \( x \in I \) implies \( x' \in I \)

then \( I \) is an (inductive) invariant of \( \mathcal{A} \). That is \( \text{Reach}_\mathcal{A}(\Theta) \subseteq I \).

- \( I_1 \): “Exactly one process has the token”.

(Start condition): Fix a \( x \in \Theta \). \( x \models \forall i \ x[i] = 0 \) therefore \( x \models I_1 \)

(Transition closure): Fix a \( x \rightarrow_a x' \) such that \( x \in I \).

Two cases to consider.

1. If \( a = \text{update}(0) \) then
   a) since \( x \models \text{Pre}(\text{update}(0)) \) it follows that \( x[0] = x[N-1] \)
   b) since \( x \models I_1 \) it follows that \( \forall i > 0 \ x[i] = x[i-1] \)
   c) \( x'[0] \neq x'[N-1] \) by applying (a) and \( \text{Eff}(\text{update}(0)) \) to \( x \)
   d) \( x'[0] \neq x[0] \) by applying (b) and \( \text{Eff}(\text{update}(0)) \) to \( x \)
   e) \( \forall i > 1 \ x'[i] = x'[i-1] \) by applying (b) and \( \text{Eff}(\text{update}(0)) \) to \( x \)

   Therefore \( x' \models I \).

2. If \( a = \text{update}(i), i > 0 \) then fix arbitrary \( i > 0 \) ... (do it as an exercise)

From above **Theorem** it follows that \( I_1 \) is an invariant of \( \text{DijkstraTR} \)

```
automan DijkstraTR(N: Nat, K: Nat), where K > N
type ID: enumeration [0,...,N-1]
type Val: enumeration [0,...,K-1]
actions
  update(i:ID)
variables
  x:[ID -> Val] initially forall i:ID x[i] = 0
transitions
  update(i:ID)
  pre i = 0 /\ x[i] = x[(N-1)]
  eff x[i] := (x[i] + 1) % K
update(i:ID)
  pre i > 0 /\ x[i] ~= x[i-1]
  eff x[i] := x[i-1]
```

Can we prove this part automatically? Yes! Use a *satisfiability* solver! (HW1)
Boolean *satisfiability* problem

Given a *well-formed formula* in propositional logic, determine whether there exists a satisfying solution

Example: \( \alpha(x_1, x_2, ..., x_n) \equiv (x_1 \land x_2 \lor x_3) \land (x_1 \land \neg x_3 \lor x_2) \)

Set of variables: \( X = \{x_1, x_2, ..., x_n\} \)

Each variable is Boolean: \( \text{type}(x_i) = \{0, 1\} \)

Formula \( \alpha \) is *well-formed* if it uses propositional operators, and \( \land \), \( \lor \), not \( \neg \), iff \( \leftrightarrow \) etc., properly

Recall, a valuation \( x \) of \( X \) maps each \( x_i \) to a value 0 or 1

A valuation \( x \) of \( X \) *satisfies* \( \alpha \) is each each \( x_i \) in \( \alpha \) replaced by the corresponding value in \( x \) evaluates to *true*. We write this as \( x \models \alpha \)

Otherwise, we write \( x \not\models \alpha \)

Example: with \( x \equiv \langle x_1 \leftrightarrow 1, x_2 \leftrightarrow 1, x_3 \leftrightarrow 0 \rangle \); \( x \models \alpha \)
Boolean *satisfiability* problem (SAT)

Given a well-formed formula in propositional logic, determine whether there exists a satisfying solution.

Restatement: \( \exists x \in val(X): x \vDash \alpha \)?

If the answer is ”No” then \( \alpha \) is said to be *unsatisfiable*

**Aside.** If \( \forall x \in val(X): x \vDash \alpha \) then \( \alpha \) is said to be *valid* or *a tautology*

If \( \alpha \) is valid then \( \neg \alpha \) is unsatisfiable.

\( \alpha \) and \( \alpha' \) are *tautologically equivalent* if they have the same truth tables

\[
\forall x \in val(X): x \vDash \alpha \iff x \vDash \alpha'
\]

What is a naïve method for solving SAT?

What is the complexity of this approach? How many evaluations of \( \alpha(x_1, x_2, ..., x_n) \)?
I Don’t Get No *Satisfaction*, but I try, try, try,...

I can prove why.

Prof. Cook

SAT is NP-complete

SAT was the first problem shown to be NP-complete [Cook 71]

2-SAT can be solved in polynomial time (Exercise)

(Read definition of NP: Nondeterministic Polytime in Appendix C)

This has real implications

1. Essentially we don’t know better than the naïve algorithm

2. A solver for SAT can be used to solve any other problem in the NP class with only polytime slowdown. i.e., makes a lot of sense to build SAT solvers

3. SAT/SMT solving is the cornerstone of many verification procedures

Details

We will assume $\alpha$ to be in conjunctive normal form (CNF)

**literals:** variable or its negation, e.g., $x_3, \neg x_3$

**clause:** disjunction (or) of literals, e.g., $(x_1 \lor x_2 \lor \neg x_3)$

**CNF formula:** conjunction (and) of clauses,

\[ (x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor x_1) \]

A variable may appear *positively* or *negatively* in a clause
Logic and circuits

Repeated subexpression is inefficient
Solution: rename \((A \land B) \leftrightarrow E\)
\[ I' \equiv (D \land E) \lor (\neg C \land E) \land ((A \land B) \leftrightarrow E) \]

\(I\) and \(I'\) are not tautologically equivalent

\(C = 0, \ A = B = 1, \ E = 0\) satisfies \(I\)

But they are equisatisfiable, i.e., \(I\) is satisfiable iff \(I'\) is also satisfiable

Recall that:
\[
\begin{align*}
A & \leftrightarrow B \\
(A \rightarrow B) \land (B \rightarrow A) \\
(\neg A \lor B) \land (\neg B \lor A)
\end{align*}
\]
Converting to CNF

• View the formula as a graph
• Give new names (variables) to non-leafs
• Relate the inputs and the outputs of the nonleafs and add this as a new clause
• Take conjunction of all of this
Converting to CNF

- \( F \leftrightarrow \neg C \)
  - \( F \rightarrow \neg C \land \neg C \rightarrow F \)
  - \( (\neg F \lor \neg C) \land (C \lor F) \)

- \( (A \land B) \leftrightarrow E \)
  - \( ((A \land B) \rightarrow E) \land (E \rightarrow (A \land B)) \)
  - \( (\neg (A \land B) \lor E) \land (\neg E \lor (A \land B)) \)
  - \( (\neg A \lor \neg B \lor E) \land (\neg E \lor A) \land (\neg E \lor B) \)

- \( (G \lor H) \leftrightarrow I \)
  - \( ((G \lor H) \rightarrow I) \land (I \rightarrow (G \lor H)) \)
  - \( (\neg G \land \neg H \lor I) \land (\neg I \lor G \lor H) \)
  - \( (\neg G \lor I) \land (\neg H \lor I) \land (\neg I \lor G \lor H) \)

- \( (D \land E) \leftrightarrow G \)
  - \( (\neg D \lor \neg E \lor G) \land (\neg G \lor D) \land (\neg G \lor E) \)

- \( (F \land E) \leftrightarrow H \)
  - \( (\neg F \lor \neg E \lor H) \land (\neg H \lor F) \land (\neg H \lor E) \)
Standard representations of CNF

- \((\neg A \lor \neg B \lor E) \land (\neg E \lor A) \land (\neg E \lor B)\)
- \((A' + B' + E)(E' + A)(E' + B)\)
- \((-1 - 2 5)(-5 1)(-5 2)\) DIMACS

- SMTLib: computer readable, standard format

[https://smtlib.cs.uiowa.edu/language.shtml](https://smtlib.cs.uiowa.edu/language.shtml)
Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Transform the given formula $\alpha$ by applying a sequence of satisfiability preserving rules

If final result has an empty clause then unsatisfiable
if final result has no clauses then the formula is satisfiable
Davis Putnam Algorithm (DP) 1960

Rule 1. **Unit propagation**
Rule 2. **Pure literal**
Rule 3. **Resolution**
Rule 1. **Unit propagation**
A clause has a single literal

\[ \alpha \equiv \ldots \land p \land \ldots \land \ldots \]

What choice do we really have?

\[ \alpha \equiv \ldots \land (x_1 \lor \lnot p \lor x_2) \land p \land \ldots \land (\lnot x_3 \lor \lnot p \lor x_1) \ldots \]
Rule 1. *Unit propagation*

A clause has a single literal

\[ \alpha \equiv \ldots \land \ldots \land p \land \ldots \land \ldots \]

What choice do we really have?

\[ \alpha' \equiv \ldots \land (x_1 \lor x_2) \land \ldots \land (\neg x_3 \lor x_1) \ldots \]

\(\alpha\) and \(\alpha'\) are equisatisfiable
Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in $\alpha$

$$\alpha \equiv \ldots \land (x_1 \lor \neg p \lor x_2) \land (x_4 \lor \neg p) \land \ldots \land (\neg x_3 \lor \neg p \lor x_1) \ldots$$

$p$ does not appear anywhere

Makes sense to set $p = 0$ and remove all occurrences of $\neg p$
Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in \( \alpha \)

\[
\alpha \equiv \ldots \land (x_1 \lor \neg p \lor x_2) \land (x_4 \lor \neg p) \land \ldots \land (\neg x_3 \lor x_1) \ldots
\]

\( p \) does not appear anywhere

Makes sense to set \( p = 0 \) and remove all clauses in which \( \neg p \) occurs

\( \alpha \) and \( \alpha' \) are equisatisfiable

\[
\alpha' \equiv \ldots \land \ldots \land (\neg x_3 \lor x_1) \ldots \ [p = 0]
\]
Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3. **Resolution**

Choose a literal $p$ that appears with both polarity in $\alpha$. Suppose $(\ell_1 \lor \ell_2 \lor p)$ be a clause in which $p$ appears positively, and $(k_1 \lor k_2 \lor \neg p)$ be a clause in which $p$ appears negatively.

Then the resolved clause is $(\ell_1 \lor \ell_2 \lor k_1 \lor k_2)$.

Pairwise, resolve each clause in which $p$ appears positively with a clause in which $p$ appears negatively, and take the conjunction of all the results.

Why is the result equisatisfiable?

What is the size of the resulting formula?
DPLL modifies resolution in DP with recursive DFS rule

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3’. Let $\Delta$ be the current set of clauses. Choose a literal $p$ in $\Delta$.

Check satisfiability of $\Delta \cup \{ p \}$ (guessing $p = 1$)

If satisfiable then return True else

return result of checking satisfiability of $\Delta \cup \{ \neg p \}$

This is essentially a depth first search
Assignments

• HW1 (due Feb 11\textsuperscript{th})
  • Install Z3

• Keep thinking about class projects! Form teams (max 2 people).

• More on DPLL next lecture