Satisfiability over a finite-state program

\( P \)-validity problem (of \( \varphi \))

Given a finite-state program \( P \) and formula \( \varphi \),

is \( \varphi \) \( P \)-valid?
i.e. do all \( P \)-computations satisfy \( \varphi \)?

\( P \)-satisfiability problem (of \( \varphi \))

Given a finite-state program \( P \) and formula \( \varphi \)

is \( \varphi \) \( P \)-satisfiable?
i.e., does there exist a \( P \)-computation which satisfies \( \varphi \)?

To determine whether \( \varphi \) is \( P \)-valid, it suffices to apply an algorithm for deciding if there is a \( P \)-computation that satisfies \( \neg \varphi \).
The Idea

To check $P$-satisfiability of $\varphi$, we combine the tableau $T_\varphi$ and the transition graph $G_P$ into one product graph, called the behavior graph $B(\{P,\varphi\})$, and search for paths

$$(s_0, A_0), (s_1, A_1), (s_2, A_2), \ldots$$

that satisfy the two requirements:

- $\sigma \models \varphi$: there exists a fulfilling path $\pi: A_0, A_1, \ldots$ in the tableau $T_\varphi$ such that $\varphi \in A_0$.

- $\sigma$ is a $P$-computation: there exists a fair path $\sigma: s_0, s_1, \ldots$ in the transition graph $G_P$. 
State transition graph $G_P$: Construction

- Place as nodes in $G_P$ all initial states $s$ ($s \models \Theta$)

- Repeat

  for some $s \in G_P$, $\tau \in T$,  
  add all its $\tau$-successors $s'$ to $G_P$  
  if not already there,  
  and add edges between $s$ and $s'$.

  Until no new states or edges can be added.

If this procedure terminates, the system is finite-state.
Example: Program mux-pet1 (Fig. 3.4)
(Peterson’s Algorithm for mutual exclusion)

local \( y_1, y_2 \): boolean where \( y_1 = F, y_2 = F \)
s : integer where \( s = 1 \)

\[ P_1 :: \]
\[ \ell_0 : \text{loop forever do} \]
\[ \begin{align*}
\ell_1 : & \text{noncritical} \\
\ell_2 : & (y_1, s) := (T, 1) \\
\ell_3 : & \text{await } (\neg y_2) \lor (s \neq 1) \\
\ell_4 : & \text{critical} \\
\ell_5 : & y_1 := F
\end{align*} \]

\[ m_0 : \text{loop forever do} \]
\[ \begin{align*}
m_1 : & \text{noncritical} \\
m_2 : & (y_2, s) := (T, 2) \\
m_3 : & \text{await } (\neg y_1) \lor (s \neq 2) \\
m_4 : & \text{critical} \\
m_5 : & y_2 := F
\end{align*} \]
Abstract state-transition graph for MUX-PET1

We use $y_1 \Leftrightarrow at_{-\ell_{3..5}}$

$y_2 \Leftrightarrow at_{-m_{3..5}}$
Some states have been lumped together:
a super-state labeled by $i$ represents $i$ states

MUX-PETL has 42 reachable states.

Based on this graph it is straightforward to check the properties

$$\psi_1 : \Box \neg (at_{-l_4} \land at_{-m_4})$$

$$\psi_2 : \Box (at_{-l_3} \land \neg at_{-m_3} \rightarrow s = 1)$$

$$\psi_3 : \Box (at_{-m_3} \land \neg at_{-l_3} \rightarrow s = 2)$$
MUX-PET1 Full state-transition graph $(l_i, m_j, s)$
Definitions

- For atom $A$, $\text{state}(A)$ is the conjunction of all state formulas in $A$
  (by $R_{\text{sat}}$, $\text{state}(A)$ must be satisfiable)

- For $A \in T_\varphi$,
  $\delta(A)$ denotes the set of successors of $A$
  in $T_\varphi$

- atom $A$ is consistent with state $s$
  if $s \models \text{state}(A)$,
  i.e. $s$ satisfies all state formulas in $A$.

- $\vartheta$: $A_0, A_1, \ldots$ path in $T_\varphi$
  $\sigma$: $s_0, s_1, \ldots$ computation of $P$

  $\vartheta$ is a trail of $T_\varphi$ over $\sigma$ if
  $A_j$ is consistent with $s_j$, for all $j \geq 0$
For finite-state program $P$ and formula $\varphi$, we construct the $(P, \varphi)$-behavior graph

$$\mathcal{B}_{(P, \varphi)} \approx G_P \times T_{\varphi}^- \text{ (pruned)}$$

such that

- **nodes** are labeled by $(s, A)$
  where $s$ is a state from $G_P$ and $A$ is an atom from $T_{\varphi}$ consistent with $s$.

- **edges**
  There is an edge $s, A \xrightarrow{\tau} s', A'$ if and only if $s' \in \tau(s)$ and $A' \in \delta(A)$

  $$
  \begin{array}{c}
  \text{in } G_P \\
  \text{in } T_{\varphi}
  \end{array}
  $$

- **initial $\varphi$-node** $(s, A)$
  if $s$ is an initial state ($s \models \Theta$) and $A$ is an initial $\varphi$-atom ($\varphi \in A$)

  It is marked $\text{marked}(s, A)$
Algorithm behavior-graph
(constructing $B_{(P,\varphi)}$)

- Place in $B$ all initial $\varphi$-nodes $(s, A)$
  ($s$ initial state of $P$,
  $A$ initial $\varphi$-atom in $T_{\varphi}^-$
  $A$ consistent with $s$)

- Repeat until no new nodes or new edges can be added:
  Let $(s, A)$ be a node in $B$
  $\tau \in T$ a transition
  $(s', A')$ a pair s.t.
  $s'$ is a $\tau$-successor of $s$
  $A' \in \delta(A)$ in pruned $T_{\varphi}^-$
  $A'$ consistent with $s'$

  - Add $(s', A')$ to $B$, if not already there
  - Draw a $\tau$-edge from $(s, A)$ to $(s', A')$, if not already there
Example: Given FTS LOOP

\[ \Theta : x = 0 \]
\[ \mathcal{T} = \{ \tau, \tau_I \} \]
with \( \tau_I \) (idling)
\[ \tau \text{ where } \rho_\tau : x' = (x + 1) \text{mod} 4 \]
\[ \mathcal{J} : \{ \tau \} \]

Check \( P \)-satisfiability of \( \psi_3 : \Diamond \Box (x \neq 3) \)

state-transition graph \( G_{\text{LOOP}} \) (Fig 5.9)
pruned \( T_{\psi_3}^- \) (Fig 5.8)
Behavior graph \( B_{(\text{LOOP}, \psi_3)} \) (Fig 5.10)
Fig. 5.9. State-transition graph $G_{\text{LOOP}}$
Pruned tableau $T_{\psi_3}^{-}$ (Fig. 5.8)

Eliminating

- MSCS’s not reachable from an initial $\psi_3$-atom and
- non-fulfilling terminal MSCS’s

Promising formulas:

\[ \Diamond \Box (x \neq 3) \text{ promising } \Box (x \neq 3) \]
\[ \neg \Box (x \neq 3) \text{ promising } (x = 3) \]

Two non-transient MSCS’s:

\[ \{ A_4^{-+}, A_5^{-+} \} \text{ not fulfilling} \]
\[ \{ A_7^{++} \} \text{ fulfilling} \]
Behavior graph $\mathcal{B}_{(\text{LOOP}, \psi_3)}$ (Fig 5.10)
Example: Given FTS ONE:

\[ \Theta: \quad x = 0 \]

\[ \mathcal{T}: \quad \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_I\} \]

with \[ \rho_{\tau_1}: \quad x = 0 \land x' = 1 \]

\[ \rho_{\tau_2}: \quad x = 1 \land x' = 0 \]

\[ \rho_{\tau_3}: \quad x = 0 \land x' = -1 \]

\[ \rho_{\tau_4}: \quad x = -1 \land x' = 0 \]

\[ J: \quad \emptyset \]

\[ C: \quad \{\tau_1, \tau_3\} \]

Transition graph \( G_{\text{ONE}} \)
We want to know whether
\[ \varphi : \Box \Diamond (x = 1) \]
is valid over ONE.

Check \( P \)-satisfiability of
\[ \neg \varphi : \underbrace{\Diamond \Box (x \neq 1)}_{\psi} \]

\( \Phi^+_\psi : \{ \psi, \lozenge \psi, \Box (x \neq 1), \lozenge \Box (x \neq 1), x = 1 \} \)

basic formulas: \{ \lozenge \psi, \lozenge \Box (x \neq 1), x = 1 \}

Promising formulas:
\[ \psi_1 : \psi = \Diamond \Box (x \neq 1) \text{ promising} \quad r_1 : \Box (x \neq 1) \]
\[ \psi_2 : \neg \Box (x \neq 1) \text{ promising} \quad r_2 : x = 1 \]
Pruned tableau $T_{\psi}$

ψ, ¬□(x ≠ 1), ○ψ, ¬□(x ≠ 1)

$A_{4}^{-+}: x = 1$  $A_{5}^{--}: x ≠ 1$

$A_{6}^{-+}: x = 1, ○□(x ≠ 1), ○ψ, □(x ≠ 1), ψ$

$A_{7}^{++}: x ≠ 1, ○□(x ≠ 1), ○ψ, □(x ≠ 1), ψ$
Behavior graph $\mathcal{B}_{\text{ONE}, \Diamond \Box (x \neq 1)}$

Two non-transient MSCS’s:

\{$(s_2, A_4^{-+})$, $(s_1, A_5^{--})$, $(s_3, A_5^{--})$\}: not fulfilling,

\{$(s_1, A_7^{++})$, $(s_3, A_7^{++})$\}: fulfilling
Claim 5.9 (paths of $\mathcal{B}_{(P,\varphi)}$)

The infinite sequence

$$\pi: \left( (s_0, A_0), (s_1, A_1), \ldots \right)$$

is a path in $\mathcal{B}_{(P,\varphi)}$

iff

$$\sigma_\pi: s_0, s_1, \ldots \text{ is a run of } P$$

(i.e. computation of $P$ less fairness)

$$\vartheta_\pi: A_0, A_1, \ldots \text{ is a trail of } T_\varphi \text{ over } \sigma_\pi$$

(i.e. $A_j$ consistent with $s_j$, for all $j \geq 0$)

Example: In $\mathcal{B}_{(\text{LOOP},\psi_3)}$ (Fig. 5.10)

$$\pi: \left( (s_0, A_5), (s_1, A_5), (s_2, A_5), (s_3, A_4) \right)^\omega$$

induces

$$\sigma_\pi: (s_0, s_1, s_2, s_3)^\omega \text{ run of LOOP}$$

$$\vartheta_\pi: (A_5, A_5, A_5, A_4)^\omega \text{ trail of } T_{\psi_3} \text{ over } \sigma_\pi$$
Proposition 5.10 (*P*-satisfiability by path)

*P* has a computation satisfying *ϕ*

iff

there is an infinite \( \varphi \)-initialized path \( \pi \)
in \( \mathcal{B}_{(P,\varphi)} \) s.t.

\[ \sigma_\pi \] is a *P*-computation (fair run of *P*)

\[ \vartheta \] is a fulfilling trail over \( \sigma_\pi \)

Searching for “good” paths in \( \mathcal{B}_{(P,\varphi)} \)

— not practical.
Definitions

For behavior graph $\mathcal{B}_{(P,\varphi)}$

- node $(s', A')$ is a $\tau$-successor of $(s, A)$ if $\mathcal{B}_{(P,\varphi)}$ contains $\tau$-edge connecting $(s, A)$ to $(s', A')$

- transition $\tau$ is enabled on node $(s, A)$ if $\tau$ is enabled on state $s$
Definitions (Con’t)

For scs $S \subseteq \mathcal{B}_{(P, \varphi)}$:

- Transition $\tau$ is taken in $S$ if there exists two nodes $(s, A), (s', A') \in S$ s.t. $(s', A')$ is a $\tau$-successor of $(s, A)$

- $S$ is \underbrace{\text{just compassionate}}_{\text{compassionate}}\) if every \{just compassionate\} transition $\tau \underbrace{\in \mathcal{J}}_{\in \mathcal{C}}$ is either taken in $S$ or is disabled on \underbrace{\text{some node}}_{\text{all nodes}} in $S$

- $S$ is \underline{fair} if it is both just and compassionate

- $S$ is \underline{fulfilling} if every promising formula $\psi \in \Phi_\psi$ is fulfilled by some atom $A$, s.t. $(s, A) \in S$ for some state $s$

- $S$ is \underline{adequate} if it is fair and fulfilling
Adequate SCS’s

**Proposition 5.11** (adequate SCS and satisfiability)

Given a finite-state program $P$ and temporal formula $\varphi$. $\varphi$ is $P$-satisfiable

iff

$B_{(P,\varphi)}$ has an adequate SCS

**Example:** Consider LOOP and

\[
\psi_3: \Diamond \Box (x \neq 3)
\]

Is $\psi_3$ LOOP-satisfiable?

Check the SCS’s in $B_{(\text{LOOP},\psi_3)}$ (Fig. 5.10)
Behavior graph $\mathcal{B}_{(\text{LOOP}, \psi_3)}$ (Fig 5.10)
Example (Con’t)

- \{ (s_0, A_{5^-}), (s_1, A_{5^-}), (s_2, A_{5^-}), (s_3, A_{4^+}) \} is fair but not fulfilling

- \{ (s_0, A_{7^+}), (s_1, A_{7^+}), (s_2, A_{7^+}) \}
  each is fulfilling but not fair
  Not just with respect to transition \( \tau \)

- \{ (s_3, A_{6^-}) \}
  is neither fair (unjust toward \( \tau \)) nor fulfilling (being transient)

No adequate subgraphs in \( \mathcal{B}_{(\text{LOOP}, \psi_3)} \)

Therefore, by proposition 5.11, \text{LOOP} has no computation that satisfies \( \psi_3: \Diamond \Box (x \neq 3) \)
Example: Consider LOOP and

\[ \varphi_3: \square \Diamond (x = 3) \]

Is \( \varphi_3 \) LOOP-satisfiable?

Promising formulas:

\[ \Diamond (x = 3) \text{ promising } (x = 3) \]
\[ \neg \square \Diamond (x = 3) \text{ promising } \neg \Diamond (x = 3) \]

Pruned tableau \( T_{\varphi_3} \) (Fig. 5.6)
Behavior graph $\mathcal{B}_{(\text{LOOP}, \varphi_3)}$ (Fig. 5.11)
\[ S = \{ (s_0, A_1^-), (s_1, A_1^-), (s_2, A_1^-), (s_3, A_0^+) \} \]

is an adequate subgraph:

fair \( \tau \) taken in \( S \)
fulfilling

Therefore, by proposition 5.11, program LOOP has a computation satisfying \( \varphi_3: \Box \Diamond (x = 3) \)

The periodic computation \( \sigma: (x: 0, x: 1, x: 2, x: 3)^\omega \) satisfies \( \varphi_3 \)
From Atom Tableau $T_\varphi$

to $\omega$-Automaton $A_\varphi$

For temporal formula $\varphi$, construct the $\omega$-automaton

$$A_\varphi : \langle N, N_0, E, \mu, F \rangle$$

where

- **Node labeling $\mu$:**
  
  For node $n \in N$ labeled by atom $A$ in $T_\varphi$,
  $$\mu(n) = \text{state}(A).$$

- **Acceptance condition $F$:**

  Muller:
  $$F = \{ \text{SCS } S \mid S \text{ is fulfilling } \}$$

  Street:
  $$F = \{ (P_\psi, R_\psi) \mid \psi \in \Phi_\varphi \text{ promises } r \},$$
  where
  $$P_\psi = \{ A \mid \neg\psi \in A \}$$
  $$R_\psi = \{ A \mid r \in A \}$$
Example: $\varphi: \Diamond p$

Tableau $T_{\varphi}$:

$$A_1^+: \{p, \Box \Diamond p, \Diamond p\} \quad A_2^-: \{\neg p, \Box \Diamond p, \Diamond p\}$$

$$A_3^+: \{p, \neg \Box \Diamond p, \Diamond p\}$$

$$A_4^+: \{\neg p, \neg \Box \Diamond p, \neg \Diamond p\}$$
Example: $A \square p$ from $T \square p$

\[
\begin{align*}
F_M &= \{\{n_1\}, \{n_1, n_2\}, \{n_4\}\} \\
F_S &= \{(P \square p, R \square p)\} \\
&= \{(\{n_4\}, \{n_1, n_3\})\} \\
&\approx \{(\{n_4\}, \{n_1\})\}
\end{align*}
\]

since no path can visit $n_3$ infinitely often.
Abstraction

Abstraction = a method to verify infinite-state systems.

Idea:

<table>
<thead>
<tr>
<th>abstraction</th>
<th>→</th>
<th>Abstract program $P^A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program $P$ (infinite state)</td>
<td>→</td>
<td>Abstract property $\varphi^A$</td>
</tr>
<tr>
<td>Property $\varphi$</td>
<td>→</td>
<td>$P^A \models \varphi^A$</td>
</tr>
</tbody>
</table>

We want to ensure that if $P^A \models \varphi^A$ then $P \models \varphi$. 
**Abstraction (Cont’d)**

How do we obtain such an abstraction function?

- 1) Abstract the domain to a finite-state one (*data abstraction*):
  For variables $\vec{x}$ ranging over domain $D$, find an abstract domain $D^A$ and an abstraction function $\alpha : D \rightarrow D^A$.

- 2) From the data abstraction it is possible to compute an abstraction for the program and for the property such that if $P^A \models \varphi^A$ then $P \models \varphi$.
Example: Abstracting Bakery

Program MUX-BAK (infinite-state program)

\[
\begin{align*}
P_1 &::= \begin{cases}
\text{loop forever do} \\
\ell_0 : \text{noncritical} \\
\ell_1 : y_1 \coloneqq y_2 + 1 \\
\ell_2 : \text{await } y_2 = 0 \lor y_1 \leq y_2 \\
\ell_3 : \text{critical} \\
\ell_4 : y_1 \coloneqq 0 \\
\end{cases} \\
\| \\
P_2 &::= \begin{cases}
\text{loop forever do} \\
m_0 : \text{noncritical} \\
m_1 : y_2 \coloneqq y_1 + 1 \\
m_2 : \text{await } y_1 = 0 \lor y_2 < y_1 \\
m_3 : \text{critical} \\
m_4 : y_2 \coloneqq 0 \\
\end{cases}
\end{align*}
\]

Abstract domain: the boolean algebra over \( B = \{b_1, b_2, b_3 : \text{boolean}\} \), with
\[
\begin{align*}
b_1 &\iff y_1 = 0 \\
b_2 &\iff y_2 = 0 \\
b_3 &\iff y_1 \leq y_2
\end{align*}
\]
Example: Abstracting Bakery (Cont’d)

Program MUX-BAK-ABSTR (finite-state program)

\[ P_1 ::\]
\[
\begin{align*}
\text{loop forever do} \\
\ell_0 : & \text{noncritical} \\
\ell_1 : & (b_1, b_3) := (\text{false, false}) \\
\ell_2 : & \text{await } b_2 \lor b_3 \\
\ell_3 : & \text{critical} \\
\ell_4 : & (b_1, b_3) := (\text{true, true})
\end{align*}
\]

\[ P_2 ::\]
\[
\begin{align*}
\text{loop forever do} \\
m_0 : & \text{noncritical} \\
m_1 : & (b_2, b_3) := (\text{false, true}) \\
m_2 : & \text{await } b_1 \lor \neg b_3 \\
m_3 : & \text{critical} \\
m_4 : & (b_2, b_3) := (\text{true, } b_1)
\end{align*}
\]

This program can now be checked for mutual exclusion, bounded overtaking, response.

Show MUX-BAK-ABSTR \( \models \square \neg (at_\ell_3 \land at_m_3) \). Then it follows that MUX-BAK \( \models \square \neg (at_\ell_3 \land at_m_3) \).