Model checking for LTL (= satisfiability over a finite-state program)

Angelo Montanari

Department of Mathematics , Computer Science, and Physics
University of Udine, Udine Italy

P-validity and P-satisfiability problems (of φ)

P**-validity** problem (of φ)

Main question: given a finite-state program P and a formula φ, is φ P-valid, that is, do all P-computations satisfy φ?

P**-satisfiability** problem (of φ)

Main question: given a finite-state program P and a formula φ, is there a P-computation which satisfies φ?

To determine whether φ is P-valid, it suffices to employ an algorithm for deciding if there is a P-computation which satisfied $\neg\varphi$.

The algorithm for solving the P-satisfiability of φ makes use of the tableau for φ T_φ

Basic definitions

- For each atom A, let *state*(A) be the conjunction of all state/local formulas in A (by R_{sat}, *state*(A) must be satisfiable).
- Atom A is **consistent** with state s if $s \models$ *state*(A), that is, all state formulas in A are satisfiable by s.
- Let $\theta : A_0, A_1, \ldots$ be a path in T_φ and let $\sigma : s_0, s_1, \ldots$ be a computation of P. θ is **trail** of T_φ over σ if A_j is consistent with s_j, for all $j \geq 0$.
- For each atom $A \in T_\varphi$, $\delta(A)$ denotes the set of successors of A in T_φ.

The behavior graph

Given a finite-state program P and an LTL formula φ, we construct the **behavior graph** of (P, φ), denoted $\mathcal{B}_{(\mathcal{P},\varphi)}$, as the product of the graph for P (G_P) and the tableau for φ (T_φ).

- **nodes** (s, A), where s is a state of P and A is an atom consistent with s;
- there exists a τ**-labeled edge** from (s, A) to (s', A') only if $s' \in \tau(s)$ (s' is a τ-successor of s) and $A' \in \delta(A)$ in the pruned tableau T_φ (A' is a successor of A in T_φ);
- **initial** φ**-nodes** are pairs (s, A), where s is an initial state for P, A is an initial φ-atom in T_φ (that is, $\varphi \in A$), and A is consistent with s.

Algorithm BEHAVIOR-GRAPH to construct $\mathcal{B}_{(\mathcal{P},\varphi)}$

Algorithm **BEHAVIOR-GRAPH**

- Place in $\mathcal{B}_{(\mathcal{P},\varphi)}$ all initial φ-nodes (s, A)
- Repeat until no new nodes or new edges can be added the following steps.
  Let (s, A) be a node in $\mathcal{B}_{(\mathcal{P},\varphi)}$, let $\tau \in \mathcal{T}$ be a transition, and let (s', A') be a pair such that: (i) s' is a τ-successor of s, $A' \in \delta(A)$ in the pruned tableau T_φ, and A' is consistent with s'.
  - Add (s', A') to $\mathcal{B}_{(\mathcal{P},\varphi)}$, if it is not already there.
  - Draw a τ-edge from (s, A) to (s', A') if it not already there.

An example: the system LOOP

The system **LOOP**

Initially $x = 0$

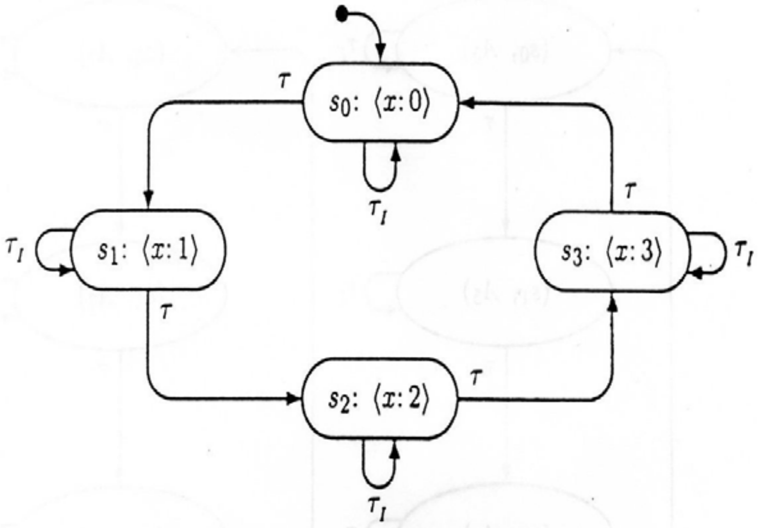
Transitions: (i) the idling transition τ_I and (ii) a transition τ, with transition relation $\rho_\tau : x' = (x + 1) \bmod 4$

The set of weakly fair (just) transitions is $\mathcal{J} = \{\tau\}$

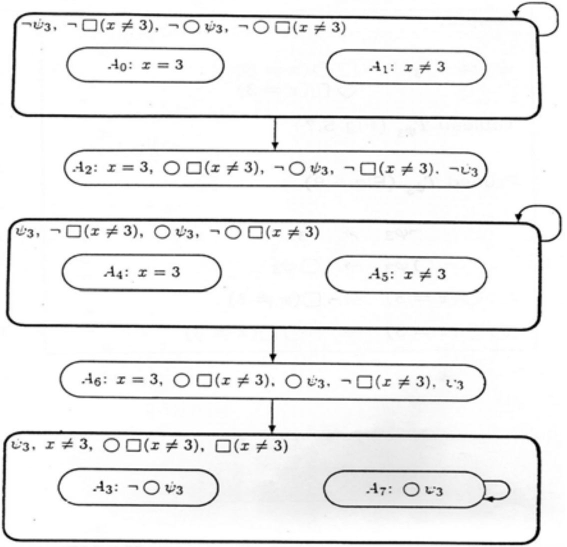
Let us consider the LTL formula $\psi_3 : \Diamond\Box(x \neq 3)$

In the next transparencies, we respectively provide the state-transition graph G_{LOOP}, the pruned tableau T_{ψ_3}, and the behavior graph $\mathcal{B}_{(LOOP,\psi_3)}$.

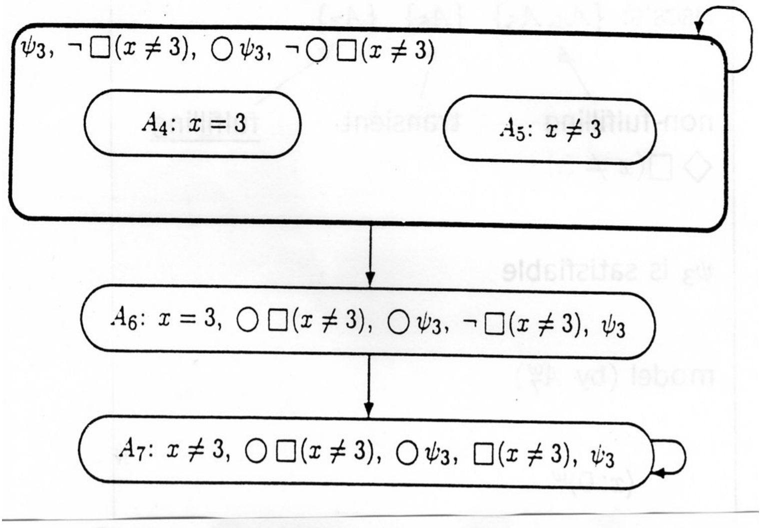
The state-transition graph of system LOOP (G_{LOOP})



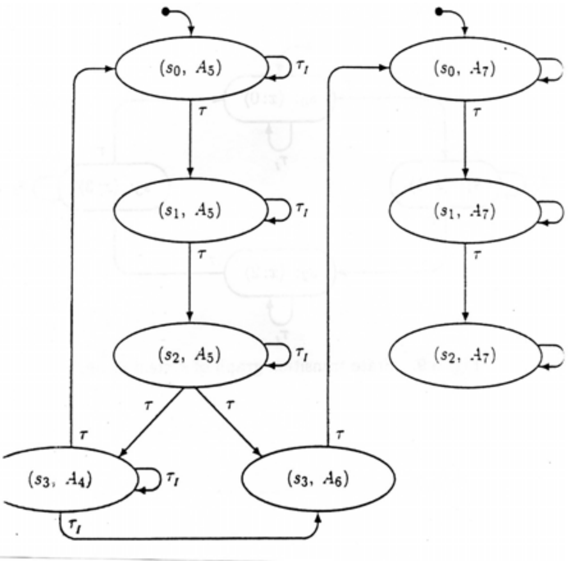
The complete tableau



The pruned tableau (T_{ψ_3})



The behavior graph $\mathcal{B}_{(LOOP,\psi_3)}$



Paths in the behavior graph $\mathcal{B}_{(\mathcal{P},\varphi)}$

Proposition.
Let φ be an LTL formula.
The infinite sequence $\pi : (s_0, A_0)(s_1, A_1)\ldots$, where (s_0, A_0) is an initial φ-node, is a path in $\mathcal{B}_{(\mathcal{P},\varphi)}$

if and only

- $\sigma_\pi : s_0 s_1 \ldots$ is a run of P (computation less fairness)
- $\theta_\pi : A_0 A_1 \ldots$ is a trail of T_φ over σ_π (for all $j \geq 0$, A_j is consistent with s_j)

Example.
In $\mathcal{B}_{(LOOP,\psi_3)}$, the path $\pi : ((s_0, A_5)(s_1, A_5)(s_2, A_5)(s_3, A_4))^\omega$ induces $\sigma_\pi : (s_0 s_1 s_2 s_3)^\omega$ (run of LOOP) and $\theta_\pi : (A_5 A_5 A_5 A_4)^\omega$ (trail of T_φ over σ_π)

P-satisfiability of φ by path

Proposition.
Let φ be an LTL formula.
There exists a P-computation which satisfies φ

if and only if

there is an infinite path π in $\mathcal{B}_{(\mathcal{P},\varphi)}$, starting from an initial φ-node, such that

- σ_π is a fair run (computation)
- θ_π is a fulfilling trail over σ_π

Example.
The trail $\theta_\pi : (A_5 A_5 A_5 A_4)^\omega$ is not fulfilling (both atoms A_5 and A_4 include $\diamondsuit\square(x \neq 3)$ and $\neg\square(x \neq 3)$).

Adequate subgraphs

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

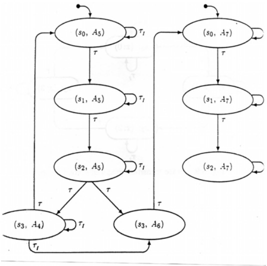
- node (s', A') is a τ**-successor** of node (s, A) if $\mathcal{B}_{(\mathcal{P},\varphi)}$ contains a τ-edge connecting (s, A) to (s', A')
- transition τ is **enabled** on node (s, A) if it is enabled on state s

Given a subgraph $S \subseteq \mathcal{B}_{(\mathcal{P},\varphi)}$,

- transition τ is **taken** in S if there exist two nodes (s, A) and (s', A') in S such that (s', A') is a τ-successor of (s, A)
- S is **just** (resp., **compassionate**) if every just (resp., compassionate) transition $\tau \in \mathcal{J}$ (resp., $\tau \in \mathcal{C}$) is either taken in S or is disabled on some nodes (resp., all nodes) in S
- S is **fair** it is both just and compassionate
- S is **fulfilling** if every promising formula is fulfilled by an atom A such that $(s, A) \in S$ for some state s
- S is **adequate** if it is fair and fulfilling

Adequate strongly connected subgraphs and satisfiability

Proposition.
A finite-state program P has a computation σ which satisfies φ if and only if the behavior graph $\mathcal{B}_{(\mathcal{P},\varphi)}$ has an adequate strongly connected subgraph
Example (LOOP and $\psi_3 : \Diamond\Box(x \neq 3)$).
The behavior graph $\mathcal{B}_{(LOOP,\psi_3)}$ has no adequate subgraphs.



Example (cont'd)

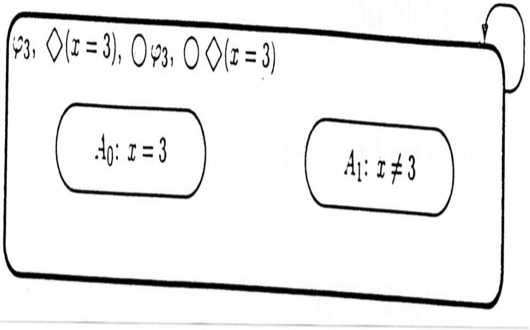
Let us check the maximal strongly connected subgraphs (MSCS):

- $\{(s_0, A_5), (s_1, A_5), (s_2, A_5), (s_3, A_4)\}$ is fair, but not fulfilling (ψ_3 belongs to both A_4 and A_5, and it promises $\Box(x \neq 3)$, but $\Box(x \neq 3) \notin A_4, A_5$)
- $\{(s_0, A_7)\}$, $\{(s_1, A_7)\}$, and $\{(s_2, A_7)\}$ are fulfilling, but not fair (they are not just with respect to transition τ)
- $\{(s_3, A_6)\}$ is neither fair (it is not just with respect to τ) nor fulfilling (it is transient)

Hence, there are **no adequate subgraphs** in $\mathcal{B}_{(LOOP, \psi_3)}$.
By the last proposition, it follows that LOOP has no computation that satisfies $\psi_3 : \Diamond\Box(x \neq 3)$

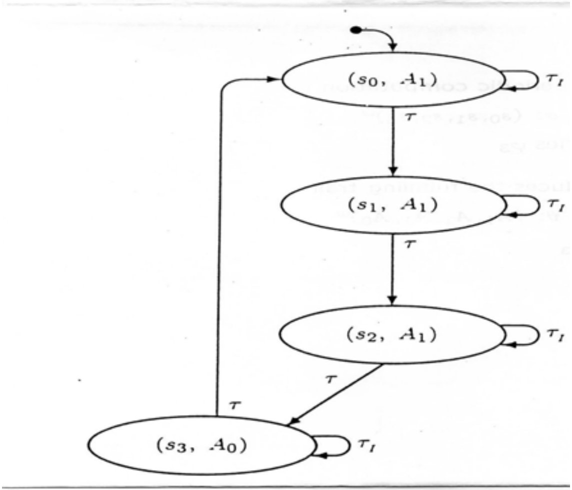
Another example

Example (LOOP and $\phi_3(= \neg\psi_3) : \Box\Diamond(x = 3)$).
The pruned tableau is the following one:



Another example (cont'd)

The behavior graph $\mathcal{B}_{(LOOP,\phi_3)}$ is the following one:



Another example (cont'd)

The subgraph $S = \{(s_0, A_1), (s_1, A_1), (s_2, A_1), (s_3, A_0)\}$ is an **adequate subgraph**, as it is both fair (τ is taken in S) and fulfilling ($\diamondsuit(x = 3)$ belongs to both A_0 and A_1, but $x = 3$ belongs to A_0)

By the last proposition, it follows that LOOP has computation that satisfies $\phi_3 : \square\diamondsuit(x = 3)$

The periodic computation $\sigma : (s_0 s_1 s_2 s_3)^\omega$ satisfies ϕ.

It induces the fulfilling trail $\theta : (A_1 A_1 A_1 A_0)^\omega$ in T_ϕ.

How to find adequate subgraphs?

Checking MSCS is **not enough**:

$$S' \subset S$$

S' just **implies** S just

S' fulfilled **implies** S fulfilled

but

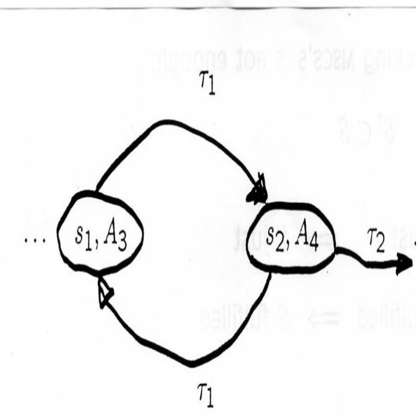
S' compassionate **do not imply** S compassionate

Therefore, it is possible that S is not adequate, but S' is adequate

A counterexample

S' compassionate **do not imply** S compassionate

Let τ_2 belong to the set of compassionate transitions. τ_2 is enabled on s_2 and disabled on s_1



A counterexample (cont'd)

The strongly connected subgraph $S' = \{(s_1, A_3)\}$ is compassionate (τ_2 is disabled on all the states in this subgraph)

The strongly connected subgraph $S = \{(s_1, A_3)(s_2, A_4)\}$, that includes S', is not compassionate (τ_2 is enabled on (s_2, A_4), but it is not taken in S)

Algorithm ADEQUATE-SUB

Algorithm **ADEQUATE-SUB**

- accepts as input a strongly connected subgraph S and returns as output a strongly connected subgraph $S' \subseteq S$

  If $S' = \emptyset$ - S contains no adequate subgraphs

  Otherwise - S' is an adequate strongly connected subgraph in S

- Notation:

  $EN(\tau, S)$ - the set of all nodes (s, A) in S on which τ is enabled

Algorithm ADEQUATE-SUB (cont'd)

Algorithm ADEQUATE-SUB checks for adequate subgraphs
recursive function adequate-sub(S: SCS) **returns** SCS
if S is not fulfilling **then return** $\emptyset$ – failure
if S is not just **then return** $\emptyset$ – failure
if S is compassionate **then return** S – success
– S is fulfilling and just but not compassionate. Let $T \subseteq \mathcal{C}$
– be the set of all compassionate transitions that are not taken
– in S. Clearly, $EN(T, S) \neq \emptyset$.
let $U = S - EN(T, S)$. Decompose U into MSCSs $U_1 \dots U_k$.
let $V = \emptyset, i = 1$
while $V = \emptyset$ and $i \leq k$ **do**
let $V = \; adequate - sub(U_i); i := i + 1$
end-while
return V

An example: the system LOOP+

The system **LOOP+**

Initially $x = 0$

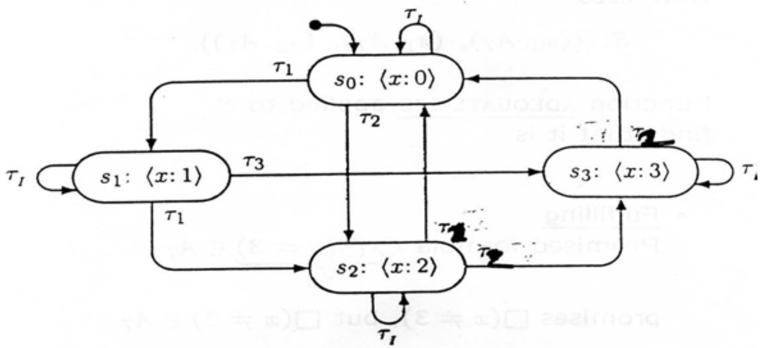
LTL formula $\psi_3 : \Diamond\Box(x \neq 3)$

Transitions: (i) the idling transition τ_I; (ii) $\mathcal{J} = \{\tau_1, \tau_2\}$; (iii) $\mathcal{C} = \{\tau_3\}$

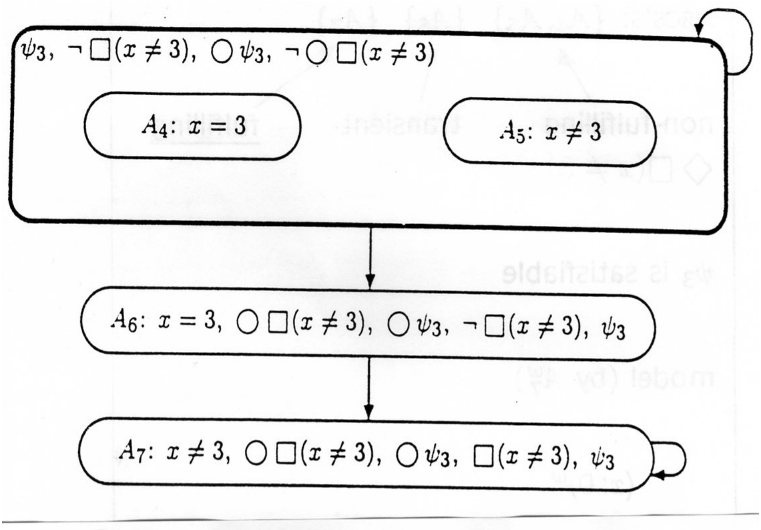
New MSCS: $S = \{(s_0, A_7), (s_1, A_7), (s_2, A_7)\}$

In the next transparencies, we respectively provide the state-transition graph G_{LOOP+}, the pruned tableau T_{ψ_3}, and the behavior graph $\mathcal{B}_{(LOOP+,\psi_3)}$.

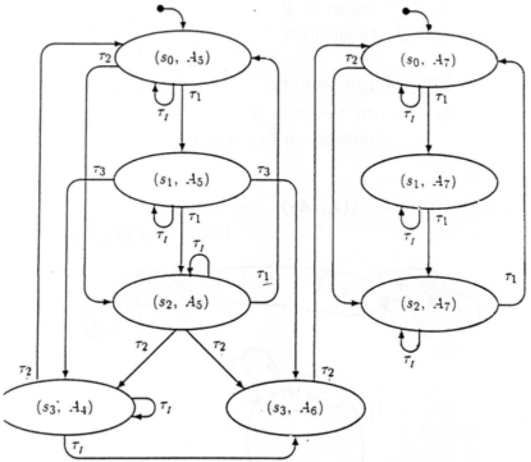
The state-transition graph of system LOOP+ (G_{LOOP+})



The pruned tableau (T_ψ)



The behavior graph $\mathcal{B}_{(\mathcal{LOOP}+,\psi)}$



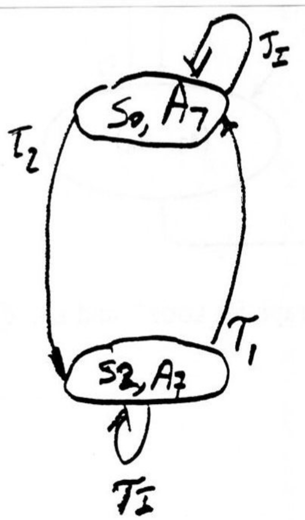
Application of the function ADEQUATE-SUB

Function **ADEQUATE-SUB** applied to S finds that it is

- **fulfilling**: the formula $\psi_3 : \Diamond\Box(x \neq 3)$, that promises $\Box(x \neq 3)$, belongs to A_7, but $\Box(x \neq 3)$ belongs to A_7 as well
- **just**: $\tau_2 \in \mathcal{J}$ is taken in S and $\tau_1 \in \mathcal{J}$ is taken in S
- **not compassionate**: $\tau_3 \in \mathcal{C}$ is not taken in S, but it is enabled on (s_1, A_7)

Construct $U : \{(s_0, A_7), (s_2, A_7)\}$ by removing (s_1, A_7)

The subgraph U



20-6

The subgraph U (cont'd)

U is a strongly connected subgraph (no decomposition is needed)

U is adequate:

- **fulfilling** – A_7 fulfills his promise $\Box(x \neq 3)$
- **fair** – τ_1 and τ_2 are enabled s_2 and s_0, respectively, and both are taken in U

Hence, system LOOP+ has a computation $\sigma : (s_0 s_2)^\omega$ that satisfies $\psi_3 : \Diamond\Box(x \neq 3)$

Algorithms SAT and P-SAT

To summarize ..

Algorithm **SAT** to check whether a temporal formula φ is satisfiable

Algorithm **P-SAT** to check the satisfiability of a formula φ over a program (to check whether a finite-state program P has a computation which satisfies a temporal formula φ)

Algorithm P-SAT

To check whether a finite-state program P has a computation that satisfies a temporal formula φ, perform the following steps:
Construct the state-transition graph G_P.
Construct the pruned tableau T_φ.
Construct the behavior graph $\mathcal{B}_{(\mathcal{P},\varphi)}$.
Decompose $\mathcal{B}_{(\mathcal{P},\varphi)}$ into MSCS $S_1, \ldots, S_t$.
For each $i = 1, \ldots, t$, **apply** algorithm **ADEQUATE-SUB** to S_i.

If any of these applications returns a nonempty result, P has a computation satisfying φ. This computation can be constructed by forming a path π that leads from an initial node to the returned adequate subgraph S, and then continues to visit each node S infinitely many times. The desired computation is the computation σ_π induced by π. If all applications return the empty set as result, P has no computation satisfying φ.

P-validity of a formula φ

To check P-validity of a formula φ, apply algorithm P-SAT to check whether there are P-computation satisfying $\neg\varphi$

- If there is a P-computation satisfying $\neg\varphi$, then φ is not P-valid
- If there are no P-computations satisfying $\neg\varphi$, then φ is P-valid