Transition Predicate Abstraction and Fair Termination

[A. Podelski & A. Rybalchenko. POPL '05]

Goal: Develop an automated approach
to checking liveness properties under fairness.

Approach:
- Generalize predicate abstraction
to transition predicate abstraction.
- We transform the program P
into an abstract program $P^\#$
that preserves • termination and • fairness.
- The abstract program $P^\#$
is a node- and edge-labeled graph:
  - ↪ Fairness depends on the edge labels
  We have an algorithm
  that goes over the abstract program
  and marks some nodes as fair.
  - ↪ Termination can be checked
  on the node labels.
  - ↪ The verification method tries to show
  that every fair state terminates.

Fairness: Different from the last lecture,
we do not rely on the automata-theoretic approach
but treat fairness directly.
to deal with fairness,

Notions:
Justice: transition continually enabled ⇒ taken infinitely often.
Compassion: transition enabled inf. often ⇒ taken inf. often.

Intuition: $P^\#$ is a new form of transition invariant,
one that encodes justice / compassion assumptions.

Abstract-Transition Programs

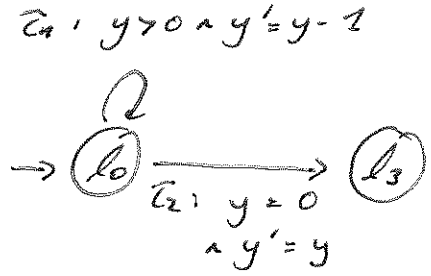
Idea: Abstract relations instead of states.
Use transition predicate abstraction
instead of predicate abstraction.

Do not abstract a program into an abstract-state program
but into an abstract-transition program.

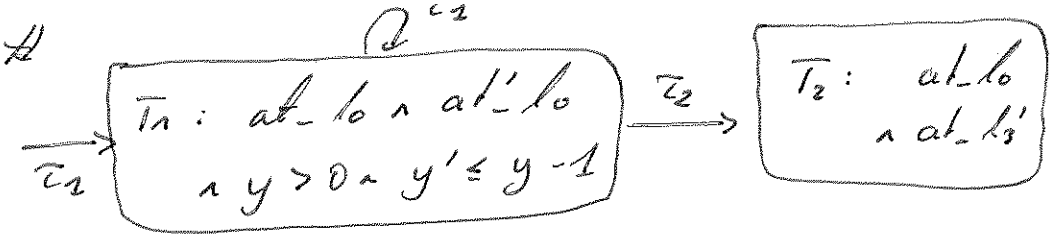
Abstract transition = binary relation represented by
a conjunction of transition predicates
↳ They form the states
in the abstract-transition program.

Example:

LOOP: l_0 : while $y > 0$ do
l_1 : $y = y - 1$
l_2 : skip
l_3 :



$LOOP^\#$



What is the meaning of $LOOP^\#$?

In abstract-state programs: $s_1 \xrightarrow{\tau} s_2$, if $\exists s_1 \in S_1 . \exists s_2 \in S_2 . s_2 \in \rho_\tau(s_1)$

Here: If the relation $(s, s') \in T_1$ can be extended
to $(s, s'') \in T_2$ by taking transition τ,
meaning $s'' \in \rho_\tau(s')$, then $T_1 \xrightarrow{\tau} T_2$.

Programs and Computations:

Definition:

We represent programs by fair transition systems

$$P = (\Sigma, \Theta, T, J, C)$$

with

Σ = set of states

$\Theta \subseteq \Sigma$ = set of initial states

T = finite set of transitions.

Each $\tau \in T$ has a transition relation $\rho_\tau \subseteq \Sigma \times \Sigma$.

$J \subseteq T$ = set of just transitions

$C \subseteq T$ = set of compassionate transitions.

A computation is a sequence of states $s_0 s_1 \ldots$

that is either infinite or no more enabled

so that
- $s_0 \in \Theta$
- $(s_i, s_{i+1}) \in \rho_\tau$ for some $\tau \in T$.

Transition Predicates:

A transition predicate is a relation $\rho \subseteq \Sigma \times \Sigma$,

typically given as assertion over unprimed and primed variables.

We let $Id = \{(s, s) \mid s \in \Sigma\}$.

From now on, we fix a finite set of transition predicates $\mathcal{P}$.

We assume $\{at_\ell, at'_\ell\} \subseteq \mathcal{P}$ for all program locations ℓ.

The set of abstract transitions is

$$\overline{\mathcal{P}}^{\#} = \{\rho_1 \wedge \ldots \wedge \rho_n \mid n \geq 0, \rho_i \in \mathcal{P}\}.$$

Definition:

An abstract-transition program

$$P^\# = (V, E, v_0, L_V, L_E)$$

consists of

- a finite set of nodes V with root node $v_0 \in V$
- a finite set of edges E,
- $L_V : V \to T_P^\#$, each node is labeled by an abstract transition, with $L_V(v_0) = Id$.
  We write T_v for $L_V(v)$.
- $L_E : E \to T$, each edge is labeled by a transition of the program.

We use $V^- = V \setminus \{v_0\}$.

Definition:

The abstract-transition program $P^\# = (V, E, v_0, L_V, L_E)$ is an abstraction of program $P = (\Sigma, \Theta, T, J, C)$, denoted by $P \preceq P^\#$, if

$$\forall v_1 \in V.\ \forall (s, s') \in T_{v_1}.\ \forall \tau \in T.\ \forall s'' \in S_\tau(s')$$
$$\exists v_2 \in V^-.\ (s, s'') \in T_{v_2} \wedge (v_1, v_2) \in E \wedge L_E((v_1, v_2)) = \tau.$$

Automated Abstraction $P \mapsto P^\#$

input: program with transitions $\tilde{\tau}$
$\mathcal{P}$ a finite set of predicates

output: abstract-transition program $P^\# = (V, E, v_0, L_V, L_E)$

begin:
$Q :=$ empty queue

$\alpha := \lambda T.\ \wedge \{p \in \mathcal{P} \mid T \subseteq p\}$ // this is as for predicate abstraction

$v_0 :=$ new node with label Id

$V := \{v_0\}$;

enqueue(Q, v_0);

$E := \emptyset$

<u>while</u> Q not empty <u>do</u>

  $u :=$ dequeue(Q);

  <u>for each</u> $\tau \in \tilde{\tau}$ <u>do</u>

    $T := \alpha(T_u \circ S_\tau)$;

    <u>if</u> $T = \emptyset$ <u>then</u> continue with next $\tau \in \tilde{\tau}$ <u>fi</u>

    <u>if</u> $\exists w \in V$ with $T = T_w$ <u>then</u>

      $v := w$;

    <u>else</u>

      $v :=$ new node labeled T;

      $V := V \cup \{v\}$

      enqueue(Q, v);

    <u>fi</u>

    $(u, v) :=$ new edge labeled by τ;

    $E := E \cup \{(u, v)\}$;

  <u>od</u>

<u>od</u>

<u>end</u>

<u>Example</u>:

Let $\mathcal{P} = \{x \geq 0,\ x' \leq x - 1,\ x' = x,\ x' \geq x + 1\}$

Let $T = x > 0 \wedge x' = x - 1$

Then $\alpha(T) = x \geq 0 \wedge x' \leq x - 1$.

Overall Method:

We compute, for each node v,
a set of transitions $\mathrm{asc}(\mathcal{L}_v) \subseteq \bar{T}$.
Then we check a condition on $\mathrm{asc}(\mathcal{L}_v)$
that depends on the fairness notion.
We mark v as *fair* if the check is positive.
We check whether T_r terminates for each fair node v.
We return *property verified*, if this holds.

It remains to define $\mathrm{asc}(\mathcal{L}_v)$.
We can understand $pt^\#$ as a deterministic finite automaton,
we just forget about the node labels.
Let $\mathcal{A}_v = (V, E, v_0, \{v\})$ have v as the only final state.
Then $\mathcal{L}_v = \mathcal{L}(\mathcal{A}_v)$.

$\mathrm{asc}(\mathcal{L}_v)$ = all transitions appearing in some word in $\mathcal{L}_v$.
$= \bigcap \{ H \mid H \subseteq \bar{T} \mid \mathcal{L}_v \subseteq H^* \}$.

Justice:

Let $\mathrm{Succ}_{\bar{T}}(s) = \{ s' \mid (s,s') \in \mathrm{Succ}_{\bar{T}} \}$.
Transition $\bar{t}$ is *enabled in* s, if $\mathrm{Succ}_{\bar{t}}(s) \neq \emptyset$.
We write $\mathrm{En}(\bar{t}) = \{ s \mid \mathrm{Succ}_{\bar{t}}(s) \neq \emptyset \}$ for the set of states
in which $\bar{t}$ is enabled.
Recall that $J \subseteq \bar{T}$ are the just transitions.

Assumption 1:

$\forall \tau^j \in J.\ \forall \tau \in \mathcal{T}.\ \tau^j \neq \tau \Rightarrow S_{\tau^j} \cap S_\tau = \emptyset.$

Just transitions are disjoint from all other transitions.

Holds in our programming model
as we have single edges between control locations.

Assumption 2:

$\forall \tau^j \in J.\ \forall \tau \in \mathcal{T}.\ \tau^j \neq \tau \Rightarrow En(\tau^j) = En(\tau) \vee En(\tau^j) \cap En(\tau) = \emptyset.$

is also not a real restriction.

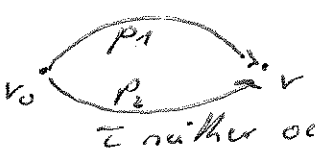
Definition:

$fairJ(v) := \forall \tau^j \in J.\ just(v, \tau^j)$

$just(v, \tau^j) := \tau^j \in asc(\pi_v) \vee \exists \tau \in asc(\pi_v).\ En(\tau) \cap En(\tau^j) = \emptyset.$

Remark:

- $just(v, \tau^j)$ says that τ^j is either taken or not continuously enabled on some path from v_0 to v.
- One may think that $fairJ(v)$ is rather approximate as there may be paths possibly leading to v that contain τ or disable τ but other paths p_2 do not.
  The point, however, is that
  v represents, say, a transition sequence from s to s'.
  Even if this sequence was p_2, there could be another computation in which it is p_1.



τ neither occurs nor is disabled.

If we call, we lose the link between the state changes and the paths.

Theorem (Just Termination):

Program P justly terminates (no infinite computation satisfies justice),

if $\forall v \in V^{-}.\ fair_J(v) \Rightarrow$ well-founded(T_v).

Proof:

Assume P does not justly terminate.

We show that there is a non-root node v

that is labeled by a non-well-founded abstract transition T_v

so that for every $\tau^i \in J$ the predicate just(v, τ^i) holds.

- Let $\sigma = s_1 s_2 \ldots$ be an infinite computation induced by the infinite sequence of transitions $\tau_1 \tau_2 \ldots$ that satisfies justice.
- Computation σ partitions J into
  - J^{d}(isabled) = transitions that are not continually enabled
  - J^{t}(aken) = transitions that are continually enabled and hence taken infinitely often.

Since σ satisfies justice, there is a set $L = l_1, l_2, \ldots \subseteq \mathbb{N}$ so that

$\forall \tau \in J^{d}.\ \forall i \geq 1.\ \exists l_i < p < l_{i+1}.\ sp \notin En(\tau)$

$\forall \tau \in J^{t}.\ \forall i \geq 1.\ \exists l_i < p < l_{i+1}.\ \tau_p = \tau.$

- For the sequence $\tau_1 \tau_2 \ldots$ and the set L, we define $f: L \longrightarrow V$ // nodes of $P^{\#}$

$(h, \ell) \longmapsto v$, if $\tau_h \ldots \tau_{\ell-1} \in \mathcal{L}_v$.

The function exists by the lemma above.

- By Ramsey's theorem, there is an infinite set of positions
  $k = k_1 < k_2 < \dots$ in L and
  a node v
  so that
  $$f(k_i, k_{i+1}) = v \quad \text{for all } i \in \mathbb{N}.$$

This means
$$(s_{k_i}, s_{k_{i+1}}) \in T_v \quad \text{for all } i \in \mathbb{N}.$$
This contradicts the fact that T_v is well-founded.

- We show that $\bar{\tau}^i \in \mathrm{abs}(\mathcal{L}_v)$ for all $\bar{\tau}^i \in J^i$.
By the choice of L and the fact that $k \in L$,
there are k_i and k_{i+1} so that
$$\bar{\tau}^i \in \{\tau_{k_i}, \dots, \tau_{k_{i+1}-1}\}.$$
Since $\tau_{k_i} \dots \tau_{k_{i+1}-1} \in \mathcal{L}_v$, we have $\bar{\tau} \in \mathrm{abs}(\mathcal{L}_v)$.

- We show that for every $\bar{\tau}^d \in J^d$
there is $\bar{\tau} \in \mathrm{abs}(\mathcal{L}_v)$ so that $\mathrm{En}(\bar{\tau}) \cap \mathrm{En}(\bar{\tau}^d) = \emptyset$.
By the choice of L, there is a position p
between k_i and k_{i+1} so that τ^d is not enabled in s_p.
This means the transition from s_p to its successor
is by $\tau \neq \tau^d$.
We have $\tau \in \mathrm{abs}(\mathcal{L}_v)$ as for $\bar{\tau}^i$ above.
By Assumption 2, $\mathrm{En}(\tau^d) \cap \mathrm{En}(\tau) = \emptyset$. □

Example:

DNY-DOWN:

l_0: **while** $x = 1$ **do**

  l_1: $y = y + 1$

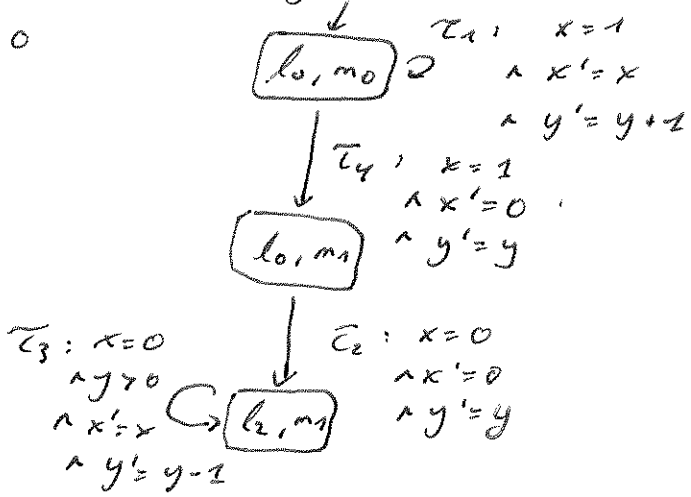
l_2: **while** $y > 0$ **do**

  l_3: $y = y - 1$

m_0: $x = 0$

m_1:

Control-flow graph:



Every transition is just,

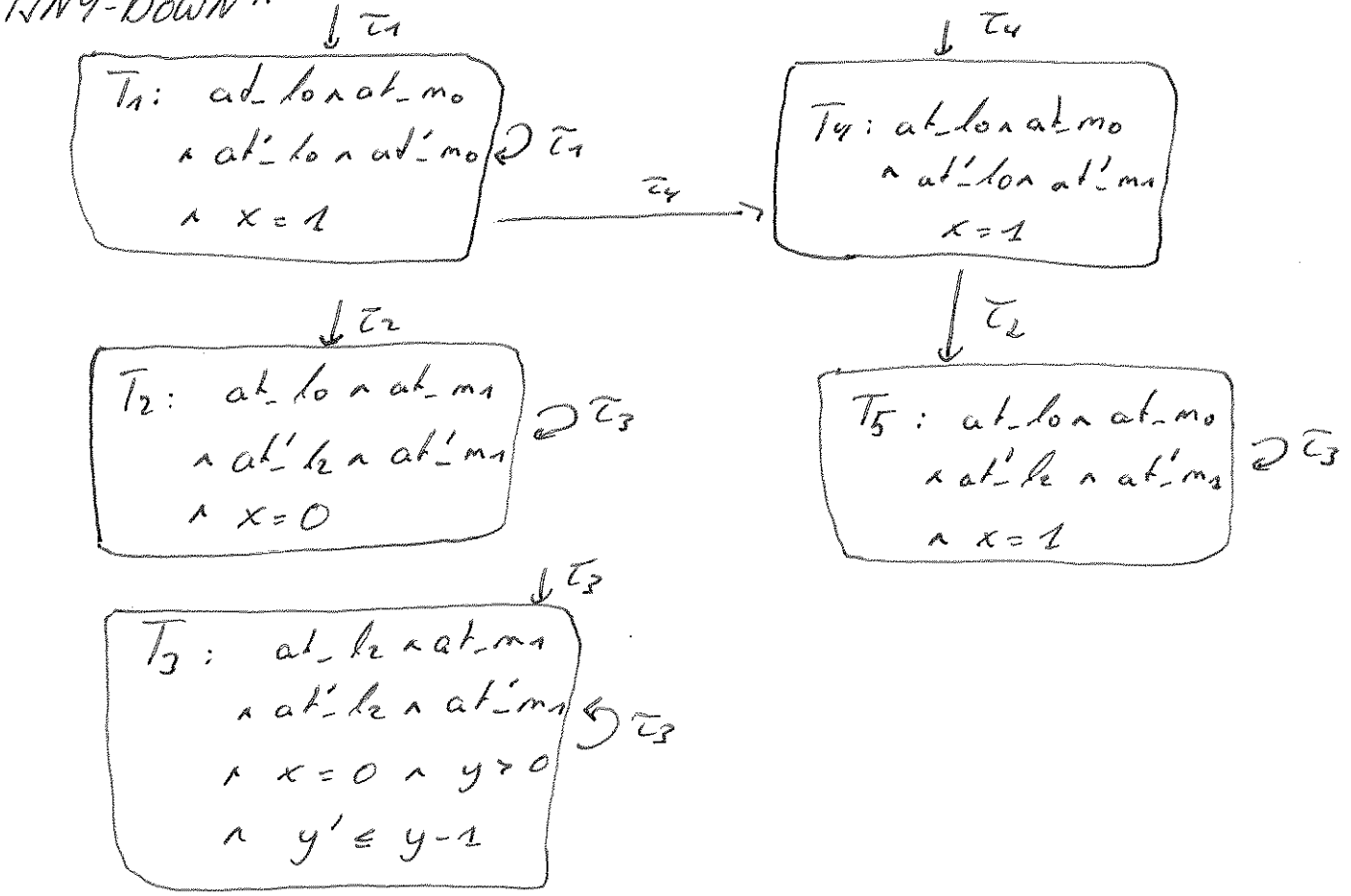
$$J = \{\tau_1, \ldots, \tau_4\}.$$

We use the transition predicates

$$\beta = \{x=0,\ x=1,\ y>0,\ y' \leq y-1,\ at\_l_0,\ at\_m_0,\ \ldots\}$$

The abstract-transition program is

DNY-DOWN#



The abstract transition T_A is *not* well-founded.

But just(v_A, T_A) does not hold

Hence, fair(v_A) does not hold and so

the well-foundedness of T_A is not required.