Temporal Verification of Reactive Systems

Yih-Kuen Tsay

Dept. of Information Management
National Taiwan University
Computational vs. Reactive Programs

Computational (Transformational) Programs

- Run to produce a final result on termination

An example:

```
[ local x : integer initially x = n;
    y := 0;
    while x > 0 do
        x, y := x - 1, y + 2x - 1
    od ]
```

- Only the initial values and the (final) result are relevant to correctness

- Can be specified by pre and post-conditions such as

  - `{n ≥ 0} y := ? {y = n^2}` or
  - `y : [n ≥ 0, y = n^2]`
Reactive Programs

- Maintaining an ongoing (typically not terminating) interaction with their environments

An example:

\[ s : \{0, 1\} \text{ initially } s = 1 \]

\[
\begin{align*}
    l_0 : & \text{ loop forever do} \\
         & \begin{align*}
             l_1 : & \text{ remainder;} \\
             l_2 : & \text{ request}(s); \\
             l_3 : & \text{ critical;} \\
             l_4 : & \text{ release}(s); \\
         \end{align*}
\end{align*}
\]

\[
\begin{align*}
    m_0 : & \text{ loop forever do} \\
         & \begin{align*}
             m_1 : & \text{ remainder;} \\
             m_2 : & \text{ request}(s); \\
             m_3 : & \text{ critical;} \\
             m_4 : & \text{ release}(s); \\
         \end{align*}
\end{align*}
\]

- Must be specified and verified in terms of their behaviors, including the intermediate states they go through.
The Framework

Computational Model: for providing an abstract syntactic base
- fair transition systems (FTS)
- fair discrete systems (FDS)

Implementation Language: for describing the actual implementation; will define syntax by examples; translated into FTS or FDS for verification

Specification Language: for specifying properties of a system; will use linear temporal logic (LTL)

Verification Techniques: for verifying that an implementation satisfies its specification
- algorithmic methods: state space exploration
- deductive methods: mathematical theorem proving
Three Kinds of Validity

- **Assertional Validity**: validity of non-temporal formulae, i.e., state formulae, over an arbitrary state (valuation).
- **General Temporal Validity**: validity of temporal formulae over arbitrary sequences of states.
- **Program Validity**: validity of a temporal formula over sequence of states that represent computations of the analyzed system.
Variables

Three kinds of variables will be needed:

- Program (system) variables
- Primed version of program variables: for referring to the values of program variables in the next state when defining a state transition.
- Specification variables: appearing only in formulae (but not in the program) that specify properties of a program.

We assume that all these variables are drawn from a universal set of variables $\mathcal{V}$.

For every unprimed variable $x \in \mathcal{V}$, its primed version $x'$ is also in $\mathcal{V}$.

Each variable has a type.
Assertions

For describing a system and its specification, we assume an underlying first-order assertion language over $\forall$.

The language provides the following elements:

- **Expressions** (corresponding to first-order terms): variables, constants, and functions applied to expressions
- **Atomic formulae**: propositions or boolean variables and predicates applied to expressions
- **Assertions** or **state formulae** (corresponding to first-order formulae): atomic formulae, boolean connectives applied to formulae, and quantifiers applied to formulae
A fair transition system (FTS) $\mathcal{P}$ is a tuple $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$:

- $V \subseteq \mathcal{V}$: a finite set of typed state variables, including data and control variables. A (type-respecting) valuation of $V$ is called a $V$-state or simply state. The set of all $V$-states is denoted $\Sigma_V$.

- $\Theta$ : the initial condition, an assertion characterizing the initial states.

- $\mathcal{T}$ : a set of transitions, including the idling transition. Each transition is associated with a transition relation, relating a state and its successor state(s).

- $\mathcal{J} \subseteq \mathcal{T}$ : a set of just (weakly fair) transitions.

- $\mathcal{C} \subseteq \mathcal{T}$ : a set of compassionate (strongly fair) transitions.
Transitions of an FTS

The transition relation of a transition $\tau \in \mathcal{T}$ is expressed as an assertion $\rho_{\tau}(V, V')$:

- **Example:** $x = 1 \land x' = 0$.
  For $s, s' \in \Sigma_V$, $\langle s, s' \rangle \models x = 1 \land x' = 0$ holds if the value of $x$ is 1 in state $s$ and the value of $x$ is 0 in (the next) state $s'$.

- **$\tau$-successor**
  - $s'$ is a $\tau$-successor of $s$ if $\langle s, s' \rangle \models \rho_{\tau}(V, V')$.
  - $\tau(s) \triangleq \{ s' \mid s' \text{ is a } \tau\text{-successor of } s \}$.

- **enabledness of $\tau$**
  - $\text{En}(\tau) \triangleq (\exists V') \rho_{\tau}(V, V')$.
  - $\tau$ is enabled in a state if $\text{En}(\tau)$ holds in that state.
  - $\tau$ is enabled in state $s$ iff $s$ has some $\tau$-successor.
Computations of an FTS

Given an FTS $\mathcal{P} = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$, a computation of $\mathcal{P}$ is an infinite sequence of states $\sigma: s_0, s_1, s_2, \ldots$ satisfying:

- **Initiation:** $s_0$ is an initial state, i.e., $s_0 \models \Theta$.

- **Consecution:** for every $i \geq 0$, $s_{i+1}$ is a $\tau$-successor of state $s_i$, i.e., $\langle s_i, s_{i+1} \rangle \models \rho_\tau(V, V')$, for some $\tau \in \mathcal{T}$. In this case, we say that $\tau$ is taken at position $i$.

- **Justice:** for every $\tau \in \mathcal{J}$, it is never the case that $\tau$ is continuously enabled, but never taken, from some point on.

- **Compassion:** for every $\tau \in \mathcal{C}$, it is never the case that $\tau$ is enabled infinitely often, but never taken, from some point on.

The set of all computations of $\mathcal{P}$ is denoted by $\text{Comp}(\mathcal{P})$. 
Program ANY-Y:

\[ x, y : \text{natural initially } x = y = 0 \]

\[
\begin{align*}
& l_0 : \textbf{while } x = 0 \textbf{ do} \\
& \quad \text{[ } l_1 : \ y := y + 1; \ \text{]} \\
& l_2 : \\
\end{align*}
\]

\[ \parallel \]

\[
\begin{align*}
& m_0 : x := 1 \\
& m_2 : \\
\end{align*}
\]

Informal description:

☀ The program consists of an \textit{asynchronous composition} of two processes.

☀ One process continuously increments \( y \) as long as it finds \( x \) to be 0, while the other simply sets \( x \) to 1 (when it gets its turn to execute).

☀ The executions of the program are all possible \textit{interleavings} of the steps of the individual processes.
Program ANY-Y as an FTS $\mathcal{P}_{\text{ANY-Y}} = \langle V, \Theta, \mathcal{T}, \mathcal{J}, C \rangle$:

- $V \triangleq \{ x, y : \text{natural}, \pi_0 : \{ l_0, l_1, l_2 \}, \pi_1 : \{ m_0, m_1 \} \}$
- $\Theta \triangleq \pi_0 = l_0 \land \pi_1 = m_0 \land x = y = 0$
- $\mathcal{T} \triangleq \{ \tau_I, \tau_{l_0}, \tau_{l_1}, \tau_{m_0} \}$, whose transition relations are
  
  \[
  \rho_I : \quad \pi'_0 = \pi_0 \land \pi'_1 = \pi_1 \land x' = x \land y' = y
  \]
  
  \[
  \rho_{l_0} : \quad \pi_0 = l_0 \land ((x = 0 \land \pi'_0 = l_1) \lor (x \neq 0 \land \pi'_0 = l_2)) \land \pi'_1 = \pi_1 \land x' = x \land y' = y
  \]

  etc.

- $\mathcal{J} \triangleq \{ \tau_{l_0}, \tau_{l_1}, \tau_{m_0} \}$

- $C \triangleq \emptyset$
Program Mux

\[ Q_0, Q_1 : \text{bool initially } Q_0 = Q_1 = false \]
\[ T : \{0, 1\} \text{ initially } T = 0 \]

\[ P_0 :: \]
\[ l_0 : \text{loop forever do} \]
\[ l_1 : \text{remainder;} \]
\[ l_2 : Q_0 := true; \]
\[ l_3 : T := 0; \]
\[ l_4 : \text{await } \neg Q_1 \lor T \neq 0; \]
\[ l_5 : \text{critical;} \]
\[ l_6 : Q_0 := false; \]

\[ P_1 :: \]
\[ m_0 : \text{loop forever do} \]
\[ m_1 : \text{remainder;} \]
\[ m_2 : Q_1 := true; \]
\[ m_3 : T := 1; \]
\[ m_4 : \text{await } \neg Q_0 \lor T \neq 1; \]
\[ m_5 : \text{critical;} \]
\[ m_6 : Q_1 := false; \]

Justice is sufficient in preventing individual starvation.
Strong Fairness (Compassion) Is Needed

Program MUX-SEM: mutual exclusion by a semaphore.

\[ s : \text{natural initially } s = 1 \]

\[
\begin{align*}
    l_0 : & \text{ loop forever do} \\
    l_1 : & \text{ remainder;} \\
    l_2 : & \text{ request}(s); \\
    l_3 : & \text{ critical;} \\
    l_4 : & \text{ release}(s); \\
\end{align*}
\]

\[
\begin{align*}
    m_0 : & \text{ loop forever do} \\
    m_1 : & \text{ remainder;} \\
    m_2 : & \text{ request}(s); \\
    m_3 : & \text{ critical;} \\
    m_4 : & \text{ release}(s); \\
\end{align*}
\]

\[ \text{request}(s) \triangleq \langle \text{await } s > 0 : s := s - 1 \rangle \]

\[ \text{release}(s) \triangleq s := s + 1 \]

\[ C : \{ \tau_{l_2}, \tau_{m_2} \} \]
Linear Temporal Logic (LTL)

环球 State formulae
Constructed from the underlying assertion language

环球 Temporal formulae

☀ All state formulae are also temporal formulae.

☀ If \( p \) and \( q \) are temporal formulae and \( x \) a variable in \( \forall \), then the following are temporal formulae:

- \( \neg p, p \lor q, p \land q, p \rightarrow q, p \leftrightarrow q \)
- \( \Diamond p, \Box p, p \mathcal{U} q, p \mathcal{W} q \)
- \( \Diamond, \Box p, \Diamond p, p \mathcal{S} q, p \mathcal{B} q \)
- \( \exists x : p, \forall x : p \)
Semantics of LTL

- Temporal formulae are interpreted over an infinite sequence of states, called a model, with respect to a position in that sequence.

- We will define the satisfaction relation \((\sigma, i) \models \varphi\) (or \(\varphi\) holds in \((\sigma, i)\)), as the formal semantics of a temporal formula \(\varphi\) over an infinite sequence of states \(\sigma = s_0, s_1, s_2, \ldots, s_i, \ldots\) and a position \(i \geq 0\).

- A sequence \(\sigma\) satisfies a temporal formula \(\varphi\), denoted \(\sigma \models \varphi\), if \((\sigma, 0) \models \varphi\).

- Variables in \(\nu\) are partitioned into flexible and rigid variables. A flexible variable may assume different values in different states, while a rigid variable must assume the same value in all states of a model.
Semantics of LTL (cont.)

For a state formula $p$:

$$(\sigma, i) \models p \iff p \text{ holds at } s_i.$$  

Boolean combinations of formulae:

$$(\sigma, i) \models \neg p \iff (\sigma, i) \models p \text{ does not hold.}$$

$$(\sigma, i) \models p \lor q \iff (\sigma, i) \models p \text{ or } (\sigma, i) \models q.$$ 

$$(\sigma, i) \models p \land q \iff (\sigma, i) \models p \text{ and } (\sigma, i) \models q.$$ 

$$(\sigma, i) \models p \rightarrow q \iff (\sigma, i) \models p \text{ implies } (\sigma, i) \models q.$$ 

$$(\sigma, i) \models p \leftrightarrow q \iff (\sigma, i) \models p \text{ if and only if } (\sigma, i) \models q.$$  

Alternatively, the latter three cases can be defined in terms of $\neg$ and $\lor$, namely $p \land q \triangleq \neg(\neg p \lor \neg q),$ 

$p \rightarrow q \triangleq \neg p \lor q,$ and $p \leftrightarrow q \triangleq (p \rightarrow q) \land (q \rightarrow p).$
Semantics of LTL: Future Operators

- **□p (next p):**
  \[(\sigma, i) \models □p \iff (\sigma, i + 1) \models p.\]

- **◇p (eventually p or sometime p):**
  \[(\sigma, i) \models ◇p \iff \text{for some } k \geq i, (\sigma, k) \models p.\]

- **□p (henceforth p or always p):**
  \[(\sigma, i) \models □p \iff \text{for every } k \geq i, (\sigma, k) \models p.\]

- **p U q (p until q):**
  \[(\sigma, i) \models p U q \iff \text{for some } k \geq i, (\sigma, k) \models q \text{ and for every } j \text{ s.t. } i \leq j < k, (\sigma, j) \models p.\]

- **p W q (p wait-for q):**
  \[(\sigma, i) \models p W q \iff \text{for every } k \geq i, (\sigma, k) \models p, \text{ or for some } k \geq i, (\sigma, k) \models q \text{ and for every } j, i \leq j < k, (\sigma, j) \models p.\]
It can be shown that, for every $\sigma$ and $i$,

- $$(\sigma, i) \models \Diamond p \text{ iff } (\sigma, i) \models true \cup p$$
- $$(\sigma, i) \models \Box p \text{ iff } (\sigma, i) \models \neg \Diamond \neg p$$
- $$(\sigma, i) \models p \forall q \text{ iff } (\sigma, i) \models \Box p \lor p \cup q$$

So, one can also take $\circ$ and $\cup$ as the primitive operators and define others in terms of $\circ$ and $\cup$:

- $\Diamond p \triangleq true \cup p$
- $\Box p \triangleq \neg \Diamond \neg p$
- $p \forall q \triangleq \Box p \lor p \cup q$
Semantics of LTL: Past Operators

- $\Theta p$ (previous $p$):
  \[(\sigma, i) \models \Theta p \iff (i > 0) \text{ and } (\sigma, i - 1) \models p.\]

- $\bowtie p$ (before $p$):
  \[(\sigma, i) \models \bowtie p \iff (i > 0) \text{ implies } (\sigma, i - 1) \models p.\]

- $\lozenge p$ (once $p$):
  \[(\sigma, i) \models \lozenge p \iff \text{ for some } k, 0 \leq k \leq i, (\sigma, k) \models p.\]

- $\square p$ (so-far $p$):
  \[(\sigma, i) \models \square p \iff \text{ for every } k, 0 \leq k \leq i, (\sigma, k) \models p.\]

- $p S q$ (p since q):
  \[(\sigma, i) \models p S q \iff \text{ for some } k, 0 \leq k \leq i, (\sigma, k) \models q \text{ and for every } j, k < j \leq i, (\sigma, j) \models p.\]
Semantics of LTL: Past Operators (cont.)

$p \mathcal{B} q$ (p back-to q):
$(\sigma, i) \models p \mathcal{B} q \iff$ for every $k$, $0 \leq k \leq i$, $(\sigma, k) \models p$, or for some $k$, $0 \leq k \leq i$, $(\sigma, k) \models q$ and for every $j$, $k < j \leq i$, $(\sigma, j) \models p$. 
It can be shown that, for every $\sigma$ and $i$,

- $(\sigma, i) \models \Diamond p$ iff $(\sigma, i) \models \neg \Diamond \neg p$
- $(\sigma, i) \models \lozenge p$ iff $(\sigma, i) \models \text{true } S p$
- $(\sigma, i) \models \Box p$ iff $(\sigma, i) \models \neg \lozenge \neg p$
- $(\sigma, i) \models p \mathcal{B} q$ iff $(\sigma, i) \models \Box p \lor p S q$

So, one can also take $\Diamond$ and $S$ as the primitive operators and define others in terms of $\Diamond$ and $S$:

- $\Diamond p \triangleq \neg \Diamond \neg p$
- $\lozenge p \triangleq \text{true } S p$
- $\Box p \triangleq \neg \lozenge \neg p$
- $p \mathcal{B} q \triangleq \Box p \lor p S q$
Semantics of LTL: Quantifiers

A sequence $\sigma'$ is called a $u$-variant of $\sigma$ if $\sigma'$ differs from $\sigma$ in at most the interpretation given to $u$ in each state.

- $\sigma', i) \models \exists u : \varphi \iff (\sigma', i) \models \varphi$ for some $u$-variant $\sigma'$ of $\sigma$.
- $\sigma, i) \models \forall u : \varphi \iff (\sigma', i) \models \varphi$ for every $u$-variant $\sigma'$ of $\sigma$.

Alternatively, $\forall u : \varphi \equiv \neg(\exists u : \neg \varphi)$.

These definitions apply to both flexible and rigid variables.
Some LTL Conventions

Let $\textit{first}$ abbreviate $\Diamond \neg \textit{false}$, which holds only at position 0; $\textit{first}$ means “this is the first state”.

We use $u^-$ to denote the previous value of $u$; by convention, $u^-$ equals $u$ at position 0.

Example: $x = x^- + 1$.

In pure LTL,

$$(\textit{first} \land x = x + 1) \lor (\neg \textit{first} \land \forall u: \Diamond(x = u) \rightarrow x = u + 1).$$

We use $u^+$ (or $u'$) to denote the next value of $u$, i.e., the value of $u$ at the next position.

Example: $x^+ = x + 1$.

In pure LTL, $\forall u: x = u \rightarrow \Box (x = u + 1)$.

These previous and next-value notations also apply to expressions.
Validity

- A state formula is *state valid* if it holds in every state.
- A temporal formula $p$ is (temporally) *valid*, denoted $\models p$, if it holds in every model.
- A state formula is *$P$-state valid* if it holds in every $P$-accessible state (i.e., every state that appears in some computation of $P$).
- A temporal formula $p$ is *$P$-valid*, denoted $P \models p$, if it holds in every computation of $P$. 
Equivalence and Congruence

- Two formulae $p$ and $q$ are **equivalent** if $p \leftrightarrow q$ is valid.
  Example: $p \forall q \leftrightarrow \Box(\Diamond \neg p \rightarrow \Diamond q)$.

- Two formulae $p$ and $q$ are **congruent** if $\Box(p \leftrightarrow q)$ is valid.
  Example: $\neg \Diamond p$ and $\Box \neg p$ are congruent, as $\Box(\neg \Diamond p \leftrightarrow \Box \neg p)$ is valid.

- Two congruent formulae may replace each other in any context.
A Hierarchy of Temporal Properties

- Classes of temporal properties; \( p, q, p_i, q_i \) below are arbitrary past temporal formulae
  - Safety properties: \( \Box p \)
  - Guarantee properties: \( \Diamond p \)
  - Obligation properties: \( \bigwedge_{i=1}^{n}(\Box p_i \lor \Diamond q_i) \)
  - Response properties: \( \Box \Diamond p \)
  - Persistence properties: \( \Diamond \Box p \)
  - Reactivity properties: \( \bigwedge_{i=1}^{n}(\Box \Diamond p_i \lor \Diamond \Box q_i) \)

- The hierarchy

  Safety  \( \subseteq \)  Obligation  \( \subseteq \)  Response  \( \subseteq \)  Reactivity

- Every temporal formula is equivalent to some reactivity formula.
More Common Temporal Properties

- **Safety properties:** □p
  Example: p ⊨ q is a safety property, as it is equivalent to □(!(⊥p → ⊨q).

- **Response properties**
  - **Canonical form:** □◊p
  - **Variant:** □ ⊨ (p → ◊q) (p leads-to q), which is equivalent to □◊(¬p ⊨ q).

- **Reactivity properties:** \( \bigwedge_{i=1}^{n} (□◊p_i \lor ◊□q_i) \)

- **(Simple) reactivity properties**
  - **Canonical form:** □◊p \lor ◊□q
  - **Variants:** □◊p → □◊q or □(□◊p → ◊q), which is equivalent to □◊q \lor ◊□¬p.
  - **Extended form:** □((p \land □◊r) → ◊q)
Rules for Safety Properties

Rule INV

I1. $\Theta \rightarrow \varphi$
I2. $\varphi \rightarrow q$
I3. $\{\varphi\} \mathcal{T} \{\varphi\}$

$\square q$

where $\{p\} \mathcal{T} \{q\}$ means $\{p\} \tau \{q\}$ (i.e., $\rho_\tau \land p \rightarrow q'$) for every $\tau \in \mathcal{T}$

The auxiliary assertion $\varphi$ is called an inductive invariant, as it holds initially and is preserved by every transition.

This rule is sound and (relatively) complete for establishing $P$-validity of the future safety formula $\square q$ (where $q$ is a state formula).
Mutual exclusion: $\square(\neg(\pi_0 = l_3 \land \pi_1 = m_3))$, which is not inductive.

The inductive $\varphi$ needed:

$$y \geq 0 \land (\pi_0 = l_3) + (\pi_0 = l_4) + (\pi_1 = m_3) + (\pi_1 = m_4) + y = 1$$

where true and false are equated respectively with 1 and 0.
Rules for Response Properties

Rule J-RESP (for a just transition $\tau \in \mathcal{J}$)

J1. $\Box(p \rightarrow (q \lor \varphi))$
J2. $\{\varphi\} \mathcal{T} \{q \lor \varphi\}$
J3. $\{\varphi\} \tau \{q\}$
J4. $\Box(\varphi \rightarrow (q \lor En(\tau)))$

\[
\Box(p \rightarrow \Diamond q)
\]

This is a “one-step” rule that relies on a helpful just transition.
Analogously, there is a one-step rule that relies on a helpful compassionate transition.

Rule C-RESP (for a compassionate transition $\tau \in C$)

C1. $\Box(p \rightarrow (q \lor \varphi))$

C2. $\{\varphi\} T \{q \lor \varphi\}$

C3. $\{\varphi\} \tau \{q\}$

C4. $T - \{\tau\} \vdash \Box(\varphi \rightarrow \Diamond(q \lor En(\tau)))$

$\Box(p \rightarrow \Diamond q)$

Premise C4 states that the proof obligation should be carried out for a smaller program with $T - \{\tau\}$ as the set of transitions.
Rule M-RESP (monotonicity) and Rule T-RESP (transitivity)

\[ \Box(p \rightarrow r), \Box(t \rightarrow q) \]
\[ \Box(r \rightarrow \Diamond t) \]
\[ \Box(p \rightarrow \Diamond q) \]
\[ \Box(p \rightarrow \Diamond r) \]
\[ \Box(r \rightarrow \Diamond q) \]
\[ \Box(p \rightarrow \Diamond q) \]

These rules belong to the part for proving general temporal validity. They are convenient, but not necessary when we have a relatively complete rule that reduce program validity directly to assertional validity.
A *ranking function* maps finite sequences of states into a well-founded set.

Rule W-RESP (with a ranking function $\delta$)

1. $\Box(p \rightarrow (q \lor \varphi))$
2. $\Box([\varphi \land (\delta = \alpha)] \rightarrow \Diamond[q \lor (\varphi \land \delta \prec \alpha)])$
3. $\Box(p \rightarrow \Diamond q)$
Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$. $\varphi$ denotes $\varphi_1 \lor \varphi_2 \lor \cdots \lor \varphi_n$ and $\delta$ is a ranking function.

Rule F-RESP

F1. $\Box (p \rightarrow (q \lor \varphi))$

for $i = 1, \ldots, m$

F2. $\{\varphi_i \land (\delta = \alpha)\} \vdash \mathcal{T} \{q \lor (\varphi \land (\delta < \alpha)) \lor (\varphi_i \land (\delta \leq \alpha))\}$

F3. $\{\varphi_i \land (\delta = \alpha)\} \vdash \tau_i \{q \lor (\varphi \land (\delta < \alpha))\}$

J4. $\Box (\varphi_i \rightarrow (q \lor En(\tau_i)))$, if $\tau_i \in \mathcal{J}$

C4. $\mathcal{T} - \{\tau_i\} \vdash \Box (\varphi_i \rightarrow \Diamond (q \lor En(\tau_i)))$, if $\tau_i \in \mathcal{C}$

$\Box (p \rightarrow \Diamond q)$

Rule F-RESP is (relatively) complete for proving the $P$-validity of any response formula of the form $\Box (p \rightarrow \Diamond q)$. 
Rule B-REAC

B1. $\Box(p \rightarrow (q \lor \varphi))$

B2. $\{\varphi \land (\delta = \alpha)\} \mathcal{T} \{q \lor (\varphi \land (\delta \preceq \alpha))\}$

B3. $\Box([\varphi \land (\delta = \alpha) \land r] \rightarrow \Diamond[q \lor (\delta < \alpha)])$

$\Box((p \land \Box\Diamond r) \rightarrow \Diamond q)$

For programs without compassionate transitions, Rule B-REAC is (relatively) complete for proving the $\mathcal{P}$-validity of any (simple, extended) reactivity formula of the form $\Box((p \land \Box\Diamond r) \rightarrow \Diamond q)$. 
An FDS $\mathcal{D}$ is a tuple $\langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$:

- $V \subseteq \mathcal{V}$: A finite set of typed state variables, containing data and control variables.

- $\Theta$: The initial condition, an assertion characterizing the initial states.

- $\rho$: The transition relation, an assertion relating the values of the state variables in a state to the values in the next state.

- $\mathcal{J} = \{ J_1, \cdots, J_k \}$: A set of justice requirements (weak fairness).

- $\mathcal{C} = \{ \langle p_1, q_1 \rangle, \cdots, \langle p_n, q_n \rangle \}$: A set of compassion requirements (strong fairness).
So, FDS is a slight variation of the model of fair transition system.

The main difference between the FDS and FTS models is in the representation of fairness constraints.

FDS enables a unified representation of fairness constraints arising from both the system being verified, and the temporal property.

A computation of $\mathcal{D}$ is an infinite sequence of states $\sigma = s_0, s_1, s_2, \cdots$ satisfying *Initiation*, *Consecution*, *Justice*, and *Compassion* conditions.
Program **Mux-Sem** as an FDS

Program **Mux-Sem**: mutual exclusion by a semaphore.

\[ s : \text{natural initially } s = 1 \]

\[
\begin{align*}
\begin{array}{l}
l_0 : \text{loop forever do} \\
l_1 : \text{remainder;} \\
l_2 : \text{request}(s); \\
l_3 : \text{critical;} \\
l_4 : \text{release}(s);
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{l}
m_0 : \text{loop forever do} \\
m_1 : \text{remainder;} \\
m_2 : \text{request}(s); \\
m_3 : \text{critical;} \\
m_4 : \text{release}(s);
\end{array}
\end{align*}
\]

\[
\begin{align*}
\odot \text{ request}(s) & \overset{\Delta}{=} \langle \text{await } s > 0 : s := s - 1 \rangle \\
\odot \text{ release}(s) & \overset{\Delta}{=} s := s + 1
\end{align*}
\]

\[ C : \{ (at_{l2} \land s > 0, at_{l3}), (at_{m2} \land s > 0, at_{m3}) \} \]
References


