

# Temporal Verification of Reactive Systems

(Based on Manna and Pnueli [1991,1995,1996])

Yih-Kuen Tsay

Department 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
  - 👤  $\{n \geq 0\} y := ? \{y = n^2\}$  or
  - 👤  $y : [n \geq 0, y = n^2]$

## 🌐 Reactive Programs

- ☀️ Maintaining an ongoing (typically not terminating) interaction with their environments
- ☀️ An example:  $s : \{0, 1\}$  **initially**  $s = 1$




$$\left[ \begin{array}{l} l_0 : \text{loop forever do} \\ \left[ \begin{array}{l} l_1 : \text{remainder;} \\ l_2 : \text{request}(s); \\ l_3 : \text{critical;} \\ l_4 : \text{release}(s); \end{array} \right] \end{array} \right] \parallel \left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{remainder;} \\ m_2 : \text{request}(s); \\ m_3 : \text{critical;} \\ m_4 : \text{release}(s); \end{array} \right] \end{array} \right]$$

- ☀️ Must be specified and verified in terms of their behaviors, including the intermediate states

# 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






- 🌐 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  $\mathcal{V}$ .
- 🌐 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

# Fair Transition Systems

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:  $x = 1 \wedge x' = 0$ .

For  $s, s' \in \Sigma_V$ ,  $\langle s, s' \rangle \models x = 1 \wedge x' = 0$  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

☀ State  $s'$  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  $\tau$

☀  $En(\tau) \triangleq (\exists V') \rho_\tau(V, V')$ .

☀  $\tau$  is enabled in a state if  $En(\tau)$  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  $\sigma : s_0, s_1, s_2, \dots$  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  $Comp(\mathcal{P})$ .

# An Example Program and Its FTS

🌐 Program ANY-Y:

$x, y$  : natural **initially**  $x = y = 0$

$$\left[ \begin{array}{l} l_0 : \mathbf{while} \ x = 0 \ \mathbf{do} \\ \quad \left[ \begin{array}{l} l_1 : \ y := y + 1; \\ l_2 : \end{array} \right] \end{array} \right] \parallel \left[ \begin{array}{l} m_0 : \ x := 1 \\ m_2 : \end{array} \right]$$

🌐 Informal description:

- ☀ The program consists of an *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 *interleavings* of the steps of the individual processes.

# An Example Program and Its FTS (cont.)

🌐 Program ANY-Y as an FTS  $\mathcal{P}_{\text{ANY-Y}} = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ :

- ☀  $V \triangleq \{x, y : \text{natural}, \pi_0 : \{l_0, l_1, l_2\}, \pi_1 : \{m_0, m_1\}\}$
- ☀  $\Theta \triangleq \pi_0 = l_0 \wedge \pi_1 = m_0 \wedge x = y = 0$
- ☀  $\mathcal{T} \triangleq \{\tau_l, \tau_{l_0}, \tau_{l_1}, \tau_{m_0}\}$ , whose transition relations are
  - $\rho_l : \pi'_0 = \pi_0 \wedge \pi'_1 = \pi_1 \wedge x' = x \wedge y' = y$ ,
  - $\rho_{l_0} : \pi_0 = l_0 \wedge ((x = 0 \wedge \pi'_0 = l_1) \vee (x \neq 0 \wedge \pi'_0 = l_2))$   
 $\wedge \pi'_1 = \pi_1 \wedge x' = x \wedge y' = y$ , etc.
- ☀  $\mathcal{J} \triangleq \{\tau_{l_0}, \tau_{l_1}, \tau_{m_0}\}$
- ☀  $\mathcal{C} \triangleq \emptyset$

# Program Mux

$Q_0, Q_1 : \text{bool}$  **initially**  $Q_0 = Q_1 = \text{false}$   
 $T : \{0, 1\}$  **initially**  $T = 0$

$P_0 ::$ $\left[ \begin{array}{l} l_0 : \text{loop forever do} \\ \left[ \begin{array}{l} l_1 : \text{remainder;} \\ l_2 : Q_0 := \text{true;} \\ l_3 : T := 0; \\ l_4 : \text{await } \neg Q_1 \vee T \neq 0; \\ l_5 : \text{critical;} \\ l_6 : Q_0 := \text{false;} \end{array} \right] \end{array} \right]$	$\parallel$	$P_1 ::$ $\left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{remainder;} \\ m_2 : Q_1 := \text{true;} \\ m_3 : T := 1; \\ m_4 : \text{await } \neg Q_0 \vee T \neq 1; \\ m_5 : \text{critical;} \\ m_6 : Q_1 := \text{false;} \end{array} \right] \end{array} \right]$
--	-------------	--

Justice is sufficient in preventing individual starvation.

# Strong Fairness (Compassion) Is Needed

🌐 Program MUX-SEM: mutual exclusion by a semaphore.

$s$  : natural **initially**  $s = 1$

$$\left[ \begin{array}{l} l_0 : \text{loop forever do} \\ \left[ \begin{array}{l} l_1 : \text{remainder;} \\ l_2 : \text{request}(s); \\ l_3 : \text{critical;} \\ l_4 : \text{release}(s); \end{array} \right] \end{array} \right] \parallel \left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{remainder;} \\ m_2 : \text{request}(s); \\ m_3 : \text{critical;} \\ m_4 : \text{release}(s); \end{array} \right] \end{array} \right]$$

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

☀️  $\text{release}(s) \triangleq s := s + 1$

🌐  $\mathcal{C}: \{\tau_{l_2}, \tau_{m_2}\}$

# Linear Temporal Logic (LTL)

🌐 State formulae  
Constructed from the underlying assertion language

🌐 Temporal formulae

☀️ All state formulae are also temporal formulae.

☀️ If  $p$  and  $q$  are temporal formulae and  $x$  a variable in  $\mathcal{V}$ , then the following are temporal formulae:

👤  $\neg p, p \vee q, p \wedge q, p \rightarrow q, p \leftrightarrow q$

👤  $\bigcirc p, \diamond p, \square p, p \mathcal{U} q, p \mathcal{W} q$

👤  $\ominus p, \odot p, \blacklozenge p, \boxplus p, p \mathcal{S} q, p \mathcal{B} q$

👤  $\exists x: p, \forall x: p$

# 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  $\sigma = s_0, s_1, s_2, \dots, s_i, \dots$  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  $\mathcal{V}$  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:

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

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






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

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

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

Alternatively, the latter three cases can be defined in terms of  $\neg$  and  $\vee$ , namely  $p \wedge q \triangleq \neg(\neg p \vee \neg q)$ ,  $p \rightarrow q \triangleq \neg p \vee q$ , and  $p \leftrightarrow q \triangleq (p \rightarrow q) \wedge (q \rightarrow p)$ .

# Semantics of LTL: Future Operators

-   $\circ p$  (next  $p$ ):  
 $(\sigma, i) \models \circ p \iff (\sigma, i + 1) \models p.$
-   $\diamond p$  (eventually  $p$  or sometime  $p$ ):  
 $(\sigma, i) \models \diamond p \iff \text{for some } k \geq i, (\sigma, k) \models p.$
-   $\square p$  (henceforth  $p$  or always  $p$ ):  
 $(\sigma, i) \models \square p \iff \text{for every } k \geq i, (\sigma, k) \models p.$
-   $p \mathcal{U} q$  ( $p$  until  $q$ ):  
 $(\sigma, i) \models p \mathcal{U} q \iff \text{for some } k \geq i, (\sigma, k) \models q \text{ and for every } j$   
 $\text{s.t. } i \leq j < k, (\sigma, j) \models p.$
-   $p \mathcal{W} q$  ( $p$  wait-for  $q$ ):  
 $(\sigma, i) \models p \mathcal{W} q \iff \text{for every } k \geq i, (\sigma, k) \models p, \text{ or for some}$   
 $k \geq i, (\sigma, k) \models q \text{ and for every } j, i \leq j < k, (\sigma, j) \models p.$

🌐 It can be shown that, for every  $\sigma$  and  $i$ ,

☀️  $(\sigma, i) \models \diamond p$  iff  $(\sigma, i) \models \text{true } \mathcal{U} p$

☀️  $(\sigma, i) \models \square p$  iff  $(\sigma, i) \models \neg \diamond \neg p$

☀️  $(\sigma, i) \models p \mathcal{W} q$  iff  $(\sigma, i) \models \square p \vee p \mathcal{U} q$






🌐 So, one can also take  $\circ$  and  $\mathcal{U}$  as the primitive operators and define others in terms of  $\circ$  and  $\mathcal{U}$ :

☀️  $\diamond p \stackrel{\Delta}{=} \text{true } \mathcal{U} p$

☀️  $\square p \stackrel{\Delta}{=} \neg \diamond \neg p$

☀️  $p \mathcal{W} q \stackrel{\Delta}{=} \square p \vee p \mathcal{U} q$

# Semantics of LTL: Past Operators

-   $\ominus p$  (previous  $p$ ):  
 $(\sigma, i) \models \ominus p \iff (i > 0) \text{ and } (\sigma, i - 1) \models p.$
-   $\odot p$  (before  $p$ ):  
 $(\sigma, i) \models \odot p \iff (i > 0) \text{ implies } (\sigma, i - 1) \models p.$
-   $\diamond p$  (once  $p$ ):  
 $(\sigma, i) \models \diamond p \iff \text{for some } k, 0 \leq k \leq i, (\sigma, k) \models p.$
-   $\boxplus p$  (so-far  $p$ ):  
 $(\sigma, i) \models \boxplus p \iff \text{for every } k, 0 \leq k \leq i, (\sigma, k) \models p.$
-   $p \mathcal{S} q$  ( $p$  since  $q$ ):  
 $(\sigma, i) \models p \mathcal{S} q \iff \text{for some } k, 0 \leq k \leq i, (\sigma, k) \models q \text{ and for every } j, k < j \leq i, (\sigma, j) \models p.$

🌐  $p \mathcal{B} q$  ( $p$  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  $j, k < j \leq i, (\sigma, j) \models p$ .

# Semantics of LTL: Past Operators (cont.)

It can be shown that, for every  $\sigma$  and  $i$ ,

$$\odot (\sigma, i) \models \ominus p \text{ iff } (\sigma, i) \models \neg \odot \neg p$$

$$\odot (\sigma, i) \models \diamond p \text{ iff } (\sigma, i) \models \text{true } \mathcal{S} p$$

$$\odot (\sigma, i) \models \Box p \text{ iff } (\sigma, i) \models \neg \diamond \neg p$$

$$\odot (\sigma, i) \models p \mathcal{B} q \text{ iff } (\sigma, i) \models \Box p \vee p \mathcal{S} q$$

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

$$\odot \ominus p \stackrel{\Delta}{=} \neg \odot \neg p$$

$$\odot \diamond p \stackrel{\Delta}{=} \text{true } \mathcal{S} p$$

$$\odot \Box p \stackrel{\Delta}{=} \neg \odot \neg p$$

$$\odot p \mathcal{B} q \stackrel{\Delta}{=} \Box p \vee p \mathcal{S} q$$

# 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.

$$\text{🌐 } (\sigma, i) \models \exists u: \varphi \iff (\sigma', i) \models \varphi \text{ for some } u\text{-variant } \sigma' \text{ of } \sigma.$$

$$\text{🌐 } (\sigma, i) \models \forall u: \varphi \iff (\sigma', i) \models \varphi \text{ for every } u\text{-variant } \sigma' \text{ of } \sigma.$$

$$\text{Alternatively, } \forall u: \varphi \stackrel{\Delta}{=} \neg(\exists u: \neg\varphi).$$

These definitions apply to both flexible and rigid variables.

# Some LTL Conventions

- 🌐 Let *first* abbreviate  $\odot \text{false}$ , which holds only at position 0; *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,
 
$$(\text{first} \wedge x = x + 1) \vee (\neg \text{first} \wedge \forall u: \odot(x = u) \rightarrow x = u + 1).$$
- 🌐 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,  $\forall u: x = u \rightarrow \bigcirc(x = u + 1)$ .
- 🌐 These previous and next-value notations also apply to expressions.



- 🌐 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:  $p \mathcal{W} q \leftrightarrow \Box(\Diamond\neg p \rightarrow \Diamond q)$ .
- 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:  $\bigwedge_{i=1}^n (\Box p_i \vee \Diamond q_i)$

☀ Response properties:  $\Box \Diamond p$

☀ Persistence properties:  $\Diamond \Box p$

☀ Reactivity properties:  $\bigwedge_{i=1}^n (\Box \Diamond p_i \vee \Diamond \Box q_i)$

🌐 The hierarchy

Safety  
Guarantee  $\subseteq$  Obligation  $\subseteq$  Response  
Persistence  $\subseteq$  Reactivity

🌐 Every temporal formula is equivalent to some reactivity formula.

# More Common Temporal Properties

🌐 Safety properties:  $\Box p$

Example:  $p \mathcal{W} q$  is a safety property, as it is equivalent to  $\Box(\Diamond \neg p \rightarrow \Diamond q)$ .

🌐 Response properties

☀ Canonical form:  $\Box \Diamond p$

☀ Variant:  $\Box(p \rightarrow \Diamond q)$  ( $p$  leads-to  $q$ ), which is equivalent to  $\Box \Diamond(\neg p \mathcal{B} q)$ .

🌐 Reactivity properties:  $\bigwedge_{i=1}^n (\Box \Diamond p_i \vee \Diamond \Box q_i)$

🌐 (Simple) reactivity properties

☀ Canonical form:  $\Box \Diamond p \vee \Diamond \Box q$

☀ Variants:  $\Box \Diamond p \rightarrow \Box \Diamond q$  or  $\Box(\Box \Diamond p \rightarrow \Diamond q)$ , which is equivalent to  $\Box \Diamond q \vee \Diamond \Box \neg p$ .

☀ Extended form:  $\Box((p \wedge \Box \Diamond r) \rightarrow \Diamond q)$

# Rules for Safety Properties

## Rule INV

$$\begin{array}{l}
 \text{I1. } \Theta \rightarrow \varphi \\
 \text{I2. } \varphi \rightarrow q \\
 \text{I3. } \frac{\{\varphi\} \mathcal{T} \{\varphi\}}{\Box q}
 \end{array}$$

where  $\{p\} \mathcal{T} \{q\}$  means  $\{p\} \tau \{q\}$  (i.e.,  $\rho_\tau \wedge p \rightarrow q'$ ) 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  $\Box q$  (where  $q$  is a state formula).

# A Safety Property of Program Mux-Sem

- 🌐 Mutual exclusion:  $\Box(\neg(\pi_0 = l_3 \wedge \pi_1 = m_3))$ , which is not inductive.
- 🌐 The inductive  $\varphi$  needed:

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

where *true* and *false* are equated respectively with 1 and 0.

# Rules for Response Properties

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

$$\begin{array}{l} \text{J1. } \Box(p \rightarrow (q \vee \varphi)) \\ \text{J2. } \{\varphi\} \mathcal{T} \{q \vee \varphi\} \\ \text{J3. } \{\varphi\} \tau \{q\} \\ \text{J4. } \Box(\varphi \rightarrow (q \vee \text{En}(\tau))) \\ \hline \Box(p \rightarrow \Diamond q) \end{array}$$

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

## Rules for Response Properties (cont.)

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

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

$$\frac{\begin{array}{l} \text{C1. } \Box(p \rightarrow (q \vee \varphi)) \\ \text{C2. } \{\varphi\} \mathcal{T} \{q \vee \varphi\} \\ \text{C3. } \{\varphi\} \tau \{q\} \\ \text{C4. } \mathcal{T} - \{\tau\} \vdash \Box(\varphi \rightarrow \Diamond(q \vee \text{En}(\tau))) \end{array}}{\Box(p \rightarrow \Diamond q)}$$

Premise C4 states that the proof obligation should be carried out for a smaller program with  $\mathcal{T} - \{\tau\}$  as the set of transitions.



# Rules for Response Properties (cont.)

Rule M-RESP (monotonicity) and Rule T-RESP (transitivity)

$$\frac{\begin{array}{l} \Box(p \rightarrow r), \Box(t \rightarrow q) \\ \Box(r \rightarrow \Diamond t) \end{array}}{\Box(p \rightarrow \Diamond q)} \qquad \frac{\begin{array}{l} \Box(p \rightarrow \Diamond r) \\ \Box(r \rightarrow \Diamond q) \end{array}}{\Box(p \rightarrow \Diamond q)}$$

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.

# Rules for Response Properties (cont.)

A *ranking function* maps finite sequences of states into a well-founded set.

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

$$\frac{\begin{array}{l} \text{W1. } \Box(p \rightarrow (q \vee \varphi)) \\ \text{W2. } \Box([\varphi \wedge (\delta = \alpha)] \rightarrow \Diamond[q \vee (\varphi \wedge \delta \prec \alpha)]) \end{array}}{\Box(p \rightarrow \Diamond q)}$$

# Rules for Response Properties (cont.)

Let  $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$ .  $\varphi$  denotes  $\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n$  and  $\delta$  is a ranking function.

## Rule F-RESP

$$F1. \quad \Box(p \rightarrow (q \vee \varphi))$$

for  $i = 1, \dots, m$

$$F2. \quad \{\varphi_i \wedge (\delta = \alpha)\} \mathcal{T} \{q \vee (\varphi \wedge (\delta < \alpha)) \vee (\varphi_i \wedge (\delta \preceq \alpha))\}$$

$$F3. \quad \{\varphi_i \wedge (\delta = \alpha)\} \tau_i \{q \vee (\varphi \wedge (\delta < \alpha))\}$$

$$J4. \quad \Box(\varphi_i \rightarrow (q \vee \mathit{En}(\tau_i))), \text{ if } \tau_i \in \mathcal{J}$$

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

---


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

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

# Rules for Reactivity Properties

## Rule B-REAC

$$\begin{array}{l} \text{B1. } \Box(p \rightarrow (q \vee \varphi)) \\ \text{B2. } \{\varphi \wedge (\delta = \alpha)\} \mathcal{T} \{q \vee (\varphi \wedge (\delta \preceq \alpha))\} \\ \text{B3. } \Box([\varphi \wedge (\delta = \alpha) \wedge r] \rightarrow \Diamond[q \vee (\delta \prec \alpha)]) \\ \hline \Box((p \wedge \Box\Diamond r) \rightarrow \Diamond q) \end{array}$$

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  $\Box((p \wedge \Box\Diamond r) \rightarrow \Diamond q)$ .

# Fair Discrete Systems

- 🌐 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.
  - ☀️  $\mathcal{J} = \{J_1, \dots, J_k\}$ : A set of justice requirements (weak fairness).
  - ☀️  $\mathcal{C} = \{\langle p_1, q_1 \rangle, \dots, \langle p_n, q_n \rangle\}$ : 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  $\sigma = s_0, s_1, s_2, \dots$  satisfying *Initiation*, *Consecution*, *Justice*, and *Compassion* conditions.

# Program Mux-Sem as an FDS

🌐 Program MUX-SEM: mutual exclusion by a semaphore.

$s$  : natural **initially**  $s = 1$




$$\left[ \begin{array}{l} l_0 : \text{loop forever do} \\ \left[ \begin{array}{l} l_1 : \text{remainder;} \\ l_2 : \text{request}(s); \\ l_3 : \text{critical;} \\ l_4 : \text{release}(s); \end{array} \right] \end{array} \right] \parallel \left[ \begin{array}{l} m_0 : \text{loop forever do} \\ \left[ \begin{array}{l} m_1 : \text{remainder;} \\ m_2 : \text{request}(s); \\ m_3 : \text{critical;} \\ m_4 : \text{release}(s); \end{array} \right] \end{array} \right]$$

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

☀️  $\text{release}(s) \triangleq s := s + 1$

🌐  $\mathcal{C}: \{(at\_l_2 \wedge s > 0, at\_l_3), (at\_m_2 \wedge s > 0, at\_m_3)\}$

# References

-  Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1991.
-  Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*, Springer-Verlag, 1995.
-  Z. Manna and A. Pnueli. *Temporal Verification: Progress*, Manuscript, 1996.