Specifying Properties of Concurrent Systems

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

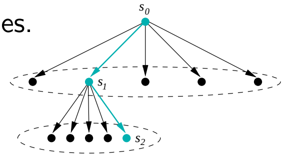
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at

1. **The Basics of Temporal Logic**

2. Specifying with Linear Time Logic

Motivation

We need a language for specifying system properties.



- A system S is a pair $\langle I, R \rangle$.
  - Initial states I, transition relation R.
  - More intuitive: reachability graph.
    - Starting from an initial state s_0, the system runs evolve.
- Consider the reachability graph as an infinite computation tree.
  - Different tree nodes may denote occurrences of the same state.
    - Each occurrence of a state has a unique predecessor in the tree.
  - Every path in this tree is infinite.
    - Every finite run $s_0 \rightarrow \ldots \rightarrow s_n$ is extended to an infinite run $s_0 \rightarrow \ldots \rightarrow s_n \rightarrow s_n \rightarrow s_n \rightarrow \ldots$
- Or simply consider the graph as a set of system runs.
  - Same state may occur multiple times (in one or in different runs).

Temporal logic describes such trees respectively sets of system runs.

Computation Trees versus System Runs

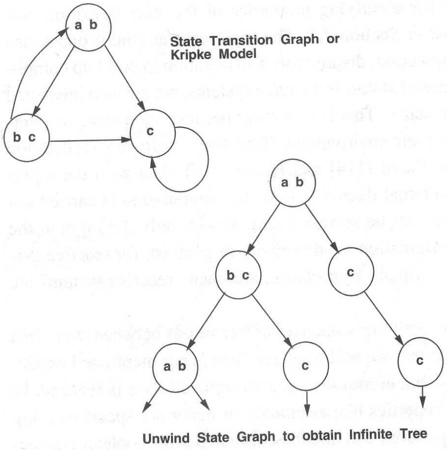


Figure 3.1
Computation trees.

Set of system runs:

$[a, b] \rightarrow c \rightarrow c \rightarrow \ldots$

$[a, b] \rightarrow [b, c] \rightarrow c \rightarrow \ldots$

$[a, b] \rightarrow [b, c] \rightarrow [a, b] \rightarrow \ldots$

$[a, b] \rightarrow [b, c] \rightarrow [a, b] \rightarrow \ldots$

$\ldots$

Edmund Clarke et al: "Model Checking", 1999.

State Formula

Temporal logic is based on classical logic.

- A state formula F is evaluated on a state s.
  - Any predicate logic formula is a state formula:
    $p(x), \neg F, F_0 \wedge F_1, F_0 \vee F_1, F_0 \Rightarrow F_1, F_0 \Leftrightarrow F_1, \forall x : F, \exists x : F$.
  - In propositional temporal logic only propositional logic formulas are state formulas (no quantification):
    $p, \neg F, F_0 \wedge F_1, F_0 \vee F_1, F_0 \Rightarrow F_1, F_0 \Leftrightarrow F_1$.
- Semantics: $s \models F$ ("F holds in state s").
  - Example: semantics of conjunction.
    - $(s \models F_0 \wedge F_1) \;:\Leftrightarrow\; (s \models F_0) \wedge (s \models F_1)$.
    - "$F_0 \wedge F_1$ holds in s if and only if F_0 holds in s and F_1 holds in s".

Classical logic reasons on individual states.

Temporal Logic

Extension of classical logic to reason about multiple states.

- Temporal logic is an instance of modal logic.
  - Logic of "multiple worlds (situations)" that are in some way related.
  - Relationship may e.g. be a temporal one.
  - Amir Pnueli, 1977: temporal logic is suited to system specifications.
  - Many variants, two fundamental classes.
- Branching Time Logic
  - Semantics defined over computation trees.

    At each moment, there are multiple possible futures.
  - Prominent variant: CTL.

    Computation tree logic; a propositional branching time logic.
- Linear Time Logic
  - Semantics defined over sets of system runs.

    At each moment, there is only one possible future.
  - Prominent variant: PLTL.

    A propositional linear time logic.

Branching Time Logic (CTL)

We use temporal logic to specify a system property F.

- Core question: $S \models F$ ("F holds in system S").
  - System $S = \langle I, R \rangle$, temporal logic formula F.
- Branching time logic:
  - $S \models F \;:\Leftrightarrow\; S, s_0 \models F$, for every initial state s_0 of S.
  - Property F must be evaluated on every pair of system S and initial state s_0.
  - Given a computation tree with root s_0, F is evaluated on that tree.

CTL formulas are evaluated on computation trees.

State Formulas

We have additional state formulas.

- A state formulas F is evaluated on state s of System S.
  - Every (classical) state formula f is such a state formula.
  - Let P denote a path formula (later).
    - Evaluated on a path (state sequence) $p = p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \ldots$.

      $R(p_i, p_{i+1})$ for every i; p_0 need not be an initial state.
  - Then the following are state formulas:

    A P ("in every path P"),
    E P ("in some path P").
  - Path quantifiers: **A**, **E**.
- Semantics: $S, s \models F$ ("F holds in state s of system S").

$$S, s \models f \;:\Leftrightarrow\; s \models f.$$
$$S, s \models \mathbf{A}\ P \;:\Leftrightarrow\; S, p \models P, \text{ for every path } p \text{ of } S \text{ with } p_0 = s.$$
$$S, s \models \mathbf{E}\ P \;:\Leftrightarrow\; S, p \models P, \text{ for some path } p \text{ of } S \text{ with } p_0 = s.$$

Path Formulas

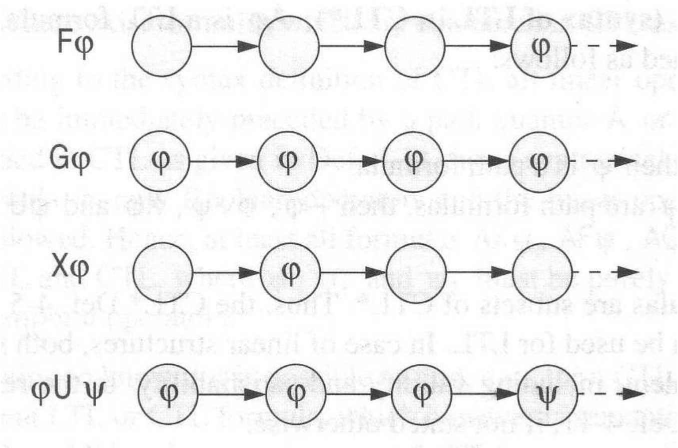
We have a class of formulas that are not evaluated over individual states.

- A path formula P is evaluated on a path p of system S.
  - Let F and G denote state formulas.
  - Then the following are path formulas:

    $\mathbf{X}\ F$ ("next time F"),
    $\mathbf{G}\ F$ ("always F"),
    $\mathbf{F}\ F$ ("eventually F"),
    $F\ \mathbf{U}\ G$ ("F until G").
  - Temporal operators: $\mathbf{X}, \mathbf{G}, \mathbf{F}, \mathbf{U}$.
- Semantics: $S, p \models P$ ("P holds in path p of system S").

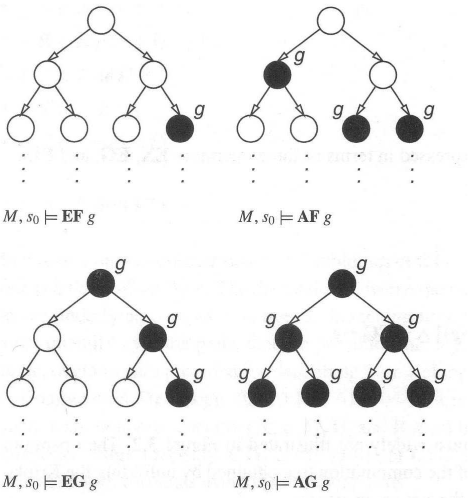
$$S, p \models \mathbf{X}\ F \ :\Leftrightarrow S, p_1 \models F.$$
$$S, p \models \mathbf{G}\ F \ :\Leftrightarrow \forall i \in \mathbb{N} : S, p_i \models F.$$
$$S, p \models \mathbf{F}\ F \ :\Leftrightarrow \exists i \in \mathbb{N} : S, p_i \models F.$$
$$S, p \models F\ \mathbf{U}\ G \ :\Leftrightarrow \exists i \in \mathbb{N} : S, p_i \models G \wedge \forall j \in \mathbb{N}_i : S, p_j \models F.$$

Path Formulas



Thomas Kropf: "Introduction to Formal Hardware Verification", 1999.

Path Quantifiers and Temporal Operators



Edmund Clarke et al: "Model Checking", 1999.

Linear Time Logic (LTL)

We use temporal logic to specify a system property P.

- Core question: $S \models P$ ("P holds in system S").
  - System $S = \langle I, R \rangle$, temporal logic formula P.
- Linear time logic:
  - $S \models P \;:\Leftrightarrow\; r \models P$, for every run r of S.
  - Property P must be evaluated on every run r of S.
  - Given a computation tree with root s_0, P is evaluated on every path of that tree originating in s_0.
    - If P holds for every path, P holds on S.

LTL formulas are evaluated on system runs.

Formulas

No path quantifiers; all formulas are path formulas.

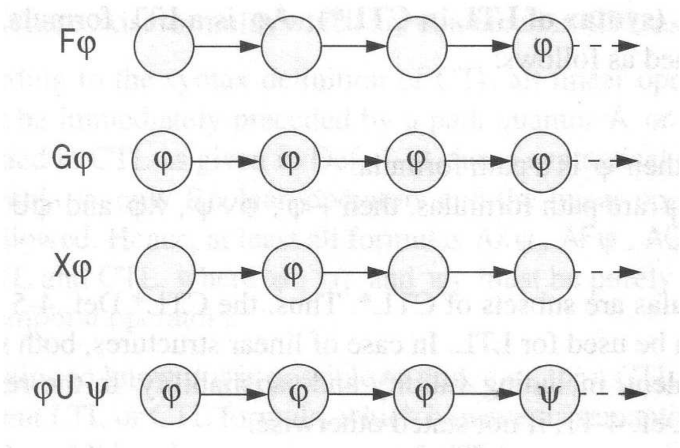
- Every formula is evaluated on a path p.
  - Also every state formula f of classical logic (see below).
  - Let F and G denote formulas.
  - Then also the following are formulas:

    $\mathbf{X}\ F$ ("next time F"), often written $\bigcirc F$,
    $\mathbf{G}\ F$ ("always F"), often written $\Box F$,
    $\mathbf{F}\ F$ ("eventually F"), often written $\Diamond F$,
    $F\ \mathbf{U}\ G$ ("F until G").

- Semantics: $p \models P$ ("P holds in path p").
  - $p^i := \langle p_i, p_{i+1}, \ldots \rangle$.

    $p \models f :\Leftrightarrow p_0 \models f$.
    $p \models \mathbf{X}\ F \ :\Leftrightarrow p^1 \models F$.
    $p \models \mathbf{G}\ F \ :\Leftrightarrow \forall i \in \mathbb{N} : p^i \models F$.
    $p \models \mathbf{F}\ F \ :\Leftrightarrow \exists i \in \mathbb{N} : p^i \models F$.
    $p \models F\ \mathbf{U}\ G \ :\Leftrightarrow \exists i \in \mathbb{N} : p^i \models G \wedge \forall j \in \mathbb{N}_i : p^j \models F$.

Formulas



Thomas Kropf: "Introduction to Formal Hardware Verification", 1999.

Branching versus Linear Time Logic

We use temporal logic to specify a system property P.

- Core question: $S \models P$ ("P holds in system S").
  - System $S = \langle I, R \rangle$, temporal logic formula P.
- Branching time logic:
  - $S \models P \;:\Leftrightarrow\; S, s_0 \models P$, for every initial state s_0 of S.
  - Property P must be evaluated on every pair (S, s_0) of system S and initial state s_0.
  - Given a computation tree with root s_0, P is evaluated on that tree.
- Linear time logic:
  - $S \models P \;:\Leftrightarrow\; r \models P$, for every run r of s.
  - Property P must be evaluated on every run r of S.
  - Given a computation tree with root s_0, P is evaluated on every path of that tree originating in s_0.
    - If P holds for every path, P holds on S.

Branching versus Linear Time Logic

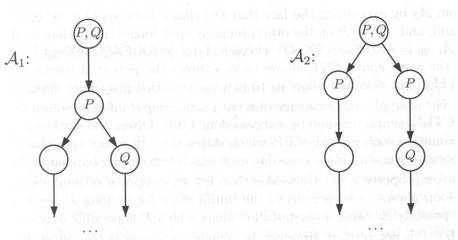


Fig. 2.4. Two automata, indistinguishable for PLTL

B. Berard et al: "Systems and Software Verification", 2001.

- Linear time logic: both systems have the same runs.
  - Thus every formula has same truth value in both systems.
- Branching time logic: the systems have different computation trees.
  - Take formula $\mathbf{AX}(\mathbf{EX}\ Q \land \mathbf{EX}\ \neg Q)$.
  - True for left system, false for right system.

The two variants of temporal logic have different expressive power.

Branching versus Linear Time Logic

Is one temporal logic variant more expressive than the other one?

- CTL formula: **AG**(**EF** F).
  - "In every run, it is at any time still possible that later F will hold".
  - Property cannot be expressed by any LTL logic formula.
- LTL formula: $\Diamond\Box F$ (i.e. **FG** F).
  - "In every run, there is a moment from which on F holds forever.".
  - Naive translation **AFG** F is not a CTL formula.
    - **G** F is a path formula, but **F** expects a state formula!
  - Translation **AFAG** F expresses a stronger property (see next page).
  - Property cannot be expressed by any CTL formula.

None of the two variants is strictly more expressive than the other one; no variant can express every system property.

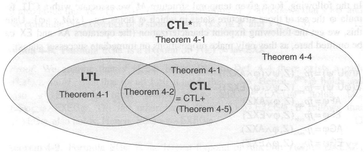
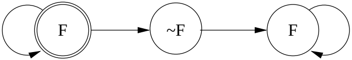


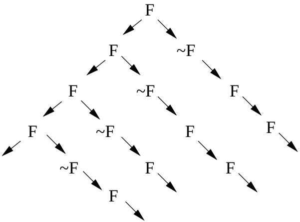
Fig. 4-8. Expressiveness of CTL*, CTL+, CTL and LTL

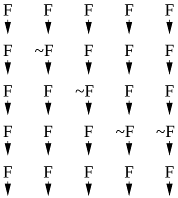
: Thomas Kropf: "Introduction to Formal Hardware Verification", 1999.

Branching versus Linear Time Logic

Proof that **AFAG** F (CTL) is different from $\Diamond\Box F$ (LTL).







AFAG F <=> false

In every run, there is a moment when it is guaranteed that from now on F holds forever.

<>[] F <=> true

In every run, there is a moment from which on F holds forever.

1. The Basics of Temporal Logic

2. Specifying with Linear Time Logic

Linear Time Logic

Why using linear time logic (LTL) for system specifications?

- LTL has many advantages:
  - LTL formulas are easier to understand.
    - Reasoning about computation paths, not computation trees.
    - No explicit path quantifiers used.
  - LTL can express most interesting system properties.
    - Invariance, guarantee, response, ... (see later).
  - LTL can express fairness constraints (see later).
    - CTL cannot do this.
    - But CTL can express that a state is reachable (which LTL cannot).
- LTL has also some disadvantages:
  - LTL is strictly less expressive than other specification languages.
    - CTL^* or μ-calculus.
  - Asymptotic complexity of model checking is higher.
    - LTL: exponential in size of formula; CTL: linear in size of formula.
    - In practice the number of states dominates the checking time.

Frequently Used LTL Patterns

In practice, most temporal formulas are instances of particular patterns.

Pattern	Pronounced	Name
$\Box F$	always F	invariance
$\Diamond F$	eventually F	guarantee
$\Box\Diamond F$	F holds infinitely often	recurrence
$\Diamond\Box F$	eventually F holds permanently	stability
$\Box(F \Rightarrow \Diamond G)$	always, if F holds, then eventually G holds	response
$\Box(F \Rightarrow (G\ \mathbf{U}\ H))$	always, if F holds, then G holds until H holds	precedence

Typically, there are at most two levels of nesting of temporal operators.

Examples

- Mutual exclusion: $\Box\neg(pc_1 = C \wedge pc_2 = C)$.
  - Alternatively: $\neg\Diamond(pc_1 = C \wedge pc_2 = C)$.
  - Never both components are simultaneously in the critical region.
- No starvation: $\forall i : \Box(pc_i = W \Rightarrow \Diamond pc_i = R)$.
  - Always, if component i waits for a response, it eventually receives it.
- No deadlock: $\Box\neg\forall i : pc_i = W$.
  - Never all components are simultaneously in a wait state W.
- Precedence: $\forall i : \Box(pc_i \neq C \Rightarrow (pc_i \neq C \ \mathbf{U}\ lock = i))$.
  - Always, if component i is out of the critical region, it stays out until it receives the shared lock variable (which it eventually does).
- Partial correctness: $\Box(pc = L \Rightarrow C)$.
  - Always if the program reaches line L, the condition C holds.
- Termination: $\forall i : \Diamond(pc_i = T)$.
  - Every component eventually terminates.

Temporal Rules

Temporal operators obey a number of fairly intuitive rules.

- Extraction laws:
  - $\Box F \;\Leftrightarrow\; F \wedge \bigcirc\Box F$.
  - $\Diamond F \;\Leftrightarrow\; F \vee \bigcirc\Diamond F$.
  - $F \;\mathbf{U}\; G \;\Leftrightarrow G \vee (F \wedge \bigcirc(F \;\mathbf{U}\; G))$.
- Negation laws:
  - $\neg\Box F \;\Leftrightarrow\; \Diamond\neg F$.
  - $\neg\Diamond F \;\Leftrightarrow \Box\neg F$.
  - $\neg(F \;\mathbf{U}\; G) \;\Leftrightarrow\; (\neg G) \;\mathbf{U}\; (\neg F \wedge \neg G)$.
- Distributivity laws:
  - $\Box(F \wedge G) \;\Leftrightarrow\; (\Box F) \wedge (\Box G)$.
  - $\Diamond(F \vee G) \;\Leftrightarrow\; (\Diamond F) \vee (\Diamond G)$.
  - $(F \wedge G) \;\mathbf{U}\; H \;\Leftrightarrow\; (F \;\mathbf{U}\; H) \wedge (G \;\mathbf{U}\; H)$.
  - $F \;\mathbf{U}\; (G \vee H) \;\Leftrightarrow\; (F \;\mathbf{U}\; G) \vee (F \;\mathbf{U}\; H)$.
  - $\Box\Diamond(F \vee G) \;\Leftrightarrow\; (\Box\Diamond F) \vee (\Box\Diamond G)$.
  - $\Diamond\Box(F \wedge G) \;\Leftrightarrow\; (\Diamond\Box F) \wedge (\Diamond\Box G)$.

Classes of System Properties

There exists two important classes of system properties.

- Safety Properties:
  - A safety property is a property such that, if it is violated by a run, it is already violated by some finite prefix of the run.
    - This finite prefix cannot be extended in any way to a complete run satisfying the property.
  - Example: $\Box F$.
    - The violating run $F \rightarrow F \rightarrow \neg F \rightarrow \ldots$ has the prefix $F \rightarrow F \rightarrow \neg F$ that cannot be extended in any way to a run satisfying $\Box F$.
- Liveness Properties:
  - A liveness property is a property such that every finite prefix can be extended to a complete run satisfying this property.
    - Only a complete run itself can violate that property.
  - Example: $\Diamond F$.
    - Any finite prefix p can be extended to a run $p \rightarrow F \rightarrow \ldots$ which satisfies $\Diamond F$.

System Properties

Not every system property is itself a safety property or a liveness property.

- Example: $P :\Leftrightarrow (\Box A) \wedge (\Diamond B)$
  - Conjunction of a safety property and a liveness property.
- Take the run $[A, \neg B] \rightarrow [A, \neg B] \rightarrow [A, \neg B] \rightarrow \ldots$ violating P.
  - Any prefix $[A, \neg B] \rightarrow \ldots \rightarrow [A, \neg B]$ of this run can be extended to a run $[A, \neg B] \rightarrow \ldots \rightarrow [A, \neg B] \rightarrow [A, B] \rightarrow [A, B] \rightarrow \ldots$ satisfying P.
  - Thus P is not a safety property.
- Take the finite prefix $[\neg A, B]$.
  - This prefix cannot be extended in any way to a run satisfying P.
  - Thus P is not a liveness property.

So is the distinction "safety" versus "liveness" really useful?.

System Properties

The real importance of the distinction is stated by the following theorem.

- Theorem:

  Every system property P is a conjunction $S \wedge L$ of some safety property S and some liveness property L.

  - If L is "true", then P itself is a safety property.
  - If S is "true", then P itself is a liveness property.
- Consequence:
  - Assume we can decompose P into appropriate S and L.
  - For proving $M \models P$, it then suffices to perform two proofs:
    - A safety proof: $M \models S$.
    - A liveness proof: $M \models L$.
  - Different strategies for proving safety and liveness properties.

For verification, it is important to decompose a system property in its "safety part" and its "liveness part".

Proving Invariance

We only consider a special case of a safety property.

- Prove $M \models \Box F$.
  - F is a state formula (a formula without temporal operator).
  - Prove that F is an invariant of system M.
- $M = \langle I, R \rangle$.
  - $I(s) :\Leftrightarrow \ldots$
  - $R(s, s') :\Leftrightarrow R_0(s, s') \lor R_1(s, s') \lor \ldots \lor R_{n-1}(s, s')$.
- Induction Proof.
  - $\forall s : I(s) \Rightarrow F(s)$.
    - Proof that F holds in every initial state.
  - $\forall s, s' : F(s) \land R(s, s') \Rightarrow F(s')$.
    - Proof that each transition preserves F.
    - Reduces to a number of subproofs:

$$F(s) \land R_0(s, s') \Rightarrow F(s')$$
$$\ldots$$
$$F(s) \land R_{n-1}(s, s') \Rightarrow F(s')$$

Example

$$\begin{array}{lcl} \textbf{var } x := 0 & & \\ \textbf{loop} & \| & \textbf{loop} \\ \quad p_0 : \textbf{wait } x = 0 & & \quad q_0 : \textbf{wait } x = 1 \\ \quad p_1 : x := x + 1 & & \quad q_1 : x := x - 1 \end{array}$$

$$State = \{p_0, p_1\} \times \{q_0, q_1\} \times \mathbb{Z}.$$

$$I(p, q, x) :\Leftrightarrow p = p_0 \wedge q = q_0 \wedge x = 0.$$
$$R(\langle p, q, x\rangle, \langle p', q', x'\rangle) :\Leftrightarrow P_0(\ldots) \vee P_1(\ldots) \vee Q_0(\ldots) \vee Q_1(\ldots).$$

$$P_0(\langle p, q, x\rangle, \langle p', q', x'\rangle) :\Leftrightarrow p = p_0 \wedge x = 0 \wedge p' = p_1 \wedge q' = q \wedge x' = x.$$
$$P_1(\langle p, q, x\rangle, \langle p', q', x'\rangle) :\Leftrightarrow p = p_1 \wedge p' = p_0 \wedge q' = q \wedge x' = x + 1.$$
$$Q_0(\langle p, q, x\rangle, \langle p', q', x'\rangle) :\Leftrightarrow q = q_0 \wedge x = 1 \wedge p' = p \wedge q' = q_1 \wedge x' = x.$$
$$Q_1(\langle p, q, x\rangle, \langle p', q', x'\rangle) :\Leftrightarrow q = q_1 \wedge p' = p \wedge q' = q_0 \wedge x' = x - 1.$$

Prove $\langle I, R\rangle \models \Box(x = 0 \vee x = 1)$.

Inductive System Properties

The induction strategy may not work for proving $\Box F$

- Problem: F is not inductive.
  - F is too weak to prove the induction step.
    - $F(s) \wedge R(s, s') \Rightarrow F(s')$.
- Solution: find stronger invariant I.
  - If $I \Rightarrow F$, then $(\Box I) \Rightarrow (\Box F)$.
  - It thus suffices to prove $\Box I$.
- Rationale: I may be inductive.
  - If yes, I is strong enough to prove the induction step.
    - $I(s) \wedge R(s, s') \Rightarrow I(s')$.
  - If not, find a stronger invariant I' and try again.
- Invariant I represents additional knowledge for every proof.
  - Rather than proving $\Box P$, prove $\Box(I \Rightarrow P)$.

The behavior of a system is captured by its strongest invariant.

Example

- Prove $\langle I, R \rangle \models \Box(x = 0 \vee x = 1)$.
  - Proof attempt fails.
- Prove $\langle I, R \rangle \models \Box G$.

$$\begin{aligned} &G :\Leftrightarrow \\ &\quad (x = 0 \vee x = 1) \wedge \\ &\quad (p = p_1 \Rightarrow x = 0) \wedge \\ &\quad (q = q_1 \Rightarrow x = 1). \end{aligned}$$

  - Proof works.
  - $G \Rightarrow (x = 0 \vee x = 1)$ obvious.

See the proof presented in class.

Proving Liveness

$$
\begin{array}{lcl}
\textbf{var } x := 0, y := 0 & & \\
\textbf{loop} & \| & \textbf{loop} \\
\quad x := x + 1 & & \quad y := y + 1
\end{array}
$$

$State = \mathbb{N} \times \mathbb{N}$; $Label = \{p, q\}$.
$I(x, y) :\Leftrightarrow x = 0 \wedge y = 0$.
$R(l, \langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$
$(l = p \wedge x' = x + 1 \wedge y' = y) \vee (l = q \wedge x' = x \wedge y' = y + 1)$.

- Prove $\langle I, R \rangle \models \Diamond x = 1$.
  - $[x = 0, y = 0] \rightarrow [x = 0, y = 1] \rightarrow [x = 0, y = 2] \rightarrow \ldots$
  - This run violates (as the only one) $\Diamond x = 1$.
  - Thus the system as a whole does not satisfy $\Diamond x = 1$.

For proving liveness properties, “unfair” runs have to be ruled out.

Enabling Condition

When is a particular transition enabled for execution?

- $Enabled_R(l,s) :\Leftrightarrow \exists t : R(l,s,t)$.
  - Labeled transition relation R, label l, state s.
  - Read: "Transition (with label) l is enabled in state s (w.r.t. R)".
- Example (previous slide):

$$\begin{array}{l} Enabled_R(p, \langle x,y\rangle) \\ \quad \Leftrightarrow \exists x', y' : R(p, \langle x,y\rangle, \langle x',y'\rangle) \\ \quad \Leftrightarrow \exists x', y' : \\ \qquad\quad (p = p \wedge x' = x+1 \wedge y' = y) \vee \\ \qquad\quad (p = q \wedge x' = x \wedge y' = y+1) \\ \quad \Leftrightarrow (\exists x', y' : p = p \wedge x' = x+1 \wedge y' = y) \vee \\ \qquad (\exists x', y' : p = q \wedge x' = x \wedge y' = y+1) \\ \quad \Leftrightarrow \text{true} \vee \text{false} \\ \quad \Leftrightarrow \text{true}. \end{array}$$

  - Transition p is always enabled.

Weak Fairness

- Weak Fairness
  - A run $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \xrightarrow{l_2} \ldots$ is weakly fair to a transition l, if
    - if transition l is eventually permanently enabled in the run,
    - then transition l is executed infinitely often in the run.

      $(\exists i : \forall j \geq i : Enabled_R(l, s_j)) \Rightarrow (\forall i : \exists j \geq i : l_j = l)$.
  - The run in the previous example was not weakly fair to transition p.
- LTL formulas may explicitly specify weak fairness constraints.
  - Let E_l denote the enabling condition of transition l.
  - Let X_l denote the predicate "transition l is executed".
  - Define $WF_l :\Leftrightarrow (\Diamond\Box E_l) \Rightarrow (\Box\Diamond X_l)$.

    If l is eventually enabled forever, it is executed infinitely often.
  - Prove $\langle I, S\rangle \models (WF_l \Rightarrow P)$.

    Property P is only proved for runs that are weakly fair to l.

Alternatively, a model may also have weak fairness "built in".

Proving a Guarantee

We only consider a special case of a liveness property.

- Prove $\langle I, R \rangle \models \Diamond F$.
  - Proof that F is a guarantee of the system.
  - F is a state formula (a formula without a temporal operator).
- Decomposition: sequence of properties $F_0, F_1, \ldots, F_n = F$.
  - Prove $\langle I, R \rangle \models \Diamond F_0$.
  - Prove $\langle I, R \rangle \models \Box(F_0 \Rightarrow \Diamond F_1)$.
  - Prove $\langle I, R \rangle \models \Box(F_1 \Rightarrow \Diamond F_2)$.
  - $\ldots$
  - Prove $\langle I, R \rangle \models \Box(F_{n-1} \Rightarrow \Diamond F)$.

Typically, guarantee proofs have to be decomposed into multiple proofs.

Proving a Guarantee

- Core proof: $\langle I, R \rangle \models \Diamond F$.
  - Find lucky transition l with enabling condition E_l.
    - The execution of l makes F true.
    - As long as F is not true, l is enabled.
    - By weak fairness, either F becomes true or l is eventually executed.
    - Until l is executed, additional property H holds.

$$\neg F(s) \wedge I(s) \Rightarrow H(s) \wedge E_l(s).$$
$$\neg F(s) \wedge H(s) \wedge E_l(s) \wedge \neg R(l, s, s') \Rightarrow H(s') \wedge E_l(s').$$
$$\neg F(s) \wedge H(s) \wedge R(l, s, s') \Rightarrow F(s').$$

- Core proofs: $\langle I, R \rangle \models \Box(G \Rightarrow \Diamond F)$.
  - Find lucky transition l with enabling condition E_l.
    - Prove: $\neg F(s) \wedge G(s) \Rightarrow H(s) \wedge E_l(s)$.
    - Prove: $\neg F(s) \wedge H(s) \wedge E_l(s) \wedge \neg R(l, s, s') \Rightarrow H(s') \wedge E_l(s')$.
    - Prove: $\neg G(s) \wedge H(s) \wedge R(l, s, s') \Rightarrow F(s')$.

Sometimes augmented by proofs using well-founded orderings.

Example

$State = \mathbb{N} \times \mathbb{N}$; $Label = \{p, q\}$.
$I(x, y) :\Leftrightarrow x = 0 \wedge y = 0$.
$R(l, \langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$
$\quad (l = p \wedge x' = x + 1 \wedge y' = y) \vee (l = q \wedge x' = x \wedge y' = y + 1)$.

- Prove $\langle I, R \rangle \models \Diamond x = 1$.
  - Lucky transition p, additional property $H :\Leftrightarrow x = 0$.

$x \neq 1 \wedge (x = 0 \wedge y = 0) \Rightarrow x = 0 \wedge \text{true}$.
$x \neq 1 \wedge x = 0 \wedge \text{true} \wedge (x' = x \wedge y' = y + 1) \Rightarrow x' = 0 \wedge \text{true}$.
$x \neq 1 \wedge x = 0 \wedge (x' = x + 1 \wedge y' = y) \Rightarrow x' = 1$.

Strong Fairness

- Strong Fairness
  - A run $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \xrightarrow{l_2} \ldots$ is strongly fair to a transition l, if
    - if l is infinitely often enabled in the run,
    - then l is also infinitely often executed the run.

      $(\forall i : \exists j \geq i : Enabled_R(l, s_j)) \Rightarrow (\forall i : \exists j \geq i : l_j = l)$.
  - If r is strongly fair to l, it is also weakly fair to l (but not vice versa).
- LTL formulas may explicitly specify strong fairness constraints.
  - Let E_l denote the enabling condition of transition l.
  - Let X_l denote the predicate "transition l is executed".
  - Define $SF_l :\Leftrightarrow (\Box\Diamond E_l) \Rightarrow (\Box\Diamond X_l)$.

    If l is enabled infinitely often, it is executed infinitely often.
  - Prove $\langle I, S\rangle \models (SF_l \Rightarrow P)$.

    Property P is only proved for runs that are strongly fair to l.

A much stronger requirement to the fairness of a system.

Example

var x=0
loop
  $a: \; x := -x$
  $b: \; \textbf{choose}\; x := 0 \;[]\; x := 1$

$State := \{a, b\} \times \mathbb{Z}; Label = \{A, B_0, B_1\}.$
$I(p, x) :\Leftrightarrow p = a \wedge x = 0.$
$R(l, \langle p, x\rangle, \langle p', x'\rangle) :\Leftrightarrow$
$\quad (l = A \wedge (p = a \wedge p' = b \wedge x' = -x)) \vee$
$\quad (l = B_0 \wedge (p = b \wedge p' = a \wedge x' = 0)) \vee$
$\quad (l = B_1 \wedge (p = b \wedge p' = a \wedge x' = 1)).$

- Prove: $\langle I, R\rangle \models \Diamond x = 1.$
  - Take violating run $[a, 0] \xrightarrow{A} [b, 0] \xrightarrow{B_0} [a, 0] \xrightarrow{A} [b, 0] \xrightarrow{B_0} [a, 0] \xrightarrow{A} \ldots$
  - $Enabled_{B_1}(p, x) :\Leftrightarrow p = b.$
  - Run is weakly fair but not strongly fair to B_1.