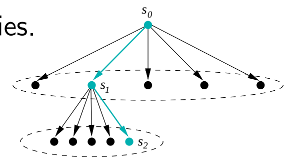
Specifying Properties of Concurrent Systems

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

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

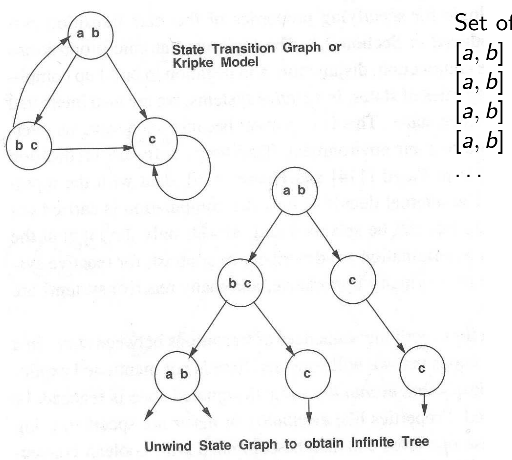


Figure 3.1
Computation trees.

Set of system runs:
$[a, b] \rightarrow c \rightarrow c \rightarrow \ldots$
$[a, b] \rightarrow [b, c] \rightarrow c \rightarrow \ldots$
$[a, b] \rightarrow [b, c] \rightarrow [a, b] \rightarrow \ldots$
$[a, b] \rightarrow [b, c] \rightarrow [a, b] \rightarrow \ldots$
...

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.

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

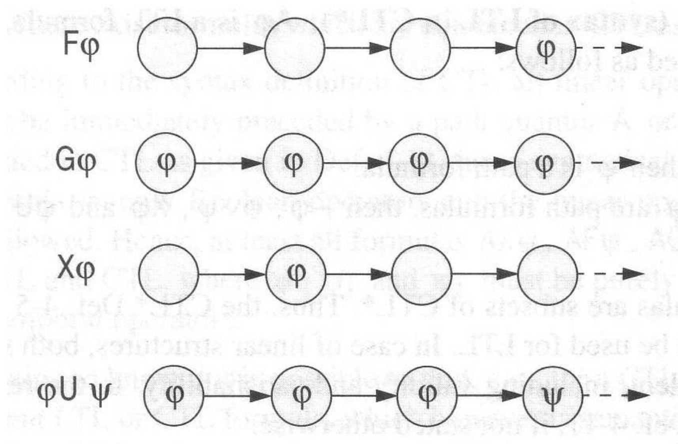
All formulas are path formulas.

- 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").
  - $p^i := \langle p_i, p_{i+1}, \ldots \rangle$.

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

Formulas



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

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.

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.
- $M = \langle I, R \rangle$.
  - $I(s) :\Leftrightarrow \ldots$
  - $R(s, s') :\Leftrightarrow R_0(s, s') \vee R_1(s, s') \vee \ldots \vee R_{n-1}(s, s')$.
- Induction Proof.
  - $\forall s : I(s) \Rightarrow F(s)$.
    - Proof that F holds in every initial state.
  - $\forall s, s' : F(s) \wedge R(s, s') \Rightarrow F(s')$.
    - Proof that each transition preserves F.
    - Reduces to a number of subproofs:

      $F(s) \wedge R_0(s, s') \Rightarrow F(s')$

      $\ldots$

      $F(s) \wedge R_{n-1}(s, s') \Rightarrow F(s')$

Proving Liveness

```
var x := 0, y := 0
loop              ||    loop
  x := x + 1              y := y + 1
```

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

- 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.

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.

A (relatively) weak requirement to the fairness of a system.

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.

      $(\forall i : \exists j \geq i : \mathit{Enabled}_R(l, s_j)) \Rightarrow (\forall i : \exists j \geq i : l_j = l)$.
  - If r is weakly fair to l, it is also strongly 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
  $a: \; x := -x$
  $b: \;$ **choose** $x := 0 \; [] \; x := 1$

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

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