Lecture 6
Liveness and Linear Time Temporal Logic

Peterson-Fischer: Possible Specifications

Variables: \(y_1, y_2: \text{boolean}, t: \{1,2\}\)
Initially: \(y_1 = y_2 = \text{false}, t = 1\)

Mutual Exclusion: the two processes never simultaneously reach \(l_3, m_3\)
Absence of Starvation: If the left process is at \(l_1\), it will later reach \(l_3\)
Bounded Overtaking: If the left process is at \(l_1\), the other process will reach \(m_3\) at most once (twice?) before the left process reaches \(l_3\)

Specifying progress:

Idea: Specify control states that should occur infinitely often,
- Typically, what can happen in infinite loops.

In Promela:
- \textit{progress} label should be visited \textbf{infinitely often}
- \textit{accept} label should \textbf{not} be visited \textbf{infinitely often}

Attempt at specifying progress

What about fairness?

Peterson-Fischer Mutual Exclusion

Attempt at specifying progress

What about fairness?
What about fairness?

```c
#define true 1
#define false 0
#define turn1 false
#define turn2 true

#include <stdbool.h>

proctype P1() {
    progress0: do :: skip
    :: y1 = true -> break
    od ;
    l1: t = turn2;
    l2: (y2 == false || t == turn1) ;
    mutex++ ;
    assert (mutex <= 1) ;
    progress3:  /* critical section */
    mutex -- ;
    atomic{ y1 = false ; goto l0 }
}

proctype P2() {
    m0: y2 = true;
    m1: t = turn1;
    m2: (y1 == false || t == turn2) ;
    mutex++ ;
    assert (mutex <= 1) ;
    m3:  /* critical section */
    mutex-- ;
    atomic{ y2 = false ; goto m0 }
}

init {
    atomic { run P1() ; run P2() }
}
```

AB Protocol (version 5, limiting checks)

```c
#define MAXMSG 4

#include <stdbool.h>

proctype Receiver() {
    byte recd, expected = 0;
    bit rbit = 1, seqno;
    do
    :: s_r ? msg (recd, seqno) ->
    if :: seqno != rbit ->
    sink!recd;
    progress: rbit = 1 - rbit
    :: else
    fi
    :: r_s ! ack (rbit)
    :: (1) -> progress2: skip
    od
}
```

Linear Time Temporal Logic

**Formulas**

- **State formulas** (denoted by $p, q$): properties about states
  
a e.g., $\varphi \cdot x = y \cdot st m3$

- Formulas built using temporal operators
  
  - $\varphi$ "in the next state $\varphi$"
  
  - $\varphi$ "always (in all future states) $\varphi$"

  - $\varphi$ "sometimes (in some future states) $\varphi$"

  - $\varphi \land \psi$ "until $\psi$"

  - $\varphi \land \psi$ "unless $\psi$"

- And boolean connectives

  $\neg$ $\land$ $\lor$ $\rightarrow$ $\leftarrow$
Linear Time Temporal Logic: interpretation

- Formulas interpreted over computations
- A formula is either true or false in each state of a computation

Example:

The following formulas hold at the particular state:
- \( \neg p \land q \)
- \( (p \land q) \land p \)

The following formula does not hold at the particular state:
- \( \Box p \)

Meaning of operators

- \( \Diamond \phi \)
- \( \Box \phi \)
- \( \phi \)
- \( \phi \lor \psi \)
- \( \phi \land \psi \)
- \( \phi \rightarrow \psi \)
- \( \phi \lor \psi \lor \psi \)

Operators: Formal semantics

Let \( \beta \) be computation \( s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, ... \)
- \( (\beta, i) \models \phi \) denotes that \( \phi \) is true in state \( s_i \)
- \( (\beta, i) \models p \) (for \( p \) state formula) iff \( p \) holds in state \( s_i \)
- Boolean connectives work as usual
- \( (\beta, i) \models \Box \phi \) iff \( \forall j \geq i \) \( (\beta, j) \models \phi \)
- \( (\beta, i) \models \Diamond \phi \) iff \( \exists j \geq i \) \( (\beta, j) \models \phi \)
- \( (\beta, i) \models \phi U \psi \) iff \( \exists j \geq i \) \( (\beta, j) \models \psi \) and \( \forall k : i \leq k < j : (\beta, k) \models \phi \)
- \( (\beta, i) \models \phi W \psi \) iff \( (\beta, j) \models \phi \lor \psi \) or \( (\beta, j) \models \Box \phi \)

Linear Temporal logic: examples

- \( \Box p \) \( p \) is invariant
- \( \Box (p \rightarrow \Box p) \) \( p \) is stable
- \( \neg q W (p \land \neg q) \) \( p \) precedes \( q \)
- \( \Box (p \rightarrow \Diamond q) \) \( p \) leads to \( q \)
- \( \Diamond p \) \( p \) infinitely often
- \( \Box \Diamond p \) \( p \) from some point on

Peterson-Fischer: Possible Specifications

Variables: \( y_1, y_2 \): boolean, \( t \): \{1,2\}
Initially \( y_1 = y_2 = false, t = 1 \)

Mutual Exclusion: \( \Box \neg (at l3 \land at m3) \)
Absence of Starvation: \( \Box (at l2 \rightarrow \Diamond at l3) \)
Bounded Overtaking: \( \Box (at l2 \rightarrow (\neg at m3 \land (at m3 U (\neg at m3 U at l3)))) \)

Example: GCD Computation

Action System
Variables: \( y_1, y_2 \): integer
Initially \( y_1=x_1, y_2=x_2 \)

Partial Correctness:
\[ x1 > 0 \land x2 > 0 \rightarrow \Box (P@stop \rightarrow y_1 = y_2 = \text{gcd}(x_1, x_2)) \]

Total Correctness:
\[ x1 > 0 \land x2 > 0 \rightarrow (\Box \text{loop} \rightarrow \Diamond P@stop) \]
Example: Termination Detection

Variables: ch[N]: FIFO Channel of {black,white}
q[N]: boolean
dist = false: boolean

Processes: Q[0] || ... || Q[N-1] || P[0] || ... || P[N-1]

q[i] := true
Q[i]:
q[i-1] -> q[i] := false ;
if i=0 then dist := true
P[i]:
ch[i]?black -> ch[i+1]!black
ch[i]?white -> if q[i] then ch[i+1]!white else ch[i+1]!black
P[0]:
ch[0]?black
ch[1]!white ; dist := false
¬ dist
dist
ch[0]?white
term

Abbreviation:
Terminated ≡ ∀ i : 0 ≤ i < N :: q[i]

Safety Property:
□ [P[0]@term -> Terminated ]

Liveness Property:
Terminated -> ◊ P[0]@term

Specifying linear properties by automata

- A linear time property is a set of computations.
- Can be seen as a language of (infinite) words,
  - Alphabet: possible assignments of truth-values to occurring state formulas
- A language can be specified by an automaton
- It turns out that it is better to represent the complement, i.e.,
  the set of computations that violate the property

Automata Specifications: examples

- □ p
- □ ( p -> □ p )
- ¬ q W ( p ∧ ¬ q)
- □( p -> ◊ q)
- □ ◊ p
- ◊□ p

Superposing Automaton on Transition system

Superposition of Automaton (Σ, Q, Q0, ->, Φ) onto Transition System (S, S0, ->, V, L) has

S x Q as the set of states
s,s' -> s'' if s -> s' and q -> q''

Accepting condition adapted from Φ

Safety vs. Liveness properties.

- Safety property is of the form
  "nothing bad will ever happen"
  A computation that violates the property will do so after a finite number of transitions
  Enough to specify set of finite violating (finite) automaton
  Can be done by (standard) finite automaton

- Liveness property is of the form
  "something good will eventually happen"
  A computation that violates the property can never so after a finite number of transitions
  We must specify set of infinite violating computations

- Any omega-regular property is conjunction of safety and liveness properties.
Automata over infinite words

Automaton over infinite words: \((\Sigma, Q, Q_0, \rightarrow, \Phi)\)

where,

- \(\Sigma\) is an alphabet (typically set of assignments of truth values to state formulas)
- \(Q\) is a set of states
- \(Q_0\) is a set of initial states
- \((\text{a subset of } S \times \Sigma \times S)\) is a transition relation
- \(\Phi\) is an acceptance condition

A run over infinite word \(a_1 a_2 a_3 a_4 a_5 a_6 a_7\) is

\[\begin{align*}
q_0 &\rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8
\end{align*}\]

which satisfies the acceptance condition \(\Phi\)

---

Classes of acceptance conditions

- **Büchi automata**
  - One set of accepting states \(F\)
  - Acceptance condition \(\bigcirc F\)
- **Generalized Büchi automata**
  - Collection of sets of accepting states \(F_1, \ldots, F_n\)
  - Acceptance condition \(\bigcirc F_1 \land \ldots \land \bigcirc F_n\)
- **Rabin automata**
  - Collection of pairs of accepting states \((F_1, G_1), \ldots, (F_n, G_n)\)
  - Acceptance condition \(\lor (\bigcirc F_1 \land \neg \bigcirc G_1)\)
- **Streett automata**
  - Collection of pairs of accepting states \((F_1, G_1), \ldots, (F_n, G_n)\)
  - Acceptance condition \(\land (\bigcirc F_i \rightarrow \bigcirc G_i)\)

---

Examples of Büchi Automata

- \(\square \diamond p\)
- \(\diamond \square p\)
- \(\square (p \rightarrow \diamond q)\)
- \(\diamond (p \land \neg q)\)

More Examples of Büchi Automata

- \((\square \diamond p \land \diamond \square q)\)
- \((\diamond \square p \rightarrow \square \diamond q)\)

---

Translation Results

- Any property expressible in linear temporal logic is accepted by some Büchi automaton
- There are properties accepted by some Büchi automaton that cannot be expressed in linear temporal logic
  - e.g., “\(p\) is true in every even state”

---

Convenience of expression

- Some properties are easier to understand as automata
- **Bounded Overtaking:**
  - \(\square (\at 11 \rightarrow (\neg \at m3 \lor (\at m3 \lor (\neg \at m3 \lor \at 13)))\)
Convenience of expression

- Some properties are easier to understand as automata
- Bounded Overtaking:
  $\square (at \ l_1 \rightarrow (\neg at \ m_3 \ U (at \ m_3 \ U (\neg at \ m_3 \ U at \ l_3))))$

Negation:

AB Protocol (version 9, w acceptance)

```plaintext
active proctype Sender() {
  byte data = 0;
  bit sbit, seqno = 0;
  source?data;
  do
  :: s_r ! msg(data, sbit) -> Slabel: skip
  :: (1) -> skip
  :: r_s ? ack(seqno);
  if :: seqno == sbit -> sbit = 1 - sbit;
  source?data
  :: else
  fi
  od
}
active proctype Receiver() {
  byte recd = 0;
  bit rbit = 1, seqno;
  do
  :: s_r ? msg(recd, seqno) ->
  if :: seqno != rbit -> Rsuccess: sink!recd;
  rbit = 1 - rbit
  :: else
  fi
  :: r_s ! ack(rbit) -> Rlabel: skip
  :: (1) -> skip
  od
}
```

AB Protocol (version 9, the never claim)

```plaintext
never { /* !(([] <> p && [] <> q) -> [] <> r) */
T0_init:
  if :: (!((r)) && (p) && (q)) -> goto accept_S485
  :: (!((r)) && (p)) -> goto T2_S485
  :: (!((r))) -> goto T0_S485
  :: (1) -> goto T0_init
  fi;
accept_S485:
  if :: (!((r))) -> goto T0_S485
  fi;
T2_S485:
  if :: (!((r)) && (q)) -> goto accept_S485
  :: (!((r))) -> goto T2_S485
  fi;
T0_S485:
  if :: (!((r)) && (p) && (q)) -> goto accept_S485
  :: (!((r)) && (p)) -> goto T2_S485
  :: (!((r))) -> goto T0_S485
  fi;
}
```