Linear Temporal Logic

and What It's Good For

Joshua D. Guttman

Worcester Polytechnic Institute

18 February 2010

One View of Computation

- A computational system:
  - At each time, some state s
  - Certain states may transition to certain other states

$$s \longrightarrow s'$$

One View of Computation

- A computational system:
  - At each time, some state s
  - Certain states may transition to certain other states

$$s \longrightarrow s'$$

- A computational system is a directed graph $\mathcal{G}$
  - Nodes are states
  - Arrows are transitions
  - Computations are paths through the graph starting from a start state s_0

One View of Computation

- A computational system:
  - At each time, some state s
  - Certain states may transition to certain other states

$$s \longrightarrow s'$$

- A computational system is a directed graph $\mathcal{G}$
  - Nodes are states
  - Arrows are transitions
  - Computations are paths through the graph starting from a start state s_0
- A specification is a set of paths $\mathcal{S}$
  - The ones we consider acceptable
  - A system $\mathcal{G}$ meets a specification $\mathcal{S}$ iff

$$\text{paths}(\mathcal{G}) \subseteq \mathcal{S}$$

For Instance

- State:
  - Account balance integer
  - Issuing-coupon (true/false)

  (and maybe other info)
- Specification:
  - Always: Issuing-coupon implies account balance $= 0$
  - Always: (Not issuing-coupon) until account balance $= 10$

A Real Example

Using shared memory for message passing

A Real Example

Using shared memory for message passing

```
receive ()
val : int
if start < end
then {
  val := read(buff[start]);
  start := start+1;
  return(some, val)
  }
else return(none, 0)
```

A Real Example

Using shared memory for message passing

```
receive ()
val : int
if start < end
then {
  val := read(buff[start]);
  start := start+1;
  return(some, val)
  }
else return(none, 0)
```

```
send (val : int)

if end < 3
then {
  write(buff[end], val);
  end := end+1;
  return(success)
  }
else return(failure)
```

A Real Example

Using shared memory for message passing

```
receive ()
val : int
if start < end
then {
  val := read(buff[start]);
  start := start+1;
  return(some, val)
  }
else return(none, 0)
```

```
send (val : int)

if end < 3
then {
  write(buff[end], val);
  end := end+1;
  return(success)
  }
else return(failure)
```

```
reset ()
if start = 3 and end = 3
then { end := 0; start := 0 }
return (ok)
```

What is the Specification?

What is the Specification?

- When execution starts,

  each buffer location is not read
  until it's been written

What is the Specification?

- When execution starts,

  each buffer location is not read
  until it's been written

- Always,

  when a buffer location is read
  it will not be read again
  until it's been written

What is the Specification?

- When execution starts,

  each buffer location is not read
  until it's been written

- Always,

  when a buffer location is read
  it will not be read again
  until it's been written

- Always,

  when a buffer location is written
  it will not be written again
  until it's been read

Propositional Linear Temporal Logic

AKA: LTL

Suppose given $\mathcal{L}$, a set of propositional atoms (finite or infinite)
The formulas $\mathcal{F}(\mathcal{L})$ of LTL over $\mathcal{L}$ are inductively defined:

- $A \in \mathcal{L}$ implies $A \in \mathcal{F}(\mathcal{L})$
- Suppose $\alpha, \beta \in \mathcal{F}(\mathcal{L})$. Then

$$\begin{aligned} \neg\alpha &\in \mathcal{F}(\mathcal{L}) \\ \alpha \vee \beta &\in \mathcal{F}(\mathcal{L}) \\ \mathsf{X}\alpha &\in \mathcal{F}(\mathcal{L}) \\ \mathsf{G}\alpha &\in \mathcal{F}(\mathcal{L}) \\ \alpha \mathsf{U} \beta &\in \mathcal{F}(\mathcal{L}) \end{aligned}$$