CS 181u Applied Logic

Lecture 11
Today’s class

Quick Review
LTL and CTL

Verifying properties of a stack
Modeling, specifying, and verifying stack properties.
Linear Temporal Logic (LTL) Review

Some paths of $M$

$M \models \phi \iff \forall \pi \ [\pi \models \phi]$

LTL Model Checking
Computation Tree Logic (CTL) expresses properties of “alternative timelines”.

\[ M \models \phi \iff \forall s \in I \ s \models \phi \]

CTL Model Checking
HW questions?
LTL questions?
CTL questions?
νSMV questions?
The Big Picture

Reactive System Code satisfies Requirements

Transition System satisfies Temporal Logic Formula

Model Checking
Can we think of a data structure as a reactive system?

- **Reactive System Code** satisfies $\models$ **Requirements**
- **Transition System** satisfies $\models$ **Temporal Logic Formula $\phi$**

The Big Picture
Data Structures as Reactive Systems

User / Client

- insert(x)
- get(x)
- empty?

Data Structure

5
FULL
Both the user / client and data structure:

- Internal state that changes over time (temporal aspects).
- Indefinite lifetime, “runs” forever.
Modeling a Stack

```
push(S, x)
y = pop(S)
```

```
y = 5
FULL / EMPTY
```
A stack has a finite buffer with integer indices and a pointer to the top of the stack.
A stack has a finite buffer with integer indices and a pointer to the top of the stack.

The stack pointer, top, indicates the next available place to store a value.

Pushing increments top, popping decrements top.
A stack has a finite buffer with integer indices and a pointer to the top of the stack.

The stack pointer, top, indicates the next available place to store a value.

Pushing increments top, popping decrements top.

When top = 0, stack is empty.
A stack has a finite buffer with integer indices and a pointer to the top of the stack.

The stack pointer, top, indicates the next available place to store a value.

Pushing increments top, popping decrements top.

When top = 0, stack is empty.

Attempting to pop an empty stack returns NULL.
Modeling a Stack

A stack has a finite **buffer** with integer indices and a pointer to the **top** of the stack.

The stack pointer, top, indicates the next available place to store a value.

Pushing increments top, popping decrements top.

When top = 0, stack is empty.

Attempting to pop an empty stack returns NULL.

When top = SIZE, stack is full.

Attempting to push when stack is full does nothing.
Modeling a Stack

An example execution

buffer

top →

0

1

2

3

4

5
Modeling a Stack

An example execution

push 31
Modeling a Stack

An example execution

push 31
push 42
Modeling a Stack

An example execution

push 31
push 42
push 99

buffer

top ->

31
42
99
5
Modeling a Stack

An example execution

push 31
push 42
push 99
v = pop    v = 99

buffer

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>top</td>
<td>99</td>
<td>42</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
An example execution

- push 31
- push 42
- push 99
- \( v = \text{pop} \) \( v = 99 \)
- push 81
Modeling a Stack

An example execution

push 31
push 42
push 99
v = pop   v = 99
push 81
push 15
Modeling a Stack

An example execution

push 31
push 42
push 99
v = pop  v = 99
push 81
push 15
push 50

buffer

50 4
top
15 3
81 2
42 1
31 0
An example execution

push 31
push 42
push 99
v = pop  v = 99
push 81
push 15
push 50
push 25  nothing happens
Modeling a Stack

An example execution

push 31  
push 42  
push 99  
v = pop \triangleright v = 99
push 81  
push 15  
push 50  
push 25  \triangleright nothing happens
v = pop \triangleright v = 50
v = pop \triangleright v = 15
v = pop \triangleright v = 81
v = pop \triangleright v = 42
v = pop \triangleright v = 31
v = pop \triangleright v = NULL
Modeling a Stack in νSMV

What do the states and transitions look like?
Modeling a Stack in νSMV

What do the states and transitions look like?

<table>
<thead>
<tr>
<th>buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>top = 2</td>
</tr>
<tr>
<td>action = pop</td>
</tr>
<tr>
<td>pop_val = 15</td>
</tr>
<tr>
<td>push_val = 50</td>
</tr>
</tbody>
</table>

action = pop
pop_val = 15
push_val = 50
Modeling a Stack in $\nu$SMV

What do the states and transitions look like?

buffer

<table>
<thead>
<tr>
<th>top</th>
<th>action</th>
<th>push_val</th>
<th>pop_val</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>pop</td>
<td>50</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>push</td>
<td>81</td>
<td>42</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>

buffer

<table>
<thead>
<tr>
<th>top</th>
<th>action</th>
<th>push_val</th>
<th>pop_val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>push</td>
<td>11</td>
<td>81</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

buffer

<table>
<thead>
<tr>
<th>top</th>
<th>action</th>
<th>push_val</th>
<th>pop_val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modeling a Stack in $\nu$SMV

What do the states and transitions look like?

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Buffer</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 4</td>
<td>50 4</td>
<td>50 4</td>
</tr>
<tr>
<td>15 3</td>
<td>15 3</td>
<td>15 3</td>
</tr>
<tr>
<td>81 2</td>
<td>81 2</td>
<td>81 2</td>
</tr>
<tr>
<td>42 1</td>
<td>42 1</td>
<td>42 1</td>
</tr>
<tr>
<td>31 0</td>
<td>31 0</td>
<td>31 0</td>
</tr>
<tr>
<td>top = 2</td>
<td>top = 1</td>
<td>top = 2</td>
</tr>
<tr>
<td>action = pop</td>
<td>action = push</td>
<td>action = push</td>
</tr>
<tr>
<td>pop_val = 15</td>
<td>pop_val = 81</td>
<td>pop_val = 81</td>
</tr>
<tr>
<td>push_val = 50</td>
<td>push_val = 11</td>
<td>push_val = 97</td>
</tr>
</tbody>
</table>
Modeling a Stack in νSMV

What do the states and transitions look like?

Our νSMV model should define all possible state transitions.
Modeling a Stack in νSMV

Overview
Modeling a Stack in νSMV

Overview

\[ \text{action} = \{ \text{push, pop} \} \]
Modeling a Stack in νSMV

Overview

\[ \text{action} = \{ \text{push, pop} \} \]

\[ \text{next}(\text{top}) := \text{increment or decrement depending on push or pop. Keep top in the correct range.} \]
Modeling a Stack in $\nu$SMV

Overview

$$\text{action} = \{ \text{push, pop} \}$$

$$\text{next}(\text{top}) := \text{increment or decrement depending on push or pop. Keep top in the correct range.}$$

$$\text{next}(\text{pop\_val}) := \text{the value under the stack pointer if the current action is pop and the stack is not empty. Otherwise NULL.}$$
Modeling a Stack in νSMV

Overview

\[ \text{action} = \{ \text{push, pop} \} \]

\[ \text{next(top)} := \text{increment or decrement depending on push or pop. Keep top in the correct range.} \]

\[ \text{next(pop\_val)} := \text{the value under the stack pointer if the current action is pop and the stack is not empty. Otherwise NULL.} \]

\[ \text{next(buffer)} := \text{update a location in the buffer based on top if action = push. If action = pop, no update.} \]
Modeling a Stack in $\nu$SMV
#define SIZE 5

MODULE main
  VAR
    pop_val : {NULL, x, y, z};
    push_val : {x, y, z};
    action : {push, pop};
  s : stack(action, push_val, pop_val);
#define SIZE 5

MODULE main
  VAR
    pop_val : {NULL, x, y, z};
    push_val : {x, y, z};
    action : {push, pop};
    s : stack(action, push_val, pop_val);

Why \{x, y, z\}?
Modeling a Stack in νSMV

#define SIZE 5

MODULE main
  VAR
    pop_val : {NULL, x, y, z};
    push_val : {x,y,z};
    action : {push, pop};
    s : stack(action, push_val, pop_val);

Why \{x, y, z\}?  
We have abstracted away the type of the stack.  
This abstraction will allow us to make  
statements about three distinct values in the  
stack, without worrying about what they are.
MODULE stack(action, push_val, pop_val)
    VAR
        top : 0 .. SIZE;
        buffer : array 0 .. SIZE - 1 of {NULL, x, y, z};
    
    DEFINE
        full := top = SIZE;
        empty := top = 0;
    
    ASSIGN
        init(top) := 0;

        next(top) :=
            case
                (action = push) & (top < SIZE) : top + 1;
                (action = pop) & (top > 0) : top - 1;
            end;
            TRUE : top;
        esac;

        next(pop_val) :=
            case
                action = pop & ! empty : buffer[top];
                TRUE : NULL;
            esac;
Modeling a Stack inνSMV

How to update the state of the buffer?
We’d like to write something like next(buffer) := ??

In νSMV, we have to update individual array elements.
Modeling a Stack in \( \nu \text{SMV} \)

How to update the state of the buffer?

We’d like to write something like \( \text{next(buffer)} := ?? \)

In \( \nu \text{SMV} \), we have to update individual array elements.

\[
\text{next(buffer[3]) := }
\begin{cases}
\text{conditional test} & \text{then-exp} \\
\text{else-exp}
\end{cases}
\]
Modeling a Stack in $\nu$SMV

How to update the state of the buffer?
We’d like to write something like $\text{next(buffer)} := ??$

In $\nu$SMV, we have to update individual array elements.

\[
\text{next(buffer[3]) := action = push \& top = 3 \ ? \ push\_val : \ buffer[3]}
\]

\[
\text{conditional test \ \ \ then-exp \ \ else-exp}
\]
Modeling a Stack in νSMV

How to update the state of the buffer?
We’d like to write something like \texttt{next(buffer) := ??}
In νSMV, we have to update individual array elements.

\texttt{next(buffer[3]) := action = push \& top = 3 \ ? push\_val : buffer[3]}
  \hspace{1cm}
  \texttt{conditional test \hspace{1cm} then-exp \hspace{1cm} else-exp}
Modeling a Stack in νSMV

How to update the state of the buffer? We’d like to write something like $\text{next(buffer)} := ??$

In νSMV, we have to update individual array elements.

\[
\text{next(buffer[3]) := action = push & top = 3 ? push_val : buffer[3]}
\]

conditional test then-exp else-exp

\[
\text{next(buffer[0]) := top = 0 & action = push ? push_val : buffer[0];}
\]
\[
\text{next(buffer[1]) := top = 1 & action = push ? push_val : buffer[1];}
\]
\[
\text{next(buffer[2]) := top = 2 & action = push ? push_val : buffer[2];}
\]
\[
\text{next(buffer[3]) := top = 3 & action = push ? push_val : buffer[3];}
\]
\[
\text{next(buffer[4]) := top = 4 & action = push ? push_val : buffer[4];}
\]
Stack Properties to Verify

$G(0 \leq s\text{.top} \land s\text{.top} \leq \text{SIZE})$

LTL property: the stack pointer is always in the correct range.
Stack Properties to Verify

\[ G(0 \leq s.\text{top} \land s.\text{top} \leq \text{SIZE}) \]

LTL property: the stack pointer is always in the correct range.

\[ G \neg(s.\text{full} \land s.\text{empty}) \]

LTL property: the stack is never empty and full at the same time.
Stack Properties to Verify

\[ G(0 \leq s.\text{top} \land s.\text{top} \leq \text{SIZE}) \]

LTL property: the stack pointer is always in the correct range.

\[ G \neg(s.\text{full} \land s.\text{empty}) \]

LTL property: the stack is never empty and full at the same time.

\[ AG (s.\text{full} \rightarrow EF s.\text{empty}) \]
Stack Properties to Verify

\[ G(0 \leq s.\text{top} \land s.\text{top} \leq \text{SIZE}) \]

LTL property: the stack pointer is always in the correct range.

\[ G \neg(s.\text{full} \land s.\text{empty}) \]

LTL property: the stack is never empty and full at the same time.

\[ AG (s.\text{full} \rightarrow EF s.\text{empty}) \]

CTL property: For all possible system states, if the stack is full, then it is possible that the stack is eventually empty.
Stack Properties to Verify

The most important property of a stack:
Stack Properties to Verify

The most important property of a stack:

Last In, First Out (LIFO)

This will be part of your next HW.
Coming up with specifications
Coming up with specifications

Karl Popper: The Logic of Scientific Discovery

“\textit{My proposal is based on an asymmetry between verifiability and falsifiability; an asymmetry which results from the logical form of universal statements. For these are never derivable from singular statements, but can be contradicted by singular statements.}"

Edsger Dijkstra

\textit{Program testing can be used to show the presence of bugs, but never to show their absence!}
Coming up with specifications

Coming up with specifications


Literary theorists have long recognized the trade-offs in optimistic and pessimistic thinking through utopias and dystopias.

Research suggests that scientists are overwhelmingly optimistic, and subject to the effect of optimism bias [1].

Software engineering researchers have a tendency to be optimistic.

Though useful, optimism bias bolsters unrealistic expectations towards desirable outcomes.

Framing software engineering research through dystopias mitigates optimism bias and engender more diverse and thought-provoking research directions.

In class activity:

1. Come up with one or two interesting stack properties that would be important to verify. Write it down as a legible English sentence.

2. In a group of two or three, swap properties. Identify if the property is LTL or CTL. Translate the property to LTL or CTL.

3. Regroup and discuss your properties and translations.

4. Choose one to write on the board (both in English and CTL / LTL) to explain to the rest of the class.
Coming up with specifications

Some hints:

What could go wrong? Negate that property.

What should go right? Assert that property.

What should happen if \texttt{push} \texttt{x} is followed directly by \texttt{pop}?

What should happen if we try to pop an empty stack?

Our examples from earlier:

\[ G(0 \leq \texttt{s.top} \land \texttt{s.top} \leq \texttt{SIZE}) \]
\[ G \neg (\texttt{s.full} \land \texttt{s.empty}) \]
\[ AG (\texttt{s.full} \rightarrow EF \texttt{s.empty}) \]
Future homework

Translate and verify some stack properties.

Model and verify a queue.
Equivalence of properties

Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?
Equivalence of properties

Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?

We say that $\alpha \equiv \beta$ iff
Equivalence of properties

Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?

We say that $\alpha \equiv \beta$ iff

$$\forall M \ (M \models \alpha \iff M \models \beta)$$
Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?

We say that $\alpha \equiv \beta$ iff

$$\forall M \ (M \models \alpha \iff M \models \beta)$$

We say that $\alpha \not\equiv \beta$ iff
Equivalence of properties

Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?

We say that $\alpha \equiv \beta$ iff

$$\forall M \ (M \models \alpha \iff M \models \beta)$$

We say that $\alpha \not\equiv \beta$ iff

$$\exists M \ ((M \models \alpha \land M \not\models \beta) \lor (M \not\models \alpha \land M \models \beta))$$
Equivalence of properties

Given two temporal logic formulas $\alpha$ and $\beta$, when can we say that the two formulas are equivalent?

We say that $\alpha \equiv \beta$ iff

$$\forall M \ (M \models \alpha \iff M \models \beta)$$

We say that $\alpha \not\equiv \beta$ iff

$$\exists M \ ((M \models \alpha \land M \not\models \beta) \lor (M \not\models \alpha \land M \models \beta))$$

In words, two formulas are not equivalent if we can find a transition system that satisfies one formula but not the other.
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \, G \, p$ \hspace{1cm} $AF \, AG \, p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \; G \; p$  \hspace{2cm}  $AF \; AG \; p$

Consider this transition system, $\mathcal{M}$:
Showing that $\alpha \nRightarrow \beta$

Consider these two temporal formulas

$F \ G \ p$  \hspace{1cm}  $AF \ AG \ p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*12^\omega$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

\[
\begin{align*}
F & \quad G & \quad p \\
\neg & \quad F & \quad G & \quad p \\
AF & \quad AG & \quad p
\end{align*}
\]

Consider this transition system, $\mathcal{M}$:

![Transition System Diagram]

Paths of $\mathcal{M}$ look like:

\[0^\omega \quad \text{or} \quad 0^*12^\omega\]

Sequences of propositions:

\[
\begin{align*}
p, p, p, p, p, p, & \ldots \\
p, p, p, & \ldots, \neg p, p, p, p, p, & \ldots
\end{align*}
\]
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$$F \ G \ p \quad \quad AF \ AG \ p$$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$$0^\omega \quad \text{or} \quad 0^*1 \ 2^\omega$$

Sequences of propositions:

$$p, p, p, p, p, \ldots$$

$$p, p, p, \ldots, \neg p, p, p, p, \ldots$$

$$\mathcal{M} \models F \ G \ p$$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas:

$$F \ G \ p$$

$$AF \ AG \ p$$

Consider this transition system, $\mathcal{M}$:

Computation tree:

Paths of $\mathcal{M}$ look like:

$$0^\omega \quad \text{or} \quad 0^*1 \ 2^\omega$$

Sequences of propositions:

$$p, p, p, p, p, \ldots$$

$$p, p, p, \ldots, \neg p, p, p, p, \ldots$$

$$\mathcal{M} \models F \ G \ p$$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \ G \ p$

$AF \ AG \ p$

Consider this transition system, $\mathcal{M}$:

\[
\begin{array}{ccc}
0 & \rightarrow & 1 \\
\downarrow & & \downarrow \\
1 & \rightarrow & 2
\end{array}
\]

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1 \ 2^\omega$

Sequences of propositions:

$p, p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, p, p, \ldots$

$\mathcal{M} \models F \ G \ p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \; G \; p$

$AF \; AG \; p$

Consider this transition system, $\mathcal{M}$:

\[\begin{array}{c}
\begin{array}{ccc}
0 & \rightarrow & 1 \\
\rightarrow & & \\
1 & \rightarrow & 2 \\
\rightarrow & & \\
2 & \rightarrow & 0 \\
\rightarrow & & \\
0 & \rightarrow & 1 \\
\rightarrow & & \\
1 & \rightarrow & 2 \\
\rightarrow & & \\
2 & \rightarrow & 0 \\
\rightarrow & & \\
0 & \rightarrow & 2 \\
\rightarrow & & \\
\end{array}
\end{array}\]

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\,2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \; G \; p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \; G \; p$

$AF \; AG \; p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\;2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \; G \; p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas $F \ G \ p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\ 2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \ G \ p$

$AF \ AG \ p$

Computation tree:
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \ G \ p$

$AF \ AG \ p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\ 2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \ G \ p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \ G \ p$

$AF \ AG \ p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\ 2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \ G \ p$
Showing that $\alpha \not\equiv \beta$ 

Consider these two temporal formulas

$F \ G \ p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega$ or $0^*1\ 2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \ G \ p$

$\mathcal{M} \not\models AF \ AG \ p$
Showing that $\alpha \not\equiv \beta$

Consider these two temporal formulas

$F \; G \; p$

$AF \; AG \; p$

Consider this transition system, $\mathcal{M}$:

Paths of $\mathcal{M}$ look like:

$0^\omega \text{ or } 0^* 1^* 2^\omega$

Sequences of propositions:

$p, p, p, p, p, \ldots$

$p, p, p, \ldots, \neg p, p, p, p, \ldots$

$\mathcal{M} \models F \; G \; p$

$\mathcal{M} \not\models AF \; AG \; p$

On your HW: Show two formulas not equivalent.