

CS 273 and Real Life

```
M0: main() {  
M1:   if(flag)  
      mumbo(n);  
  
M2:   n = jumbo(n);  
M3:   if (n <= 55) goto m2;  
      }
```

Behavior: Segmentation fault

CS 273 and Real Life

```
M0: main() {  
M1:   if(flag)  
      mumbo(n);  
  
M2:   n = jumbo(n);  
M3:   if (n <= 55) goto m2;  
      }
```

Behavior: Segmentation fault

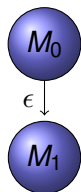


Call-return test

```
M0: main() {  
M1:   if(flag) {  
        print "mumbo"; mumbo(n); print "ret";  
    }  
M2:   print "jumbo"; n = jumbo(n); print "ret";  
M3:   if (n <= 55) goto m2;  
    }
```

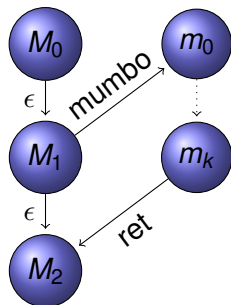
Behavior: mumbo ret jumbo... Segmentation fault

Transform program...



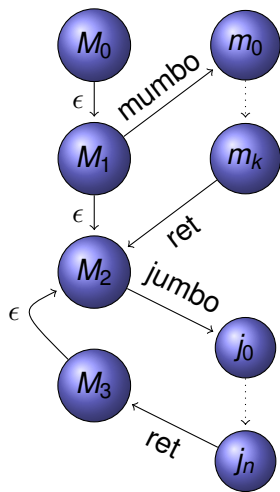
```
M1:  if(flag) {  
      print "mumbo"; mumbo(n); print "ret";  
    }  
M2:  print "jumbo"; n = jumbo(n); print "ret";  
M3:  if (n <= 55) goto m2;  
    }
```

...into an *NFA* model...

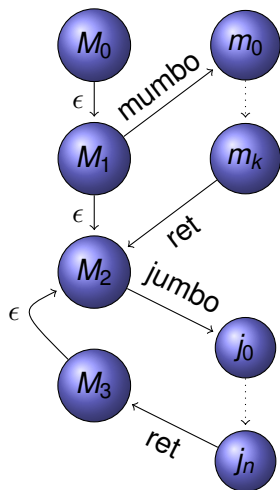


```
M2:  print "jumbo"; n = jumbo(n); print "ret";  
M3:  if (n <= 55) goto m2;  
    }
```

... that captures behavior

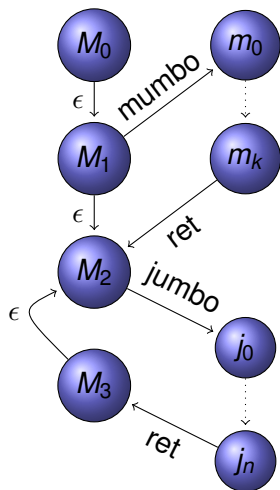


... that captures behavior



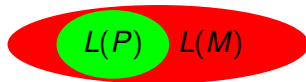
Every possible **program** behavior is also a **model** behavior

... that captures behavior

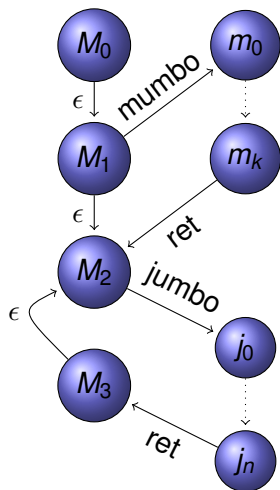


Every possible **program** behavior is also a **model** behavior

Model may have *more* behaviors (inexact)

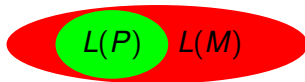


... that captures behavior



Every possible **program** behavior is also a **model** behavior

Model may have *more* behaviors (inexact)



Specification:

$((\text{mumbo} + \text{jumbo}) \text{ret})^*$

Model Checking

Program model M is a generator of possible behaviors

Specification S defines the set of “good” behaviors

Model Checking

Program model M is a generator of possible behaviors

Specification S defines the set of “good” behaviors

Verification task: Is $L(M) \subseteq S$?



Model Checking

Program model M is a generator of possible behaviors

Specification S defines the set of “good” behaviors

Verification task: Is $L(M) \subseteq S$?



- ▶ If so, all program behaviors are “good”

Model Checking

Program model M is a generator of possible behaviors

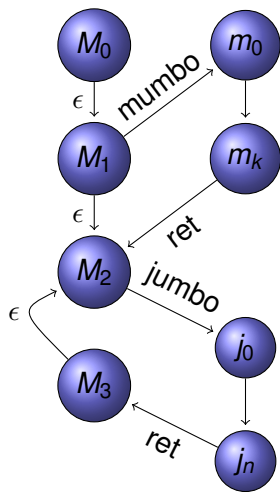
Specification S defines the set of “good” behaviors

Verification task: Is $L(M) \subseteq S$?



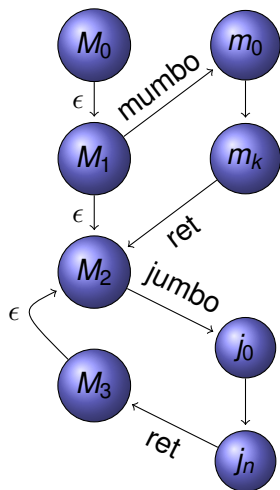
- ▶ If so, all program behaviors are “good”
- ▶ If not, **either** a counter-example or a “false positive”

Complex programs



Specification:

More powerful models



Specification: Each mumbo, jumbo has a **matching** ret

(CFL, *not* regular)

Model Checking with PDAs

Program model M is a **PDA**

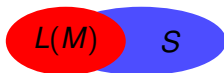
Specification S (“good” behaviors) is a **CFL**

Model Checking with PDAs

Program model M is a **PDA**

Specification S (“good” behaviors) is a **CFL**

Verification task: Is $L(M) \subseteq S$?



Model Checking with PDAs

Program model M is a **PDA**

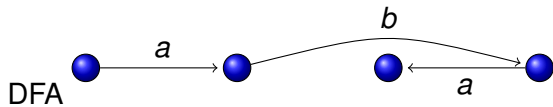
Specification S (“good” behaviors) is a **CFL**

Verification task: Is $L(M) \subseteq S$?

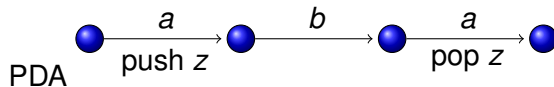
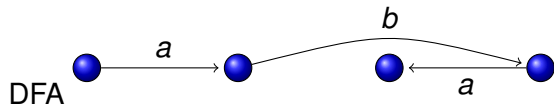


$L(M) \subseteq S$ if and only if $L(M) - S = \emptyset$

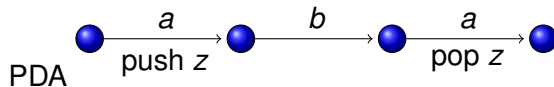
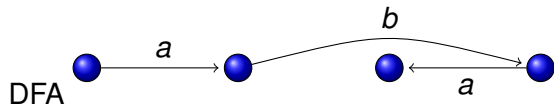
Why is $\text{PDA} \times \text{DFA}$ a PDA?



Why is PDA \times DFA a PDA?



Why is PDA \times DFA a PDA?



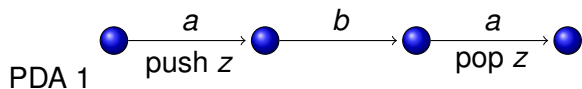
Proof: Finger-waving.

Finger-waving proofs

Why is $\text{PDA} \times \text{PDA}$ **not** always a PDA?

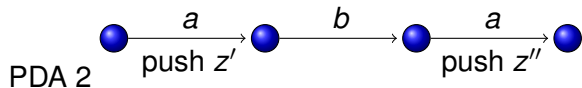
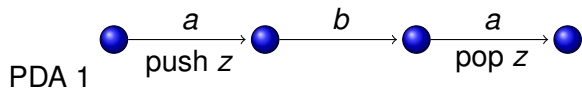
Finger-waving proofs

Why is $\text{PDA} \times \text{PDA}$ **not** always a PDA?



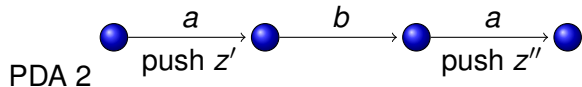
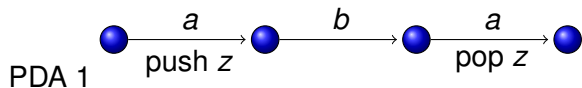
Finger-waving proofs

Why is $\text{PDA} \times \text{PDA}$ **not** always a PDA?



Finger-waving proofs

Why is $\text{PDA} \times \text{PDA}$ **not** always a PDA?



PDA with **two** stacks equals Turing machine!

Synchronize on pushes and pops

Partition input alphabet into:

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Makes pushes (calls) and pops (returns) **visible**

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Makes pushes (calls) and pops (returns) **visible**

Visibly Pushdown Automata (VPA)

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Makes pushes (calls) and pops (returns) **visible**

Visibly Pushdown Automata (VPA), Languages (VPL)

Synchronize on pushes and pops

Partition input alphabet into:

- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Makes pushes (calls) and pops (returns) **visible**

Visibly Pushdown Automata (VPA), Languages (VPL)

- ▶ VPL closed under complement, intersection, union, *

Synchronize on pushes and pops

Partition input alphabet into:

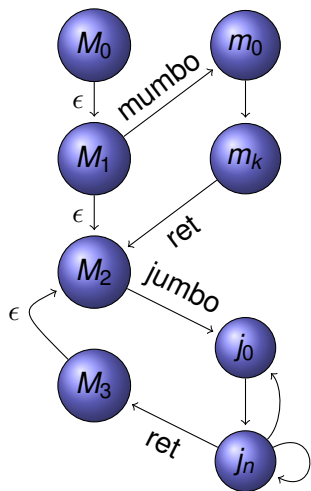
- ▶ **push** symbols: always *push* one stack symbol
- ▶ **pop** symbols: always *pop* one stack symbol
- ▶ **internal** symbols: no stack operation

Makes pushes (calls) and pops (returns) **visible**

Visibly Pushdown Automata (VPA), Languages (VPL)

- ▶ VPL closed under complement, intersection, union, *
- ▶ Algorithm for $L(M) \subseteq S$

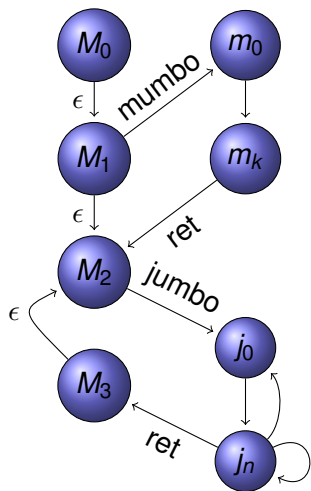
VPA models



Specification: Each mumbo, jumbo has a matching `ret`

Visibly Pushdown Language

VPA models

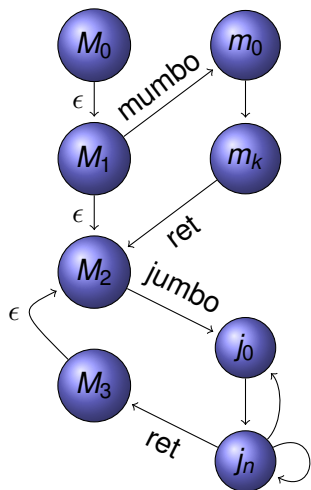


Specification: Each mumbo, jumbo has a matching `ret`

Visibly Pushdown Language

XML: text-based format for structured information

VPA models



Specification: Each mumbo, jumbo has a matching `ret`

Visibly Pushdown Language

XML: text-based format for structured information

- ▶ For each opening `<tag>`, there is a closing `</tag>`