Using Isabelle to Help Verify Code that Uses Abstract Data Types

Jason Kirschenbaum
The Ohio State University

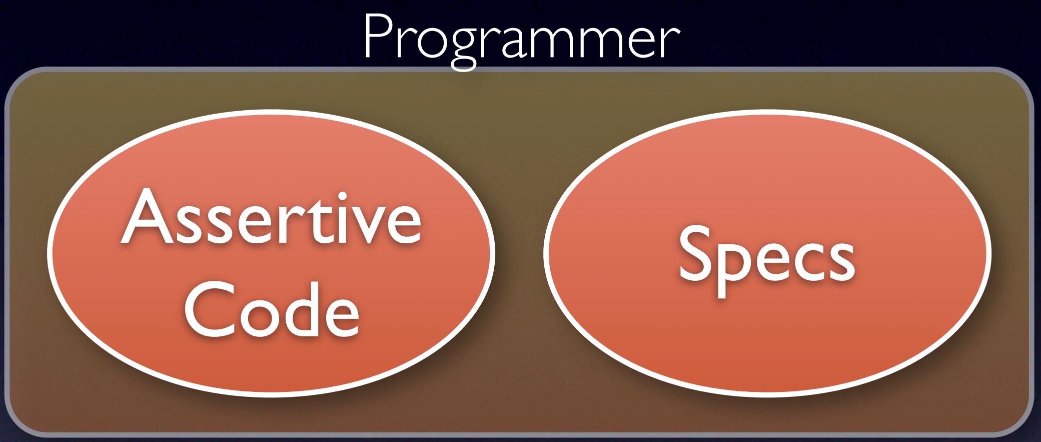
Bruce M. Adcock   Derek Bronish   Paolo Bucci   Bruce W. Weide
The Ohio State University

The Grand Challenge

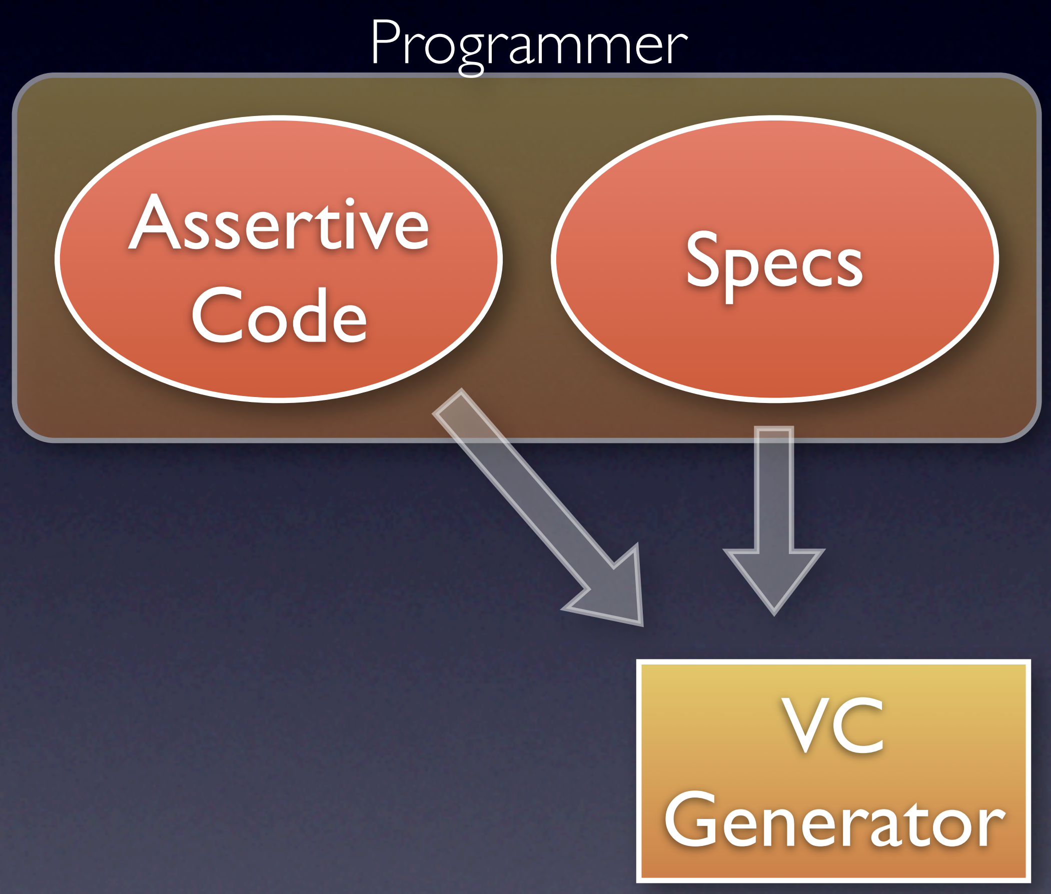
> “I revive an old challenge: the construction and application of a verifying compiler that guarantees correctness of a program before running it.”

Tony Hoare, The Verifying Compiler: A Grand Challenge
for Computing Research, 2003

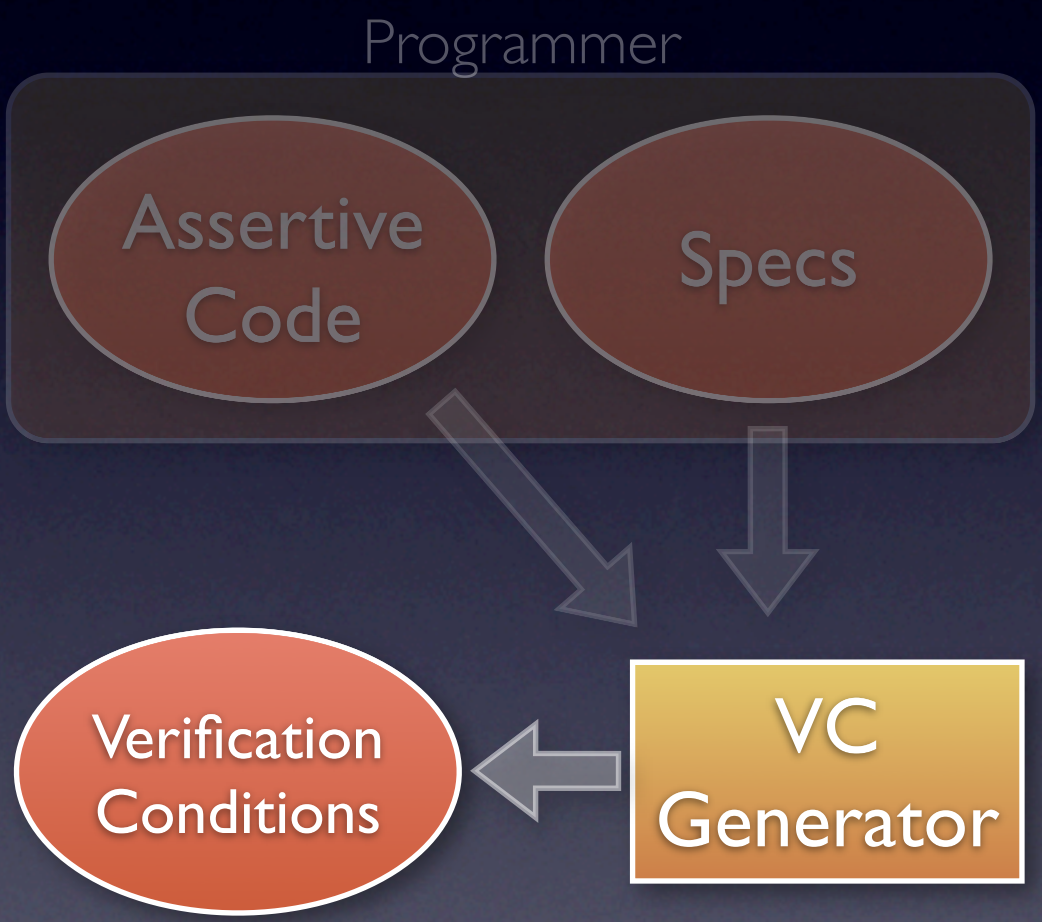
Organization of verification system



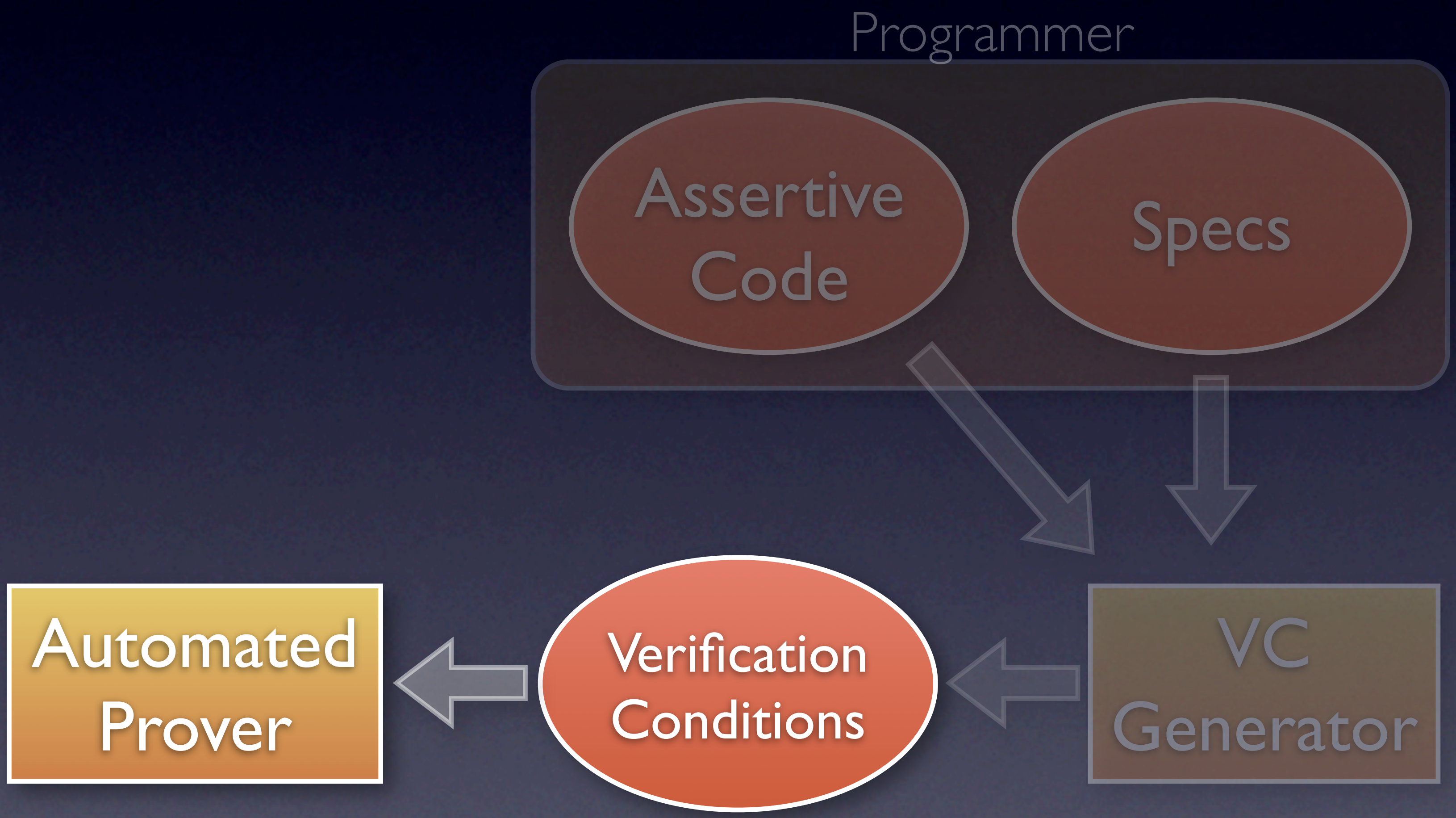
Organization of verification system



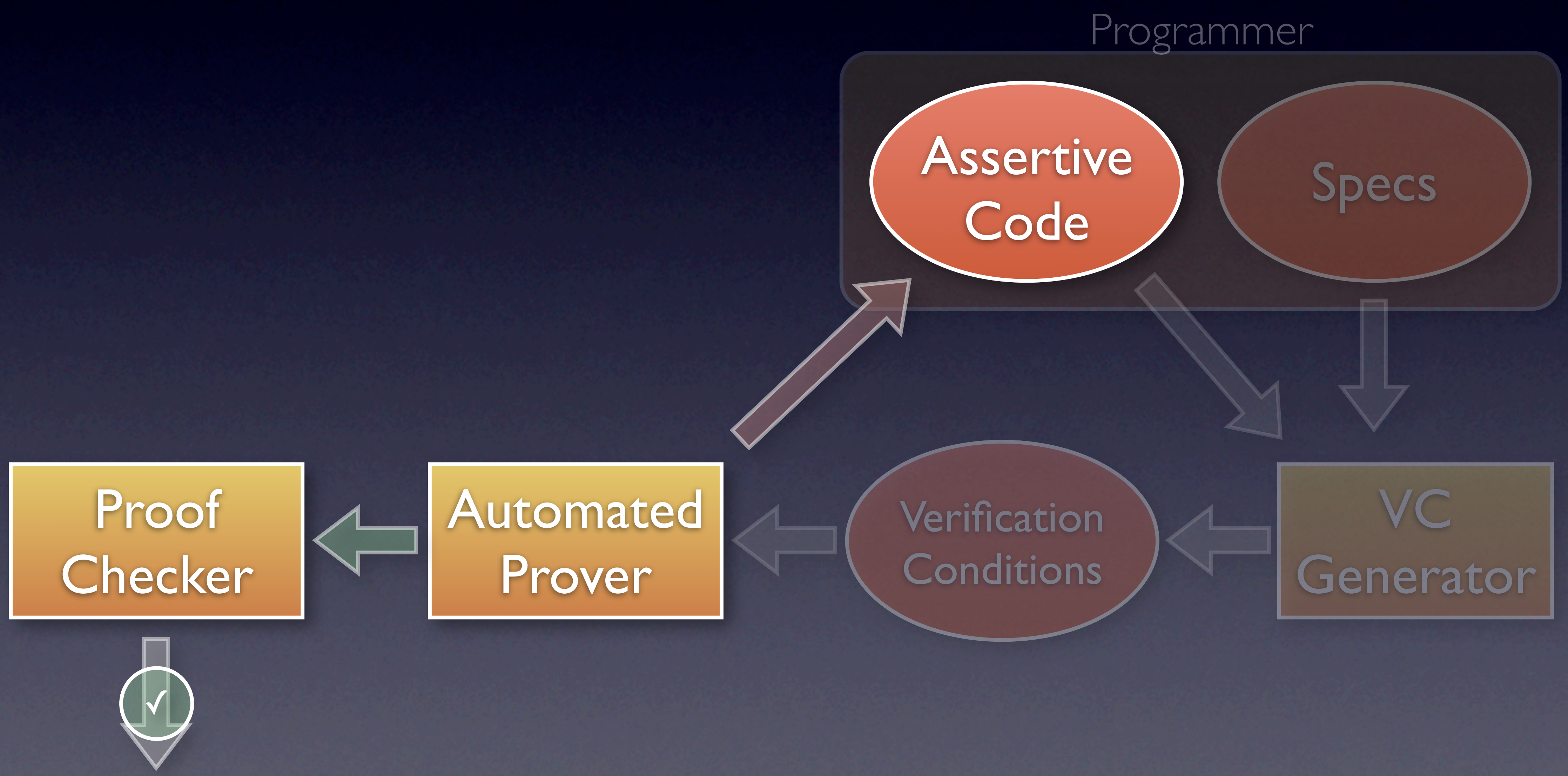
Organization of verification system



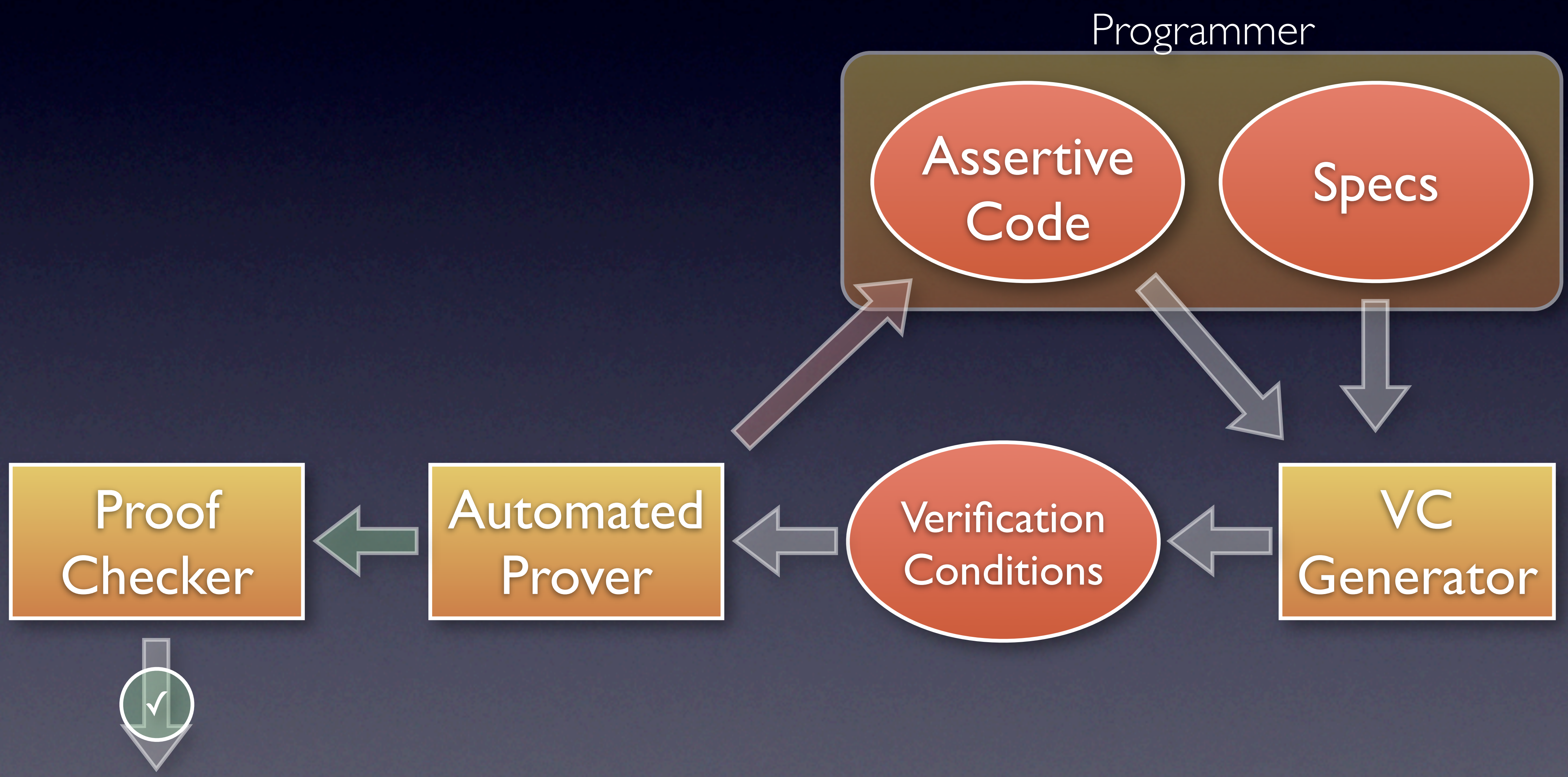
Organization of verification system



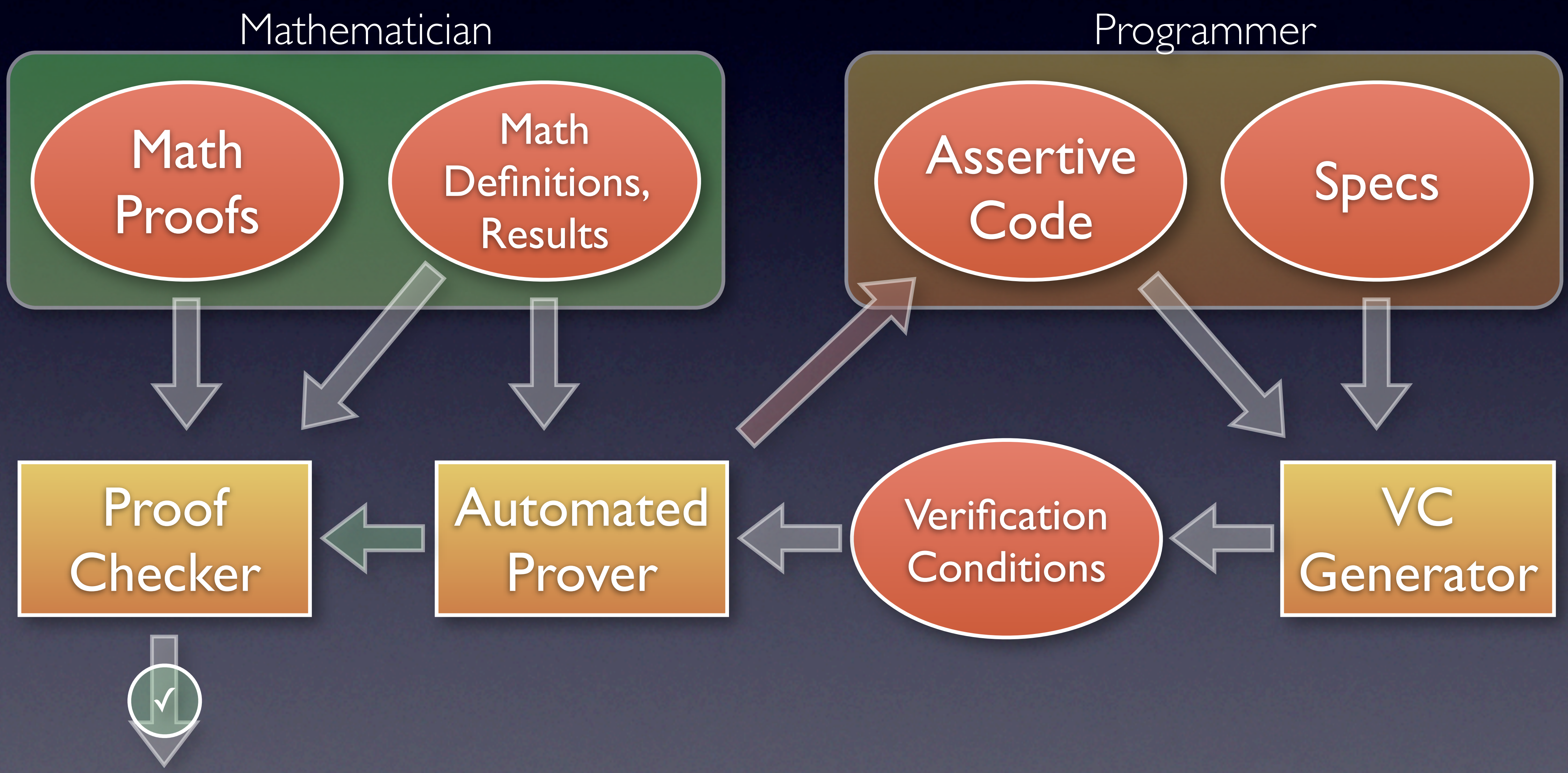
Organization of verification system



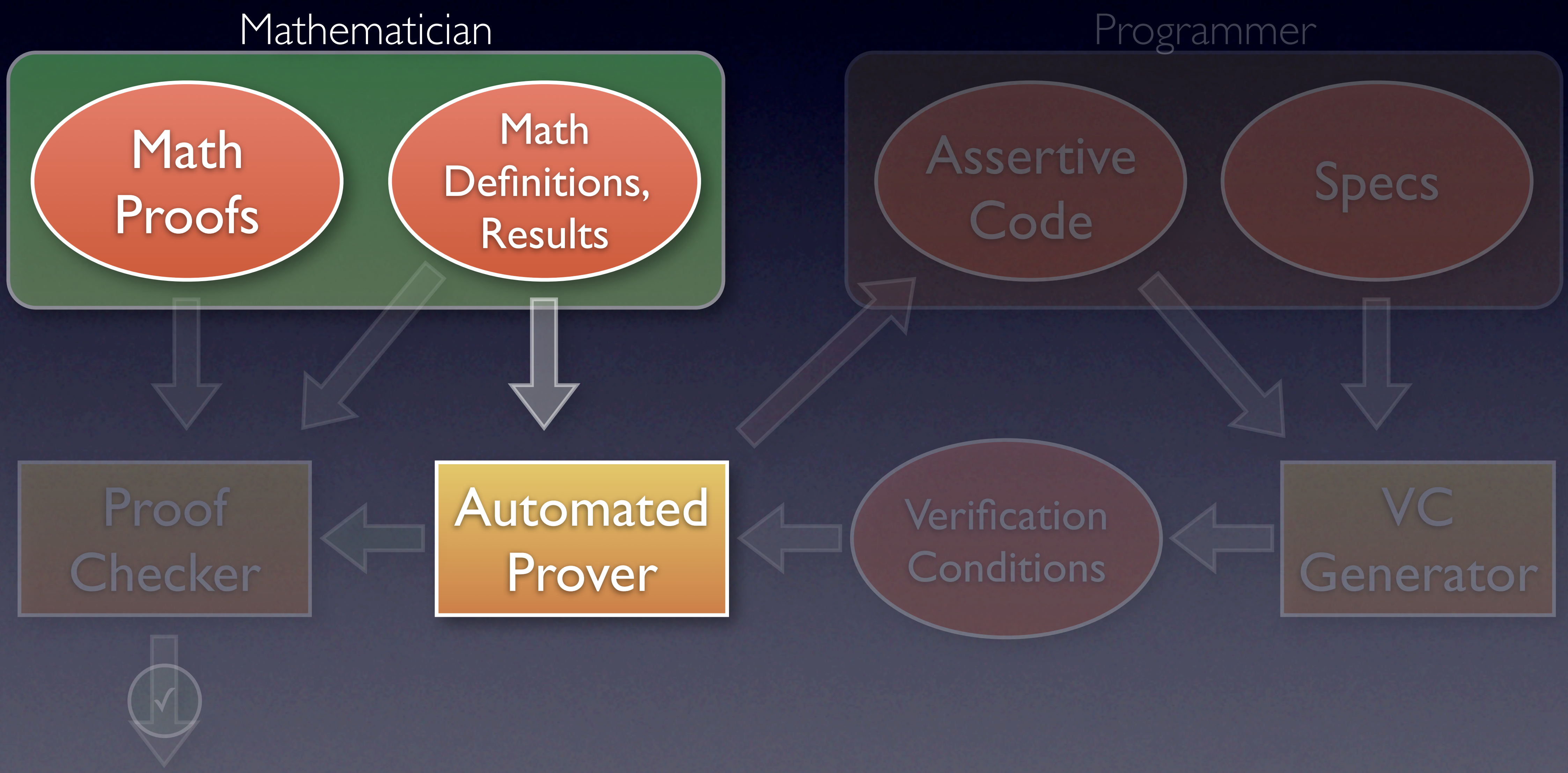
Organization of verification system



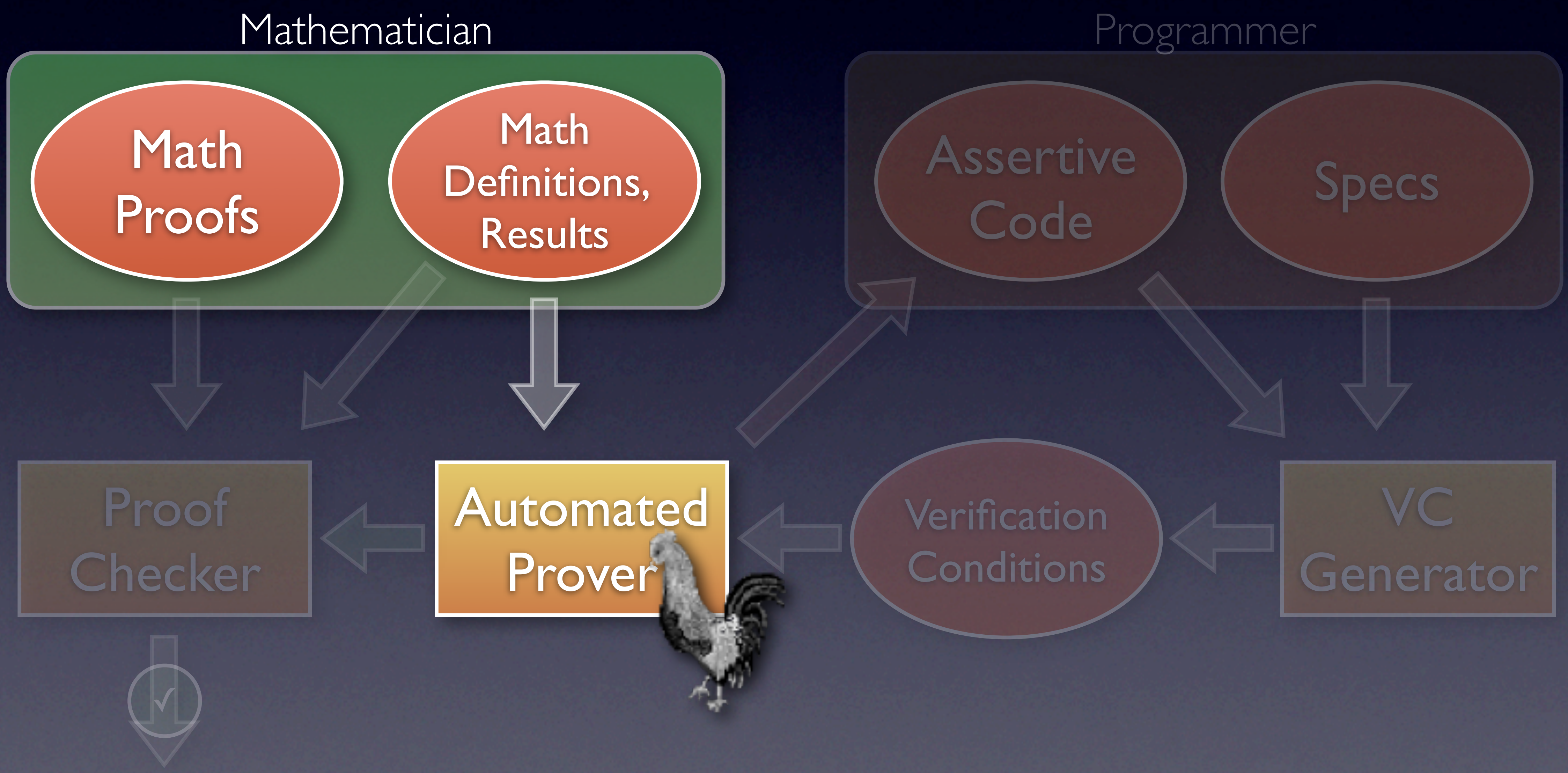
Organization of verification system



Organization of verification system



Organization of verification system



Mathematician

Math Proofs

Math Definitions, Results

Programmer

Assertive Code

Specs

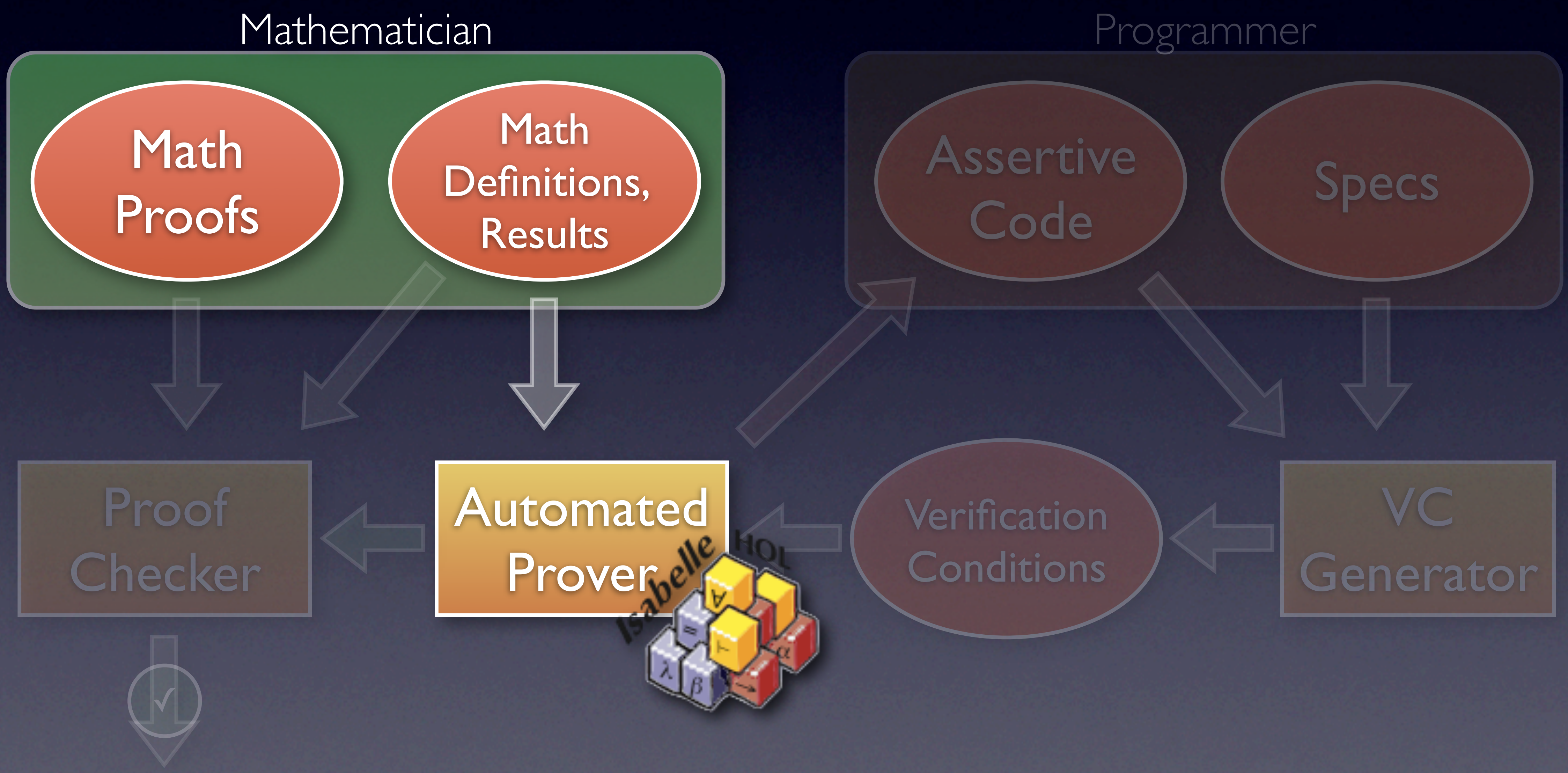
Proof Checker

Automated Prover

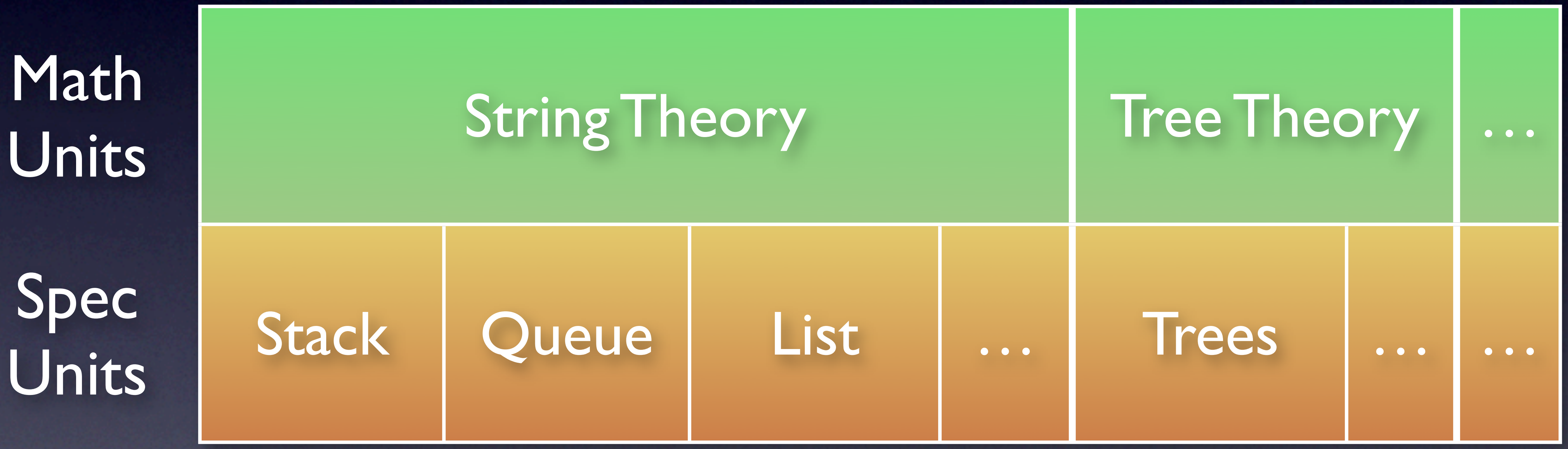
Verification Conditions

VC Generator

Organization of verification system



Mathematical Theory Reuse



Math Units	String Theory				Tree Theory		…
Spec Units	Stack	Queue	List	…	Trees	…	…

Research Question

- Can we use an off-the-shelf theorem prover to prove VCs automatically and still maintain the properties identified earlier?
- We explore this question via an extension to a queue ADT.

Queue Specification

```
contract QueueTemplate (type Item)

    math subtype QUEUE_MODEL is string of Item

    type Queue is modeled by QUEUE_MODEL
        exemplar q
        initialization ensures
            q = empty_string

    procedure Enqueue (updates q: Queue,
                       clears x: Item)
        ensures
            q = #q * <#x>

    procedure Dequeue (updates q: Queue,
                       replaces x: Item)
        requires
            q /= empty_string
        ensures
            #q = <x> * q

    function IsEmpty (restores q: Queue): control
        ensures
            IsEmpty = (q = empty_string)

end QueueTemplate
```

Specs

Queue Specification

```
contract QueueTemplate (type Item)

    math subtype QUEUE_MODEL is string of Item

    type Queue is modeled by QUEUE_MODEL
        exemplar q
        initialization ensures
            q = empty_string

    procedure Enqueue (updates q: Queue,
                       clears x: Item)
        ensures
            q = #q * <#x>

    procedure Dequeue (updates q: Queue,
                       replaces x: Item)
        requires
            q /= empty_string
        ensures
            #q = <x> * q

    function IsEmpty (restores q: Queue): control
        ensures
            IsEmpty = (q = empty_string)

end QueueTemplate
```

Specs

Queue Specification

```
contract QueueTemplate (type Item)

    math subtype QUEUE_MODEL is string of Item

    type Queue is modeled by QUEUE_MODEL
        exemplar q
        initialization ensures
            q = empty_string

    procedure Enqueue (updates q: Queue,
                       clears x: Item)
        ensures
            q = #q * <#x>

    procedure Dequeue (updates q: Queue,
                       replaces x: Item)
        requires
            q /= empty_string
        ensures
            #q = <x> * q

    function IsEmpty (restores q: Queue): control
        ensures
            IsEmpty = (q = empty_string)

end QueueTemplate
```

Specs

Queue Flip Specification

```
contract QueueFlip enhances QueueTemplate

    procedure Flip (updates q: Queue)
        ensures
            q = reverse (#q)

end QueueFlip
```

Specs

Iterative Implementation

```
realization Iterative implements QueueFlip

    facility StackFacility is StackTemplate (Item)

    procedure Flip (updates q: Queue)
        variable s: Stack
        loop
            maintains reverse(#s) * #q = reverse(s) * q
            decreases |q|
        while not IsEmpty (q) do
            variable x: Item
            Dequeue (q, x)
            Push (s, x)
        end loop
        loop
            maintains #q * #s = q * s
            decreases |s|
        while not IsEmpty (s) do
            variable x : Item
            Pop (s, x)
            Enqueue (q, x)
        end loop
    end Flip

end Iterative
```

Assertive Code

Iterative Implementation

```
realization Iterative implements QueueFlip

    facility StackFacility is StackTemplate (Item)

    procedure Flip (updates q: Queue)
        variable s: Stack
        loop
            maintains reverse(#s) * #q = reverse(s) * q
            decreases |q|
        while not IsEmpty (q) do
            variable x: Item
            Dequeue (q, x)
            Push (s, x)
        end loop
        loop
            maintains #q * #s = q * s
            decreases |s|
        while not IsEmpty (s) do
            variable x : Item
            Pop (s, x)
            Enqueue (q, x)
        end loop
    end Flip

end Iterative
```

Assertive Code

Iterative Implementation

```
realization Iterative implements QueueFlip

    facility StackFacility is StackTemplate (Item)

    procedure Flip (updates q: Queue)
        variable s: Stack
        loop
            maintains reverse(#s) * #q = reverse(s) * q
            decreases |q|
        while not IsEmpty (q) do
            variable x: Item
            Dequeue (q, x)
            Push (s, x)
        end loop
        loop
            maintains #q * #s = q * s
            decreases |s|
        while not IsEmpty (s) do
            variable x : Item
            Pop (s, x)
            Enqueue (q, x)
        end loop
    end Flip

end Iterative
```

Assertive Code

Iterative Implementation

```
realization Iterative implements QueueFlip

    facility StackFacility is StackTemplate (Item)

    procedure Flip (updates q: Queue)
        variable s: Stack
        loop
            maintains reverse(#s) * #q = reverse(s) * q
            decreases |q|
        while not IsEmpty (q) do
            variable x: Item
```

```
loop
    maintains reverse(#s) * #q = reverse(s) * q
    decreases |q|
```

```
            variable x : Item
            Pop (s, x)
            Enqueue (q, x)
        end loop
    end Flip

end Iterative
```

Assertive Code

Recursive Implementation

```
realization Recursive implements QueueFlip

    procedure Flip (updates q: Queue)
        decreases |q|
        if not IsEmpty (q) then
            variable x: Item
            Dequeue (q, x)
            Flip (q)
            Enqueue (q, x)
        end if
    end Flip

end Recursive
```

Assertive Code

A VC for Recursive Implementation

is_initial (x_2)

is_initial (x_5)

$\langle x_3 \rangle \circ q_3 \neq \lambda$

$\text{reverse}\ (q_3) \circ \langle x_3 \rangle = \text{reverse}\ (\langle x_3 \rangle \circ q_3)$

Verification Conditions

Theory Libraries

- The required theorem was already defined in string theory (requirement of well-developed theories).
- Should be able to use any proof assistant with the theory library.

Math Proofs
Math Definitions, Results

Lessons Learned

- It is possible to use an interactive proof assistant as an automated VC prover.
- Interactive proof assistants can be used off-the-shelf by importing the mathematical theories of interest.

Questions?