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.”

Organization of verification system

- Assertive Code
- Specs

Programmer
Organization of verification system
Organization of verification system

- VC Generator
- Assertive Code
- Specs
- Verification Conditions
- Programmer
- VC Generator
Organization of verification system
Organization of verification system
Organization of verification system

Proof Checker → Automated Prover → Verification Conditions → VC Generator

Assertive Code → Specs

Programmer
Organization of verification system

Mathematician
- Math Proofs
- Math Definitions, Results

Programmer
- Assertive Code
- Specs
- Verification Conditions

Proof Checker
- Automated Prover
- VC Generator

✓
Organization of verification system

Mathematician:
- Math Proofs
- Math Definitions, Results

Automated Prover
- Proof Checker
- Verification Conditions

Programmer:
- Assertive Code
- Specs
- VC Generator
- Verifications

✓
Organization of verification system

Mathematician
- Math Proofs
- Math Definitions, Results

Automated Prover

Proof Checker

Programmer
- Assertive Code
- Specs
- Verification Conditions
- VC Generator

✓

✓
Organization of verification system

Mathematician
- Math Proofs
- Math Definitions, Results

Programmer
- Assertive Code
- Specs
- Verification Conditions
- VC Generator

Automated Prover
- Proof Checker
- Math Definitions, Results
- Verification Conditions

Labelled HOL
Mathematical Theory
Reuse

| String Theory | Tree Theory | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td>Queue</td>
<td>List</td>
</tr>
<tr>
<td>Trees</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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
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
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
Queue Flip
Specification

contract QueueFlip enhances QueueTemplate

procedure Flip (updates q: Queue)
ensures
  q = reverse (#q)
end QueueFlip
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
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
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
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
    end 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
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
A VC for Recursive Implementation

is_initial (x₂)

is_initial (x₅)

⟨x₃⟩ ∘ q₃ ≠ λ

reverse (q₃) ∘ ⟨x₃⟩ = reverse (⟨x₃⟩ ∘ q₃)
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.
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?