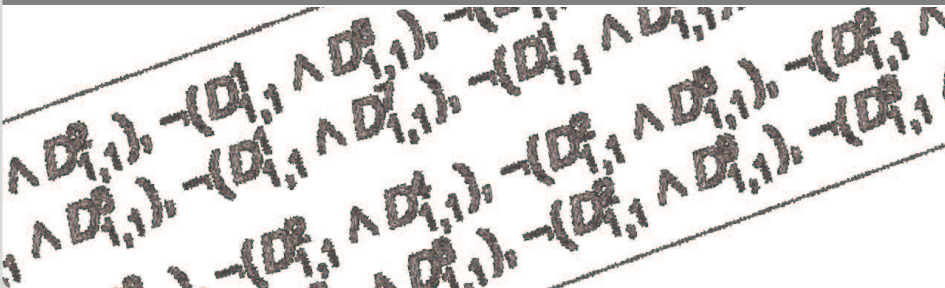


Bridging the interaction gap between logic and code

Sarah Grebing, Mattias Ulbrich

Institute for Theoretical Informatics, KIT



Program verification is an iterative process.

- initial attempts (often) fail
- understand reason for failing is crucial

Interaction is ...

- inspection of proof state
- advancing proof state

..for each iteration

Interaction on:

- specification
- program code
- logic/proof obligation

Switching levels is costly and not well supported

Overall Goal

Interaction concept supporting

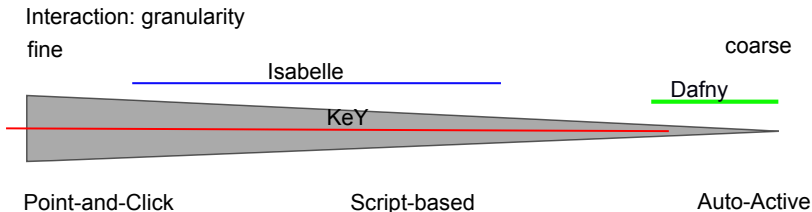
- interactive
- semi-automated
- seamless
- coupled
- fluent

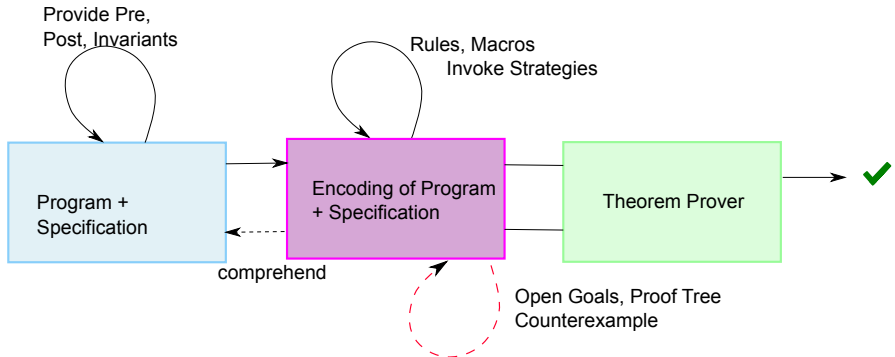
proof guidance for an effective and efficient proof process.

State-of-the-Art Verification Systems

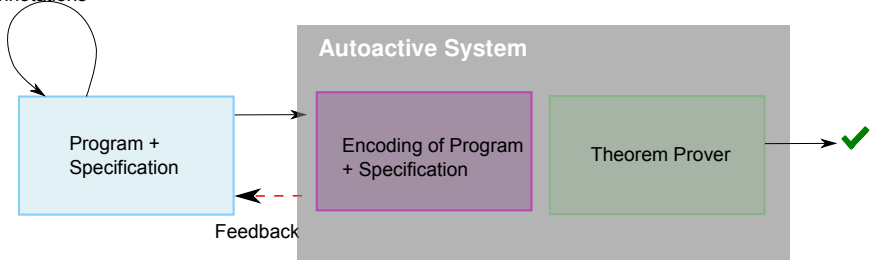
Three interaction principles for guiding the proof search:

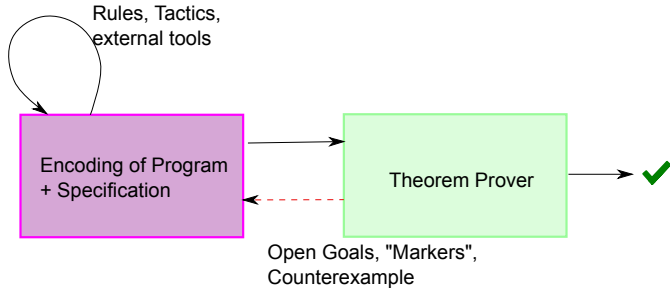
- autoactive
- point-and-click style
- script-based





Pre, Post, Invariant,
Annotations





Pros and Cons

- +/- all necessary information available
- + full proof control
- interaction can be tedious
- error recovery
- two mental models have to be kept in sync

Pros and Cons

- + interaction on input representation → only one mental model
- + comprehensible annotations
 - missing detailed insight into logical level when proof attempt fails
 - leaky abstraction

Pros and Cons

- + problem/proof decomposition in smaller parts
- + interaction steps more coarse grained than point-and-click → more readable/comprehensible and proof plan expressible
- +/- limited insight into logical level
 - two mental models may have to be kept in sync

Interaction concept supporting

- interactive
- semi-automated
- seamless

proof guidance for an effective and efficient proof process.

Support:

- comprehension of failure of proof attempt
- advancing proof

Interaction concept supporting

- interactive
- semi-automated
- seamless

proof guidance for an effective and efficient proof process.

Support:

- comprehension of failure of proof attempt
 - feedback on input language
 - lightweight tools to discharge simple proof problems
 - modularize proof
- advancing proof

Interaction concept supporting

- interactive
- semi-automated
- seamless

proof guidance for an effective and efficient proof process.

Support:

- comprehension of failure of proof attempt
 - feedback on input language
 - lightweight tools to discharge simple proof problems
 - modularize proof
- advancing proof
 - proof exploration
 - fast error recovery

Decrease Costs of Iterations

Goal:

Interaction on suitable abstraction level

Decrease Costs of Iterations

Goal:

Interaction on suitable abstraction level (problem and user dependent)

Goal:

Interaction on suitable abstraction level (problem and user dependent)

Decrease number of level-changes

- stay on input language as long as possible (input and feedback)
- coarse proof steps (proof plan)
- clear overview over overall proof state

Goal:

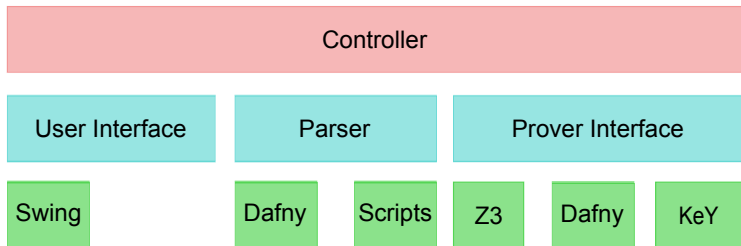
Interaction on suitable abstraction level (problem and user dependent)

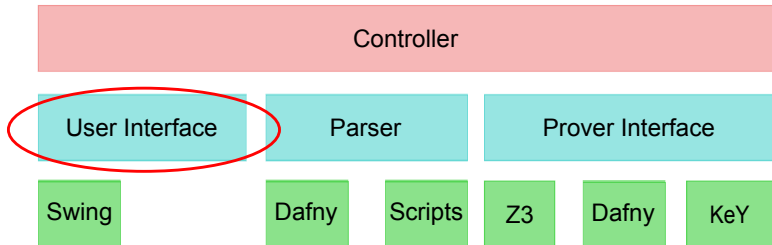
Decrease number of level-changes

- stay on input language as long as possible (input and feedback)
- coarse proof steps (proof plan)
- clear overview over overall proof state

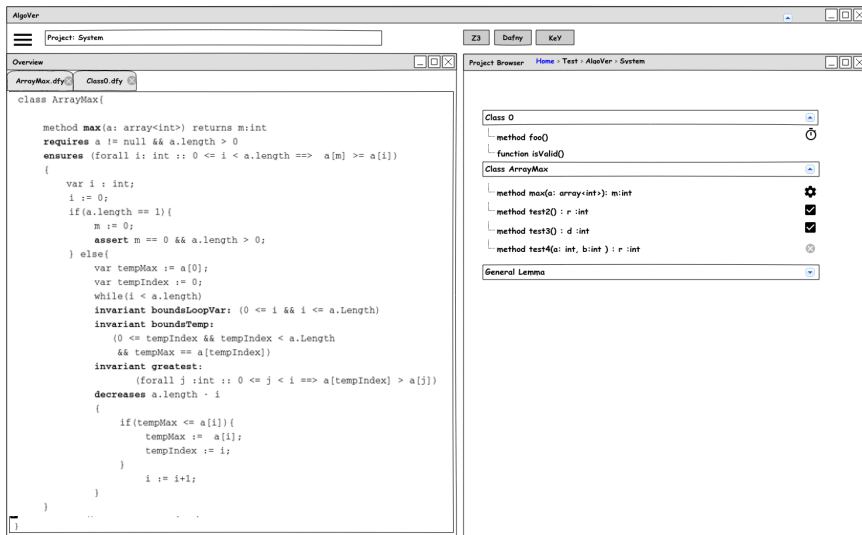
Decrease cost per level-change

- state inspection on logical level
- visible dependencies between levels
- proof exploration on logical level





Example



The screenshot displays the AlgorVer IDE interface. The main editor on the left shows a Dafny program for finding the maximum element in an array. The program includes a class `ArrayMax` with a `max` method. The method takes an array `a` of integers and returns the maximum value `m`. It includes preconditions (`requires`), postconditions (`ensures`), and an invariant (`invariant`) to ensure the correctness of the loop. The loop uses a `while` statement with a `boundsLoopVar` and a `boundsTemp` variable. The loop body updates the maximum value and the index. The program also includes a `test4` method that calls the `max` method on a specific array.

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
  {  
    var i : int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j :int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
  }  
}
```

The right-hand pane shows the Project Browser, which lists the classes and methods in the project. The classes listed are `Class O`, `Class ArrayMax`, and `General Lemma`. The methods listed under `Class ArrayMax` are `method foo()`, `function isValid()`, `method max(a: array<int>): m:int`, `method test2(): r:int`, `method test3(): d:int`, and `method test4(a: int, b:int): r:int`.

Example

AlgoVer

Project: System

Z3 Dafny KeY

Overview

Project Browser Home Test AlgoVer System

ArrayMax.dfy Class0.dfy

```
class ArrayMax{

  method max(a: array<int>) returns m:int
  requires a != null && a.length > 0
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >
  {
    var i : int;
    i := 0;
    if(a.length == 1){
      m := 0;
      assert m == 0 && a.length > 0;
    } else{
      var tempMax := a[0];
      var tempIndex := 0;
      while(i < a.length)
        invariant boundsLoopVar: (0 <= i && i <= a.Length
        invariant boundsTemp:
          (0 <= tempIndex && tempIndex < a.Length
          && tempMax == a[tempIndex])
        invariant greatest:
          (forall j :int :: 0 <= j < i ==> a[temp
        decreases a.length - i
        {
          if(tempMax <= a[i]){
            tempMax := a[i];
            tempIndex := i;
          }
          i := i+1;
        }
      }
    }
  }
}
```

Class 0

method foo()

function isValid()

Class ArrayMax

method max(a: array<int>): m:int

method test2() : r :int

method test3() : d :int

method test4(a: int, b:int) : r :int

General Lemma

Example

AlgoVer

Project: System

Overview

ArrayMax.dfy OtherFile.dfy

```
class ArrayMax{  
  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
  {  
    var i : int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j : int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
    method test2() returns r:int {...}  
  }  
}
```

Z3 Dafny KeY

Project Browser Home Test AlgoVer System

- Class ArrayMax
- method max(a: array<int>): m:int
 - if (a.length == 1)
 - assert (m == 0 & a.length > 0)
 - show ensures
 - else (a.length != 1)
 - assert label
 - invariant initially valid "boundsLoopVar"
 - invariant initially valid "boundsTemp"
 - invariant initially valid "greatest"
 - body preserves invariant "boundsLoopVar"
 - body preserves invariant "boundsTemp"
 - body preserves invariant "greatest"
 - invariant use case
 - loop decreases

Example

AlgoVer

Project: System

Z3 Dafny KeY

Overview

ArrayMax.dfy OtherFile.dfy

```
class ArrayMax{  
  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
    method test2() returns r:int {...}  
  }  
}
```

Class ArrayMax

method max(a: array<int>) : m:int

if(a.length == 1)

assert (m== 0 & a.length > 0)

show ensures

else (a.length != 1)

assert label

invariant initially valid "boundsLoopVar"

invariant initially valid "boundsTemp"

invariant initially valid "greatest"

body preserves invariant "boundsLoopVar"

body preserves invariant "boundsTemp"

body preserves invariant "greatest"

invariant use case

loop decreases

Example

AlgoVer

Project: System

Overview

ArrayMax.dfy OtherFile.dfy

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
  {  
    var i: int;  
    m := 0;  
    assert m == 0 && a.length > 0;  
  } else{  
    var tempMax := a[0]:  
    var tempIndex := 0;  
    while(i < a.length)  
    invariant boundsLoopVar: (0 <= i && i <= a.Length)  
    invariant boundsTemp:  
      (0 <= tempIndex && tempIndex < a.Length  
      && tempMax == a[tempIndex])  
    invariant greatest:  
      (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
    decreases a.length - i  
    {  
      if(tempMax <= a[i]){  
        tempMax := a[i];  
        tempIndex := i;  
      }  
      i := i+1;  
    }  
  }  
  method test2() returns r:int {...}  
}
```

Project Browser

Home · Test · AlgoVer · System

Class ArrayMax

method max(a: array<int>): m:int

if (a.length == 1)

assert (m == 0 && a.length > 0)

show ensures

else (a.length != 1)

assert label

invariant initially valid "boundsLoopVar"

invariant initially valid "boundsTemp"

invariant initially valid "greatest"

body preserves invariant "boundsLoopVar"

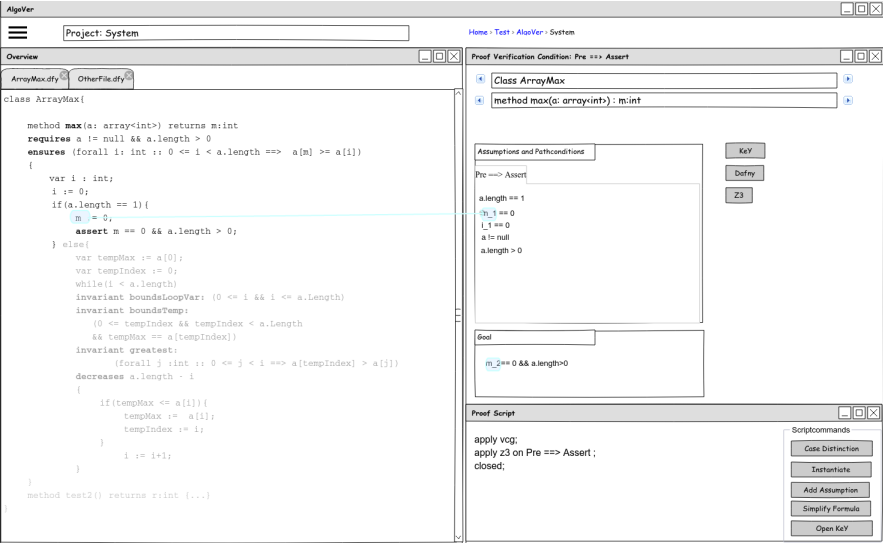
body preserves invariant "boundsTemp"

body preserves invariant "greatest"

invariant use case

loop decreases

Example



The screenshot displays the AlgaVer IDE interface. The main editor shows the source code for the `ArrayMax` class. The code defines a `max` method that takes an array of integers and returns the maximum value. It includes preconditions, postconditions, and an invariant. The verification conditions panel on the right shows the proof verification condition for the `max` method. The assumptions and path conditions are listed, and the goal is to prove the postcondition. The proof script panel shows the commands used to verify the goal.

```
class ArrayMax{  
  
    method max(a: array<int>) returns m:int  
    requires a != null && a.length > 0  
    ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
    {  
        var i : int;  
        i := 0;  
        if(a.length == 1){  
            m = 0;  
            assert m == 0 && a.length > 0;  
        } else{  
            var tempMax := a[0];  
            var tempIndex := 0;  
            while(i < a.length)  
            invariant boundsLoopVar: (0 <= i && i <= a.Length)  
            invariant boundsTemp:  
                (0 <= tempIndex && tempIndex < a.Length  
                 && tempMax == a[tempIndex])  
            invariant greatest:  
                (forall j :int :: 0 <= j < i ==> a[tempIndex] > a[j])  
            decreases a.length - i  
            {  
                if(tempMax <= a[i]){  
                    tempMax := a[i];  
                    tempIndex := i;  
                }  
                i := i+1;  
            }  
        }  
        method test2() returns r:int {...}  
    }  
}
```

Proof Verification Condition: Pre ==> Assert

Class ArrayMax

method max(a: array<int>) : m:int

Assumptions and Pathconditions

Pre ==> Assert

a.length == 1
¬ i_1 == 0
i_1 == 0
a != null
a.length > 0

Goal

m_2 == 0 && a.length > 0

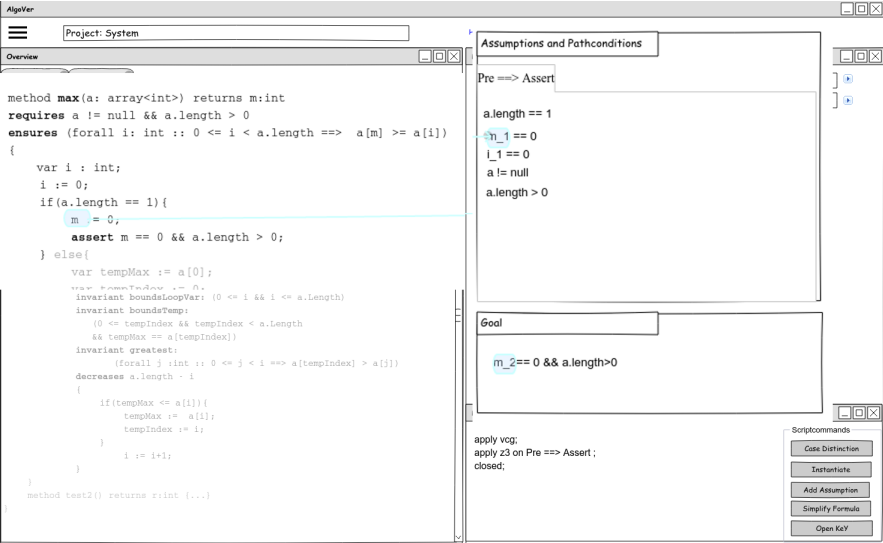
Proof Script

apply vcg;
apply z3 on Pre ==> Assert ;
closed;

Scriptcommands

Case Distinction
Instantiate
Add Assumption
Simplify Formula
Open Key

Example



AligoVer

Project: System

Overview

```
method max(a: array<int>) returns m:int
requires a != null && a.length > 0
ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])
{
    var i : int;
    i := 0;
    if(a.length == 1){
        m := 0;
        assert m == 0 && a.length > 0;
    } else{
        var tempMax := a[0];
        var tempIndex := 0;
        invariant boundsLoopVar: (0 <= i && i <= a.Length)
        invariant boundsTemp:
            (0 <= tempIndex && tempIndex < a.Length
            && tempMax == a[tempIndex])
        invariant greatest:
            (forall j :int :: 0 <= j < i ==> a[tempIndex] > a[j])
        decreases a.length - i
        {
            if(tempMax <= a[i]){
                tempMax := a[i];
                tempIndex := i;
            }
            i := i+1;
        }
    }
    method test2() returns r:int {...}
}
```

Assumptions and Pathconditions

Pre ==> Assert

a.length == 1

i_1 == 0

i_1 == 0

a != null

a.length > 0

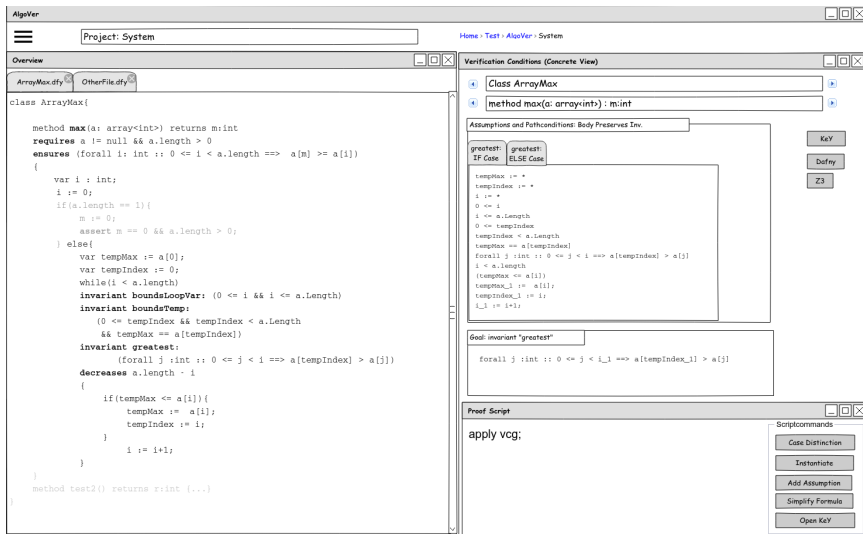
Goal

m_2 == 0 && a.length > 0

Scriptcommands

- Case Distinction
- Instantiate
- Add Assumption
- Simplify Formula
- Open Key

Example



The screenshot displays the AlgoVer IDE interface. The top bar shows the project name "Project: System" and navigation links "Home", "Test", "AlgoVer", and "System". The left sidebar contains "Overview" and "ArrayMax.dfy" (selected). The main editor shows the source code of the `ArrayMax` class, which includes a `max` method with preconditions, postconditions, and an invariant. The right sidebar shows the "Verification Conditions (Concrete View)" for the `max` method. It displays the "Assumptions and Pathconditions: Body Preserves Inv." section, which includes the "greatest" invariant and the "greatest" case. The "Proof Script" section at the bottom shows the command `apply vcg;` and a list of "Scriptcommands" including "Case Distinction", "Instantiate", "Add Assumption", "Simplify Formula", and "Open Key".

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
  }  
  method test2() returns r:int {...}  
}
```

Verification Conditions (Concrete View)

Class ArrayMax

method max(a: array<int>): m:int

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case greatest: ELSE Case

tempMax := *
tempIndex := *
i := *
0 <= i
i <= a.Length
0 <= tempIndex
tempIndex < a.Length
tempMax == a[tempIndex]
forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]
i < a.Length
(tempMax <= a[i])
tempMax_1 := a[i];
tempIndex_1 := i;
i_1 := i+1;

Goal: invariant "greatest"

forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]

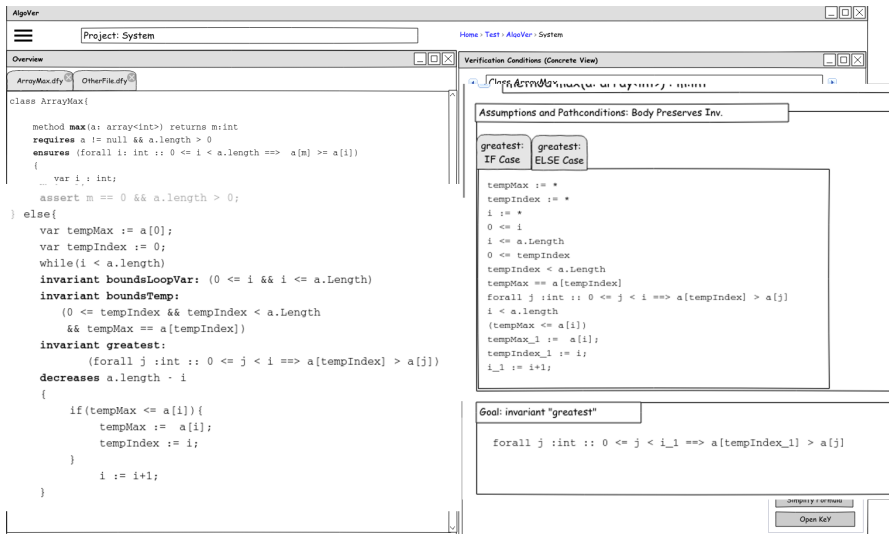
Proof Script

apply vcg;

Scriptcommands

Case Distinction
Instantiate
Add Assumption
Simplify Formula
Open Key

Example



AlgoVer

Project: System

Home > Test > AlgoVer > System

Overview

ArrayMax.dfy OtherFile.dfy

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
    requires a != null && a.length > 0  
    ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
    {  
      ... var i: int;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j :int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
}
```

Verification Conditions (Concrete View)

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case greatest: ELSE Case

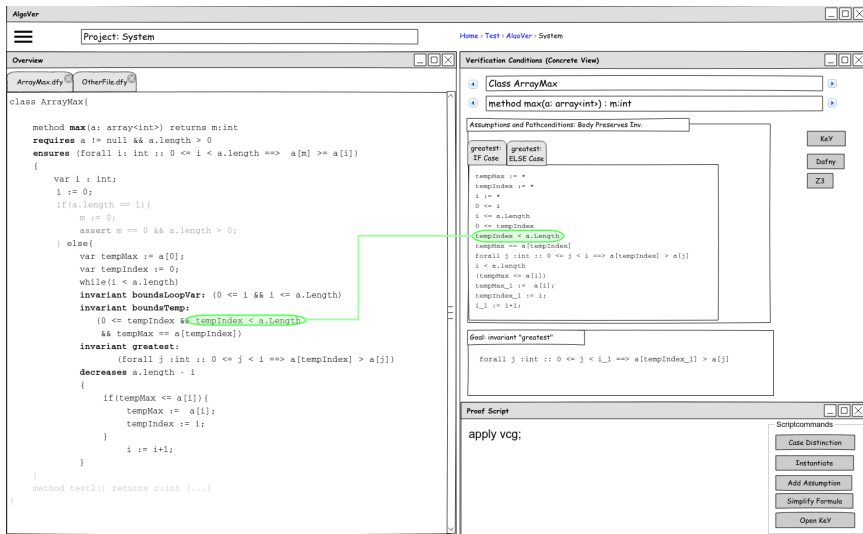
```
tempMax := *  
tempIndex := *  
i := *  
0 <= i  
i <= a.Length  
0 <= tempIndex  
tempIndex < a.Length  
tempMax == a[tempIndex]  
forall j :int :: 0 <= j < i ==> a[tempIndex] > a[j]  
i < a.length  
(tempMax <= a[i])  
tempMax_1 := a[i];  
tempIndex_1 := i;  
i_1 := i+1;
```

Goal: invariant "greatest"

```
forall j :int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]
```

Open Key

Example



The screenshot displays the AlgoVer IDE interface. The main editor shows the source code of a Java class `ArrayMax`. The code includes a `max` method that takes an array of integers and returns the maximum value. It uses a loop to find the maximum element, with an invariant `greatest` that is updated as the loop progresses. The invariant is defined as `greatest := a[i]` where `i` is the current index being compared. The invariant is maintained by updating `tempMax` and `tempIndex` when a new maximum is found.

```
class ArrayMax{  
    method max(a: array<int>) returns m:int  
    requires a != null && a.length > 0  
    ensures (forall i: int :: 0 <= i < a.length ==> a[m] >= a[i])  
    {  
        var i: int;  
        i := 0;  
        if(a.length == 1){  
            m := 0;  
            assert m == 0 && a.length > 0;  
        } else{  
            var tempMax := a[0];  
            var tempIndex := 0;  
            while(i < a.length)  
            invariant boundsLoopVar: (0 <= i && i <= a.Length)  
            invariant boundsTemp:  
                (0 <= tempIndex && tempIndex < a.Length  
                && tempMax == a[tempIndex])  
            invariant greatest:  
                (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
            decreases a.length - i  
            {  
                if(tempMax <= a[i]){  
                    tempMax := a[i];  
                    tempIndex := i;  
                }  
                i := i+1;  
            }  
        }  
        method test2() returns r:int {...}  
    }  
}
```

The right-hand pane shows the **Verification Conditions (Concrete View)** for the `max` method. It displays the assumptions and path conditions for the body, including the invariant `greatest` and the loop condition. The invariant is defined as `greatest := a[i]` and is maintained by the loop body. The loop condition is `tempIndex < a.Length`. The invariant is updated as the loop progresses.

The bottom pane shows the **Proof Script** with the command `apply vcg;` and a list of script commands: `Case Distinction`, `Instantiate`, `Add Assumption`, `Simplify Formula`, and `Open Key`.

Example

AlgoVer

Project: System

Home · Test · AlgoVer · System

Overview

ArrayMax.dfy OtherFiles.dfy

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures {forall i: int :: 0 <= i < a.length ==> a[m] >= a[i]}  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    }  
  } else{  
    var tempMax := a[0];  
    var tempIndex := 0;  
    while(i < a.length)  
    invariant boundsLoopVar: (0 <= i && i <= a.Length)  
    invariant boundsTemp:  
      (0 <= tempIndex && tempIndex < a.Length  
      && tempMax == a[tempIndex])  
    invariant greatest:  
      (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
    decreases a.length - i  
    {  
      if(tempMax <= a[i]){  
        tempMax := a[i];  
        tempIndex := i;  
      }  
      i := i+1;  
    }  
  }  
}
```

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case greatest: ELSE Case

```
tempMax := *  
tempIndex := *  
i := *  
0 <= i  
i <= a.Length  
0 <= tempIndex  
tempIndex < a.Length  
tempMax == a[tempIndex]  
forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]  
i < a.length  
(tempMax <= a[i])  
tempMax_1 := a[i];  
tempIndex_1 := i;  
i_1 := i+1;
```

Goal: invariant "greatest"

```
forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]
```

Simplify Formula
Open Key

Example

AlgoVer

Project: System

Home · Test · AlgoVer · System

Overview

ArrayMax.dfy · OtherFiles.dfy

```
class ArrayMax{  
  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures {forall i: int :: 0 <= i < a.length ==> a[m] >= a[i]}  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          {forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]}  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
  }  
  method test2() returns r:int {...}  
}
```

Verification Conditions (Concrete View)

Class ArrayMax

method max(a: array<int>) : m:int

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case greatest: IF Case* greatest: ELSE Case

tempMax := *
tempIndex := *
i := *
0 <= i
i <= a.Length
0 <= tempIndex
tempIndex < a.Length
tempMax == a[tempIndex]
forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]
i < a.Length
tempMax <= a[i]
tempMax_1 := a[i]
tempIndex_1 := i
i_1 := i+1
tempIndex == a[i]

Goal: invariant "greatest"

forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]

Proof Script

Scriptcommands

apply vcg;
add assumption (tempIndex == a[i]);
apply z3;

Case Distinction
Instantiate
Add Assumption
Simplify Formula
Open Key

Example

AlgoVer

Project: System

Overview

ArrayMax.dfy OtherFile.dfy

```
class ArrayMax{

  method max(a: array<int>) returns m:int
  requires a != null && a.length > 0
  ensures {forall i: int :: 0 <= i < a.length ==> a[m] >= a[i]}
  {
    var i: int;
    i := 0;
    if(a.length == 1){
      m := 0;
      assert m == 0 && a.length > 0;
    } else{
      var tempMax := a[0];
      var tempIndex := 0;
      while(i < a.length)
        invariant boundsLoopVar: (0 <= i && i <= a.Length)
        invariant boundsTemp:
          (0 <= tempIndex && tempIndex < a.Length
           && tempMax == a[tempIndex])
        invariant greatest:
          (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])
        decreases a.length - i
        {
          if(tempMax <= a[i]){
            tempMax := a[i];
            tempIndex := i;
          }
          i := i+1;
        }
      }
    method test2() returns r:int {...}
  }
}
```

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case

greatest: IF Case*

greatest: ELSE Case

```
tempMax := *
tempIndex := *
i := *
0 <= i
i <= a.Length
0 <= tempIndex
tempIndex < a.Length
tempMax == a[tempIndex]
forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]
i < a.length
tempMax <= a[i]
tempMax_1 := a[i]
tempIndex_1 := i
i_1 := i+1
tempIndex == a[i]
```

Goal: invariant "greatest"

```
forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]
```

Proof Script

```
apply vcg;
add assumption (tempIndex == a[i]);
apply z3;
```

Scriptcommands

Case Distinction

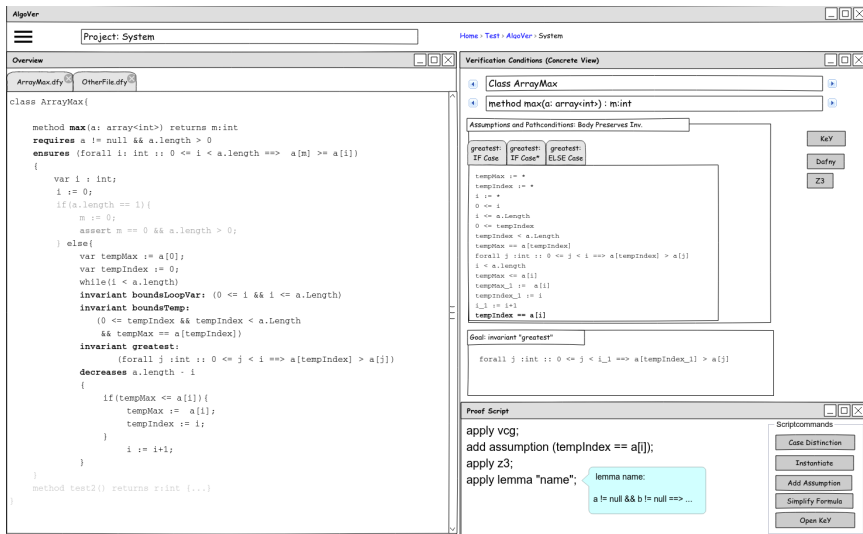
Instantiate

Add Assumption

Simplify Formula

Open KeY

Example



The screenshot shows the AlgoVer IDE interface. The top bar includes the project name "Project: System" and navigation links "Home", "Test", "AlgoVer", and "System". The main window is divided into three panes:

- Overview:** Displays the source code of the `ArrayMax` class. The code defines a `max` method that takes an array of integers and returns the maximum value. It includes preconditions, postconditions, and an invariant.
- Verification Conditions (Concrete View):** Shows the generated verification conditions for the `max` method. It includes the class signature, the method signature, and the body of the method with its assumptions and path conditions. The conditions are listed in a table with columns for "greatest:", "IF Case", and "ELSE Case".
- Proof Script:** Contains the proof script for the method. It starts with `apply vcg;`, followed by `add assumption (tempIndex == a[i]);`, `apply z3;`, and `apply lemma "name";`. A tooltip for the lemma name shows `a != null && b != null ==> ...`.

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures {forall i: int :: 0 <= i < a.length ==> a[m] >= a[i]}  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
  }  
  method test2() returns r:int {...}  
}
```

greatest:	IF Case	ELSE Case
tempMax := *	tempIndex := *	i := *
0 <= i	0 <= i	0 <= i
i <= a.Length	0 <= tempIndex	tempIndex < a.Length
tempMax <= a[tempIndex]	tempMax == a[tempIndex]	forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]
i < a.Length	tempMax <= a[i]	tempMax_1 := a[i]
tempMax <= a[i]	tempIndex_1 := i	i_1 := i+1
tempIndex == a[i]		

Goal: invariant "greatest"

```
forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]
```

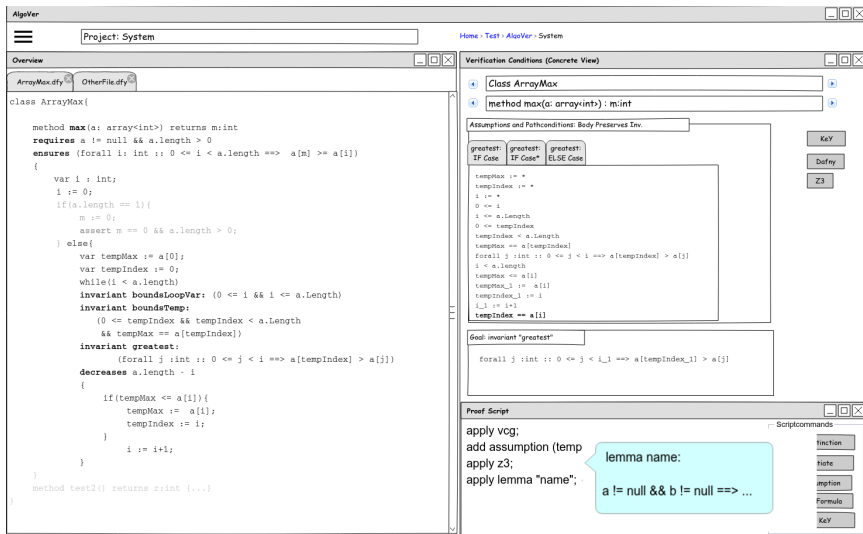
Proof Script

```
apply vcg;  
add assumption (tempIndex == a[i]);  
apply z3;  
apply lemma "name";
```

Scriptcommands

- Case Distinction
- Instantiate
- Add Assumption
- Simplify Formula
- Open Key

Example



The screenshot displays the AlgoVer IDE interface. The top bar shows the project name "Project: System". The left pane, titled "Overview", shows the file structure with "ArrayMax.dfy" selected. The main editor displays the code for the `ArrayMax` class, which includes a `max` method and a `test2` method. The right pane, titled "Verification Conditions (Concrete View)", shows the verification conditions for the `max` method. It includes the class signature, the method signature, and the body of the method. The body is divided into two sections: "Assumptions and Pathconditions: Body Preserves Inv." and "Goal: invariant 'greatest'". The "Assumptions and Pathconditions" section contains the code for the `max` method, including the initialization of `tempMax` and `tempIndex`, the loop, and the update of `tempMax` and `tempIndex`. The "Goal: invariant 'greatest'" section contains the goal statement: `forall j : int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]`. The bottom pane, titled "Proof Script", shows the script commands for the proof, including `apply vcg;`, `add assumption (temp`, `apply z3;`, and `apply lemma "name";`. A callout box highlights the lemma name: `a != null && b != null ==> ...`.

```
class ArrayMax{  
  method max(a: array<int>) returns m:int  
  requires a != null && a.length > 0  
  ensures {forall i: int :: 0 <= i < a.length ==> a[m] >= a[i]}  
  {  
    var i: int;  
    i := 0;  
    if(a.length == 1){  
      m := 0;  
      assert m == 0 && a.length > 0;  
    } else{  
      var tempMax := a[0];  
      var tempIndex := 0;  
      while(i < a.length)  
        invariant boundsLoopVar: (0 <= i && i <= a.Length)  
        invariant boundsTemp:  
          (0 <= tempIndex && tempIndex < a.Length  
           && tempMax == a[tempIndex])  
        invariant greatest:  
          (forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j])  
        decreases a.length - i  
        {  
          if(tempMax <= a[i]){  
            tempMax := a[i];  
            tempIndex := i;  
          }  
          i := i+1;  
        }  
      }  
    }  
  }  
  method test2() returns r:int {...}  
}
```

Verification Conditions (Concrete View)

Class ArrayMax

method max(a: array<int>) : m:int

Assumptions and Pathconditions: Body Preserves Inv.

greatest: IF Case greatest: IF Case* greatest: ELSE Case

```
tempMax := *  
tempIndex := *  
i := *  
0 <= i  
i <= a.Length  
0 <= tempIndex  
tempIndex < a.Length  
tempMax == a[tempIndex]  
forall j: int :: 0 <= j < i ==> a[tempIndex] > a[j]  
i < a.length  
tempMax <= a[i]  
tempMax_1 := a[i]  
tempIndex_1 := i  
i_1 := i+1  
tempIndex == a[i]
```

Goal: invariant "greatest"

```
forall j: int :: 0 <= j < i_1 ==> a[tempIndex_1] > a[j]
```

Proof Script

Scriptcommands

```
apply vcg;  
add assumption (temp  
apply z3;  
apply lemma "name";
```

lemma name:
a != null && b != null ==> ...

Bridging interaction gap between code and logic by

- fluent transition between levels
- different coupled proof views
- interaction on all levels
- seamless integration of sophisticated methods

Future Work

- implement concept
- evaluate concept with users

In your experience:

What is the bottleneck for interaction?