

Teaching with Logika

Conceiving and Constructing Correct Software

Stefan Hallerstede¹ (sha@ece.au.dk)

John Hatcliff² (hatcliff@ksu.edu)

Robby² (robby@ksu.edu)

¹Aarhus University

²Kansas State University

FMTea, Milan, Italy, 10 September 2024

Context, Approach and Evolution

Use and Significance of Slang and Logika

Feedback

Discussion

Next Steps

Context, Approach and Evolution

Use and Significance of Slang and Logika

Feedback

Discussion

Next Steps

Software Engineering Curriculum

- Development of *new BSc/MSc curriculum* at Aarhus University (Engineering, Fall 2018)
- *Problem solving, modelling, reasoning, and verification* are woven into “common” courses

Software Engineering Curriculum

- Development of *new BSc/MSc curriculum* at Aarhus University (Engineering, Fall 2018)
- *Problem solving, modelling, reasoning, and verification* are woven into “common” courses
 - Introduction to Programming (BSc 10 ECTS – informal)
 - Software Architecture (BSc 5 ECTS – informal)
 - Discrete Mathematics (BSc 5 ECTS – informal)
 - *Programming and Modelling* (BSc 10 ECTS – formal)
 - Declarative Programming (BSc&MSc 10 ECTS – informal/formal)
 - **Software Correctness** (MSc 5 ECTS – formal)

Software Engineering Curriculum

- Development of *new BSc/MSc curriculum* at Aarhus University (Engineering, Fall 2018)
- *Problem solving, modelling, reasoning, and verification* are woven into “common” courses
 - Introduction to Programming (BSc 10 ECTS – informal)
 - Software Architecture (BSc 5 ECTS – informal)
 - Discrete Mathematics (BSc 5 ECTS – informal)
 - *Programming and Modelling* (BSc 10 ECTS – formal)
 - Declarative Programming (BSc&MSc 10 ECTS – informal/formal)
 - **Software Correctness** (MSc 5 ECTS – formal)
- Local students are prepared for formal methods thinking
- They see **Slang** and **Logika** in *Programming and Modelling* and **Software Correctness**
 - **Slang**: Scala dialect with verification support
 - **Logika**: Interactive support for programming and verifying with Slang

Software Engineering Curriculum

- Development of *new BSc/MSc curriculum* at Aarhus University (Engineering, Fall 2018)
- *Problem solving, modelling, reasoning, and verification* are woven into “common” courses
 - Introduction to Programming (BSc 10 ECTS – informal)
 - Software Architecture (BSc 5 ECTS – informal)
 - Discrete Mathematics (BSc 5 ECTS – informal)
 - *Programming and Modelling* (BSc 10 ECTS – formal)
 - Declarative Programming (BSc&MSc 10 ECTS – informal/formal)
 - **Software Correctness** (MSc 5 ECTS – formal)
- Local students are prepared for formal methods thinking
- They see **Slang** and **Logika** in *Programming and Modelling* and **Software Correctness**
 - **Slang**: Scala dialect with verification support
 - **Logika**: Interactive support for programming and verifying with Slang
- Cohort on MSc level is mixed – **background at MSc level varies**
 - Slang and Logika are well-suited for this situation

Evolution of the Course

- In 2012 started predecessor course (as **programming course**)
 - Using Java, Scheme and Prolog
 - **Some reasoning** about functional and logical programs
 - Used **informal inductive proofs**

Evolution of the Course

- In 2012 started predecessor course (as **programming course**)
 - Using Java, Scheme and Prolog
 - **Some reasoning** about functional and logical programs
 - Used **informal inductive proofs**
- In 2016 tried to include some **Coq**
- In 2017 tried to include some **Isabelle**
 - Both of these were **difficult to get across to the students**

Evolution of the Course

- In 2012 started predecessor course (as **programming course**)
 - Using Java, Scheme and Prolog
 - **Some reasoning** about functional and logical programs
 - Used **informal inductive proofs**
- In 2016 tried to include some **Coq**
- In 2017 tried to include some **Isabelle**
 - Both of these were **difficult to get across to the students**
- In 2018 introduced **Slang/Logika**
 - This worked well, staying focused on **programming** methodology
 - Started design of new course based on Slang/Logika (*dropping Prolog*)

Evolution of the Course

- In 2012 started predecessor course (as **programming course**)
 - Using Java, Scheme and Prolog
 - **Some reasoning** about functional and logical programs
 - Used **informal inductive proofs**
- In 2016 tried to include some **Coq**
- In 2017 tried to include some **Isabelle**
 - Both of these were **difficult to get across to the students**
- In 2018 introduced **Slang/Logika**
 - This worked well, staying focused on **programming** methodology
 - Started design of new course based on Slang/Logika (*dropping Prolog*)
- In 2023 the **new course** Software Correctness was established

Content and Objectives

- Schedule:

- **Week 1: Introduction** – Reasoning about software (and **tool** installation)
- **Week 2: Tracing Facts** – Pick up the students **reasoning in familiar ways**
- **Week 3: Conditionals** – **Progress slowly** discussing different approaches
- **Week 4: Contracts (Test)** – Ensure **students see benefit** for their programming skills
- **Week 5: Contracts (Proof)** – Based on preceding week but using compositional **proof**
- **Week 6: Loops and Recursion** – Some theory: programs are just another kind of formula
- **Week 7: Unfolding and Fixpoints** – More theory with large and complex formulas
- **Week 8: Loops and Recursion Testing** – Ensure **students see benefit**
- **Week 9: Sequences and Arrays** – Increase complexity of programs
- **Finally: Verification Examples and Practice** – Provide **methodology** backed by examples

Content and Objectives

- Schedule:

- **Week 1: Introduction** – Reasoning about software (and **tool** installation)
- **Week 2: Tracing Facts** – Pick up the students **reasoning in familiar ways**
- **Week 3: Conditionals** – **Progress slowly** discussing different approaches
- **Week 4: Contracts (Test)** – Ensure **students see benefit** for their programming skills
- **Week 5: Contracts (Proof)** – Based on preceding week but using compositional **proof**
- **Week 6: Loops and Recursion** – Some theory: programs are just another kind of formula
- **Week 7: Unfolding and Fixpoints** – More theory with large and complex formulas
- **Week 8: Loops and Recursion Testing** – Ensure **students see benefit**
- **Week 9: Sequences and Arrays** – Increase complexity of programs
- **Finally: Verification Examples and Practice** – Provide **methodology** backed by examples
- Accompanied by a **programming project** where some test and proof is applied (mostly at home)
- Exercises **only in class** (teacher helps)

Content and Objectives

- Schedule:

- **Week 1: Introduction** – Reasoning about software (and **tool** installation)
- **Week 2: Tracing Facts** – Pick up the students **reasoning in familiar ways**
- **Week 3: Conditionals** – **Progress slowly** discussing different approaches
- **Week 4: Contracts (Test)** – Ensure **students see benefit** for their programming skills
- **Week 5: Contracts (Proof)** – Based on preceding week but using compositional **proof**
- **Week 6: Loops and Recursion** – Some theory: programs are just another kind of formula
- **Week 7: Unfolding and Fixpoints** – More theory with large and complex formulas
- **Week 8: Loops and Recursion Testing** – Ensure **students see benefit**
- **Week 9: Sequences and Arrays** – Increase complexity of programs
- **Finally: Verification Examples and Practice** – Provide **methodology** backed by examples
- Accompanied by a **programming project** where some test and proof is applied (mostly at home)
- Exercises **only in class** (teacher helps)

- Objectives:

- Improve **programming** skills, **testing** skills, **documentation** skills, **reasoning** skills
- Do not limit students' vision to Slang,
so the material becomes relevant **beyond the course**

Context, Approach and Evolution

Use and Significance of Slang and Logika

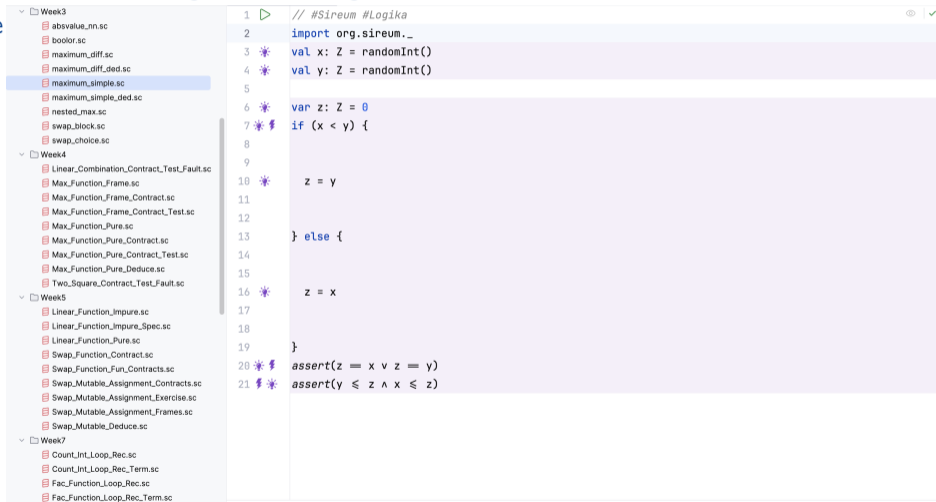
Feedback

Discussion

Next Steps

A Quick Tour of Slang and Logika

The User Interface



```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9
10     z = y
11
12 } else {
13
14     z = x
15
16 }
17
18
19
20 assert(z == x v z == y)
21 assert(y <= z ^ x <= z)
```


A Quick Tour of Slang and Logika

The User Interface

The screenshot shows the Slang IDE interface. On the left is a file explorer with a tree view containing folders for Week3, Week4, Week5, and Week7, each with several .sc files. The main area is a code editor showing a Slang program. The code is as follows:

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9
10     z = y
11
12
13 } else {
14
15
16     z = x
17
18
19 }
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

A callout box with a purple border and background contains the text: "Slang is a dialect of the Scala programming language".

A Quick Tour of Slang and Logika

The User Interface

The screenshot shows the Slang IDE interface. On the left is a file explorer with a tree view containing folders for Week3, Week4, Week5, and Week7, each with several .sc files. The main area is a code editor showing a Slang program. The code includes comments, imports, variable declarations, conditional logic, and assertions. Two callout boxes are overlaid on the code:

- A purple box containing the text: "Slang is a dialect of the Scala programming language".
- A purple box containing the text: "Functional and imperative programming".

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9
10     z = y
11
12
13 } else {
14
15
16     z = x
17
18
19 }
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

A Quick Tour of Slang and Logika

The User Interface

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9
10     z = y
11
12
13 } else {
14
15
16     z = x
17
18
19 }
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

Slang is a dialect of the
Scala programming language

Functional and imperative programming

Dedicated basic data types with
well-defined semantics

A Quick Tour of Slang and Logika

The User Interface

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9
10     z = y
11
12
13 } else {
14
15
16     z = x
17
18
19 }
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

Slang is a dialect of the
Scala programming language

Functional and imperative programming

Dedicated basic data types with
well-defined semantics

Support for algebraic types, records and arrays

A Quick Tour of Slang and Logika

The User Interface

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7
8
9
10 z = y
11
12
13 } else {
14
15
16 z = x
17
18
19 }
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

Click to show scribed incantations

File Explorer:

- Week3
 - absvalue_mn.sc
 - boolean.sc
 - maximum_diff.sc
 - maximum_diff_ded.sc
 - maximum_simple.sc
 - maximum_simple_ded.sc
 - nested_max.sc
 - swap_block.sc
 - swap_choice.sc
- Week4
 - Linear_Combination_Contract_Test_Fault.sc
 - Max_Function_Frame.sc
 - Max_Function_Frame_Contract.sc
 - Max_Function_Frame_Contract_Test.sc
 - Max_Function_Pure.sc
 - Max_Function_Pure_Contract.sc
 - Max_Function_Pure_Contract_Test.sc
 - Max_Function_Pure_Deduce.sc
 - Two_Square_Contract_Test_Fault.sc
- Week5
 - Linear_Function_Impure.sc
 - Linear_Function_Impure_Spec.sc
 - Linear_Function_Pure.sc
 - Swap_Function_Contract.sc
 - Swap_Function_Fun_Contracts.sc
 - Swap_Mutable_Assignment_Contracts.sc
 - Swap_Mutable_Assignment_Exercise.sc
 - Swap_Mutable_Assignment_Frames.sc
 - Swap_Mutable_Deduce.sc
- Week7
 - Count_Int_Loop_Rec.sc
 - Count_Int_Loop_Rec_Term.sc
 - Fac_Function_Loop_Rec.sc
 - Fac_Function_Loop_Rec_Term.sc

A Quick Tour of Slang and Logika

The User Interface

The screenshot displays the Logika IDE interface. On the left is a file explorer showing a project structure with folders for Week3, Week4, Week5, and Week7, containing various .sc files. The main editor shows Slang code for a program named #Sireum #Logika. The code includes imports, variable declarations for random integers, an if-then-else block, and assertions. The console on the right shows the output of the program, including satisfiability checks and a list of claims. A purple callout box with a white border and rounded corners is overlaid on the code editor, containing the text: "Logika uses SMT solvers in the background".

```
// #Sireum #Logika
import org.sireum._
val x: Z = randomInt()
val y: Z = randomInt()

var z: Z = 0
if (x < y) {
  z = y
} else {
  z = x
}
assert(z == x v z == y)
assert(y ≤ z ∧ x ≤ z)
```

Satisfiability check for if-then at [7, 7]: Sat
Satisfiability check for if-else at [7, 7]: Sat

; Satisfiability check for if-then at [7, 7]
; Result: Sat

; Claims:
; x == At[Z](".random", 0),
; y == At[Z](".random", 1),
; z == 0,
; x < y
; (set-logic ALL)

(define-sort B () Bool)
(define-fun |B.unary_!| ((x B)) B (not x))
(define-fun |B.unary_~| ((x B)) B (not x))
(define-fun |B.==| ((x B) (y B)) B (= x y))
(define-fun |B.!=| ((x B) (y B)) B (not (= x y))
(define-fun |B.&| ((x B) (y B)) B (and x y))
(define-fun |B.| | ((x B) (y B)) B (or x y))
(define-fun |B.^| ((x B) (y B)) B (xor x y))
(define-fun |B._>:| ((x B) (y B)) B (=> x y))
(define-fun |B.->:| ((x B) (y B)) B (=> x y))

A Quick Tour of Slang and Logika

The User Interface

The screenshot displays the Logika IDE interface. On the left is a file explorer showing a project structure with folders for Week3, Week4, Week5, and Week7, each containing several .sc files. The main editor shows Slang code for a program named #Sireum #Logika. The code defines variables x and y as random integers, then checks if x < y. If true, it sets z = y; otherwise, it sets z = x. Finally, it asserts that z = x ∨ z = y and that y ≤ z ∧ x ≤ z. The right-hand side of the IDE shows the output of the SMT solver, including satisfiability checks for if-then and if-else blocks, and a list of claims including x < y and (set-logic ALL). A list of logical operators is also shown, such as unary negation, binary conjunction, disjunction, and implication.

```
// #Sireum #Logika
import org.sireum._
val x: Z = randomInt()
val y: Z = randomInt()

var z: Z = 0
if (x < y) {
  z = y
} else {
  z = x
}
assert(z == x ∨ z == y)
assert(y ≤ z ∧ x ≤ z)
```

Satisfiability check for if-then at [7, 7]: Sat
Satisfiability check for if-else at [7, 7]: Sat

; Satisfiability check for if-then at [7, 7]
; Result: Sat

; Claims:
; x < y
; (set-logic ALL)

(define-sort B () Bool)
(define-fun |B.unary_!| ((x B)) B (not x))
(define-fun |B.unary_~| ((x B)) B (not x))
(define-fun |B.==| ((x B) (y B)) B (= x y))
(define-fun |B.!=| ((x B) (y B)) B (not (= x y)))
(define-fun |B.&| ((x B) (y B)) B (and x y))
(define-fun |B.| | ((x B) (y B)) B (or x y))
(define-fun |B.^| ((x B) (y B)) B (xor x y))
(define-fun |B.==>| ((x B) (y B)) B (=> x y))
(define-fun |B.->:| ((x B) (y B)) B (=> x y))

Logika uses **SMT solvers** in the background

It also simplifies formulas by rewriting

A Quick Tour of Slang and Logika

The User Interface

The screenshot displays the Logika IDE interface. On the left is a file explorer showing a project structure with folders for Week3, Week4, Week5, and Week7, each containing several .sc files. The main editor shows a Slang script:

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9     z = y
10
11 } else {
12
13     z = x
14
15 }
16
17
18
19
20 assert(z == x v z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

On the right, the console output shows the results of Satisfiability checks:

```
Satisfiability check for if-then at [7, 7]: Sat
Satisfiability check for if-else at [7, 7]: Sat

; Satisfiability check for if-then at [7, 7]
; Result: Sat
; Claims:
; x < y
```

Three callout boxes highlight key features:

- Logika uses **SMT solvers** in the background
- It also simplifies formulas by rewriting
- All of this can be inspected **interactively**

At the bottom right, a list of built-in operators is visible:

```
(define-fun |B.unary_!| ((x B)) B (not x))
(define-fun |B.unary_~| ((x B)) B (not x))
(define-fun |B.==| ((x B) (y B)) B (= x y))
(define-fun |B.!=| ((x B) (y B)) B (not (= x y))
(define-fun |B.&| ((x B) (y B)) B (and x y))
(define-fun |B.| ((x B) (y B)) B (or x y))
(define-fun |B.^| ((x B) (y B)) B (xor x y))
(define-fun |B._>:| ((x B) (y B)) B (=> x y))
(define-fun |B.->:| ((x B) (y B)) B (=> x y))
```


A Quick Tour of Slang and Logika

The User Interface

The screenshot displays the Logika IDE interface. On the left is a file explorer showing a project structure with folders for Week3, Week4, Week5, and Week7, containing various .sc files. The main editor shows a code snippet in Logika:

```
1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt()
4 val y: Z = randomInt()
5
6 var z: Z = 0
7 if (x < y) {
8
9     z = y
10
11 } else {
12
13     z = x
14
15 }
16
17
18
19
20 assert(z == x ∨ z == y)
21 assert(y ≤ z ∧ x ≤ z)
```

On the right, the SMT solver output is visible, showing satisfiability checks for if-then and if-else blocks, and a list of claims including $x < y$. Below the code editor, four purple callout boxes provide key insights:

- Logika uses **SMT solvers** in the background
- It also simplifies formulas by rewriting
- All of this can be inspected **interactively**
- There's **no magic**

A Quick Tour of Slang and Logika

The User Interface

The screenshot displays the Slang IDE interface. On the left is a file explorer showing a project structure with folders for Week3, Week4, Week5, and Week7, containing various .sc files. The central code editor shows a Slang script with the following content:

```

1 // #Sireum #Logika
2 import org.sireum._
3 val x: Z = randomInt(0, 10)
4 val y: Z = randomInt(0, 10)
5
6 var z: Z = 0
7
8 if (x < y) {
9   Deduce(⊢ (x < y))
10  Deduce(⊢ (x ≤ y))
11  z = y
12  Deduce(⊢ (x ≤ z))
13  Deduce(⊢ (z = y))
14 } else {
15   Deduce(⊢ ¬(x < y))
16   Deduce(⊢ (y ≤ x))
17   z = x
18   Deduce(⊢ (y ≤ z))
19   Deduce(⊢ (z = x))
20 }
21 assert(z = x ∨ z = y)
22 assert(y ≤ z ∧ x ≤ z)

```

On the right, the console shows the output of a satisfiability check:

```

Satisfiability check for if-then at [7, 7]: Sat
Satisfiability check for if-else at [7, 7]: Sat
; Result: Sat
; Solver: /Users/au443183/Applications/Sireum/bir
; Arguments: -smt2 -in rlimit=2000000 -t:500
; Time: 0.052s
; Claims:
;
; x == At[Z](".random", 0),
; y == At[Z](".random", 1),
; z == 0,
; x < y
;
(set-logic ALL)

(define-sort B () Bool)
(define-fun |B.unary_!| ((x B)) B (not x))
(define-fun |B.unary_~| ((x B)) B (not x))
(define-fun |B.==| ((x B) (y B)) B (= x y))
(define-fun |B.!=| ((x B) (y B)) B (not (= x y))
(define-fun |B.&| ((x B) (y B)) B (and x y))
(define-fun |B.| ((x B) (y B)) B (or x y))
(define-fun |B.^| ((x B) (y B)) B (xor x y))
(define-fun |B._>:| ((x B) (y B)) B (=> x y))
(define-fun |B.->:| ((x B) (y B)) B (=> x y))

```

A purple callout box with a white border and shadow is overlaid on the code editor, containing the text: "Proof commands available in Slang syntax (not in comments)".

A Quick Tour of Slang and Logika

Contracts & Proof

```
1  // #Sireum #Logika
2  import org.sireum._
3
4  def swap(a: ZS, i:Z, j: Z) : Unit = {
5    Contract(
6      Requires(0 ≤ i, i < a.size, 0 ≤ j, j < a.size),
7      Modifies(a),
8      Ensures(
9        a(i) = In(a)(j),
10       a(j) = In(a)(i),
11       ∀(a.indices)(k ⇒ k = i ∨ k = j ∨ a(k) = In(a)(k)),
12       a.size = In(a).size
13     )
14   )
15   val t: Z = a(i)
16   Deduce(⊢ (t = In(a)(i)))
17   a(i) = a(j)
18   Deduce(⊢ (t = In(a)(i)))
19   Deduce(⊢ (a(i) = In(a)(j)))
20   a(j) = t
21   Deduce(⊢ (t = In(a)(i)))
22   Deduce(⊢ (a(j) = t))
23   Deduce(⊢ (a(j) = In(a)(i)))
24   Deduce(⊢ (a(i) = In(a)(j)))
25 }
26
```

Contracts for compositional reasoning

A Quick Tour of Slang and Logika

Contracts & Proof

```
1 // #Sireum #Logika
2 import org.sireum._
3
4 def swap(a: ZS, i:Z, j: Z) : Unit = {
5   Contract(
6     Requires(0 ≤ i, i < a.size, 0 ≤ j, j < a.size),
7     Modifies(a),
8     Ensures(
9       a(i) = In(a)(j),
10      a(j) = In(a)(i),
11      ∀(a.indices)(k ⇒ k = i ∨ k = j ∨ a(k) = In(a)(k)),
12      a.size = In(a).size
13    )
14  )
15  val t: Z = a(i)
16  Deduce(⊢ (t = In(a)(i)))
17  a(i) = a(j)
18  Deduce(⊢ (t = In(a)(i)))
19  Deduce(⊢ (a(i) = In(a)(j)))
20  a(j) = t
21  Deduce(⊢ (t = In(a)(i)))
22  Deduce(⊢ (a(j) = t))
23  Deduce(⊢ (a(j) = In(a)(i)))
24  Deduce(⊢ (a(i) = In(a)(j)))
25 }
26
```

Contracts for **compositional reasoning**

Hoare-style reasoning about imperative commands

A Quick Tour of Slang and Logika

Contracts & Proof

```
1 // #Sireum #Logika
2 import org.sireum._
3
4 def swap(a: ZS, i:Z, j: Z) : Unit = {
5   Contract(
6     Requires(0 ≤ i, i < a.size, 0 ≤ j, j < a.size),
7     Modifies(a),
8     Ensures(
9       a(i) == In(a)(j),
10      a(j) == In(a)(i),
11      ∀(a.indices)(k ⇒ k == i ∨ k == j ∨ a(k) == In(a)(k)),
12      a.size == In(a).size
13    )
14  )
15  val t: Z = a(i)
16  Deduce(⊢ (t == In(a)(i)))
17  a(i) = a(j)
18  Deduce(⊢ (t == In(a)(i)))
19  Deduce(⊢ (a(i) == In(a)(j)))
20  a(j) = t
21  Deduce(⊢ (t == In(a)(i)))
22  Deduce(⊢ (a(j) == t))
23  Deduce(⊢ (a(j) == In(a)(i)))
24  Deduce(⊢ (a(i) == In(a)(j)))
25  }
26
```

Contracts for **compositional reasoning**

Hoare-style reasoning about imperative commands

Proof in Slang as close as possible to **programming**

A Quick Tour of Slang and Logika

Contracts & Proof

```
1 // #Sireum #Logika
2 import org.sireum._
3
4 def swap(a: ZS, i:Z, j: Z) : Unit = {
5   Contract(
6     Requires(0 ≤ i, i < a.size, 0 ≤ j, j < a.size),
7     Modifies(a),
8     Ensures(
9       a(i) = In(a)(j),
10      a(j) = In(a)(i),
11      ∀(a.indices)(k ⇒ k = i ∨ k = j ∨ a(k) = In(a)(k)),
12      a.size = In(a).size
13    )
14  )
15  val t: Z = a(i)
16  Deduce(⊢ (t = In(a)(i)))
17  a(i) = a(j)
18  Deduce(⊢ (t = In(a)(i)))
19  Deduce(⊢ (a(i) = In(a)(j)))
20  a(j) = t
21  Deduce(⊢ (t = In(a)(i)))
22  Deduce(⊢ (a(j) = t))
23  Deduce(⊢ (a(j) = In(a)(i)))
24  Deduce(⊢ (a(i) = In(a)(j)))
25 }
26
```

Familiar Interface for students

k: Z

Example

A Quick Tour of Slang and Logika

Contracts & Proof

```
1 // #Sireum #Logika
2 import org.sireum._
3
4 def swap(a: ZS, i:Z, j: Z) : Unit = {
5   Contract(
6     Requires(0 ≤ i, i < a.size, 0 ≤ j, j < a.size),
7     Modifies(a),
8     Ensures(
9       a(i) = In(a)(j),
10      a(j) = In(a)(i),
11      ∀(a.indices)(k ⇒ k = i ∨ k = j ∨ a(k) = In(a)(k)),
12      a.size = In(a).size
13    )
14  )
15  val t: Z = a(i)
16  Deduce(⊢ (t = In(a)(i)))
17  a(i) = a(j)
18  Deduce(⊢ (t = In(a)(i)))
19  Deduce(⊢ (a(i) = In(a)(j)))
20  a(j) = t
21  Deduce(⊢ (t = In(a)(i)))
22  Deduce(⊢ (a(j) = t))
23  Deduce(⊢ (a(j) = In(a)(i)))
24  Deduce(⊢ (a(i) = In(a)(j)))
25 }
26
```

Familiar Interface for students

Interactive inspection of all elements, including formulas and proof

k: Z

Example

A Quick Tour of Slang and Logika

Proof Information

```
5  @pure def sorted(seq: ISZ[Z]): B = {
6    Contract(
7      Ensures(Res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i)))
8    )
9    ✨ var res: B = true
10   ✨ var k: Z = 1
11   ✨ while (k < seq.size) {
12     Invariant(
13       Modifies(k, res),
14       ✨ k ≥ 1,
15       ✨ k-1 ≥ 0,
16       ✨ k-1 ≤ seq.size,
17       ✨ seq.size ≥ 2 || res = true,
18       ✨ seq.size < 2 ∨ k ≤ seq.size,
19       ✨ seq.size < 2 || res = All(1 until k)(i => seq(i-1) ≤ seq(i))
20     )
21     ✨ ✨ Deduce(⊢ (seq.size ≥ 2))
22     ✨ ✨ if (seq(k - 1) > seq(k)) {
23       ✨ res = false
24     }
25     ✨ k = k + 1
26   }
27   ✨ ✨ Deduce(⊢ (seq.size ≥ 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
28   ✨ ✨ Deduce(⊢ (seq.size < 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
29   ✨ return res
30 }
31
```

Proof information available **interactively**

A Quick Tour of Slang and Logika

Proof Information

```
5  @pure def sorted(seq: ISZ[Z]): B = {
6  Contract(
7  Ensures(Res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i)))
8  )
9  var res: B = true
10 var k: Z = 1
11 while (k < seq.size) {
12   Invariant(
13     Modifies(k, res),
14     k ≥ 1,
15     k-1 ≥ 0,
16     k-1 ≤ seq.size,
17     seq.size ≥ 2 || res = true,
18     seq.size < 2 ∨ k ≤ seq.size,
19     seq.size < 2 || res = All(1 until k)(i => seq(i-1) ≤ seq(i))
20   )
21   Deduce(⊢ (seq.size ≥ 2))
22   if (seq(k - 1) > seq(k)) {
23     Click to show some hints lse
24   }
25   k = k + 1
26   }
27   Deduce(⊢ (seq.size ≥ 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
28   Deduce(⊢ (seq.size < 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
29   return res
30 }
31
```

Proof information available **interactively**

“Click the *light bulb*”

A Quick Tour of Slang and Logika

Proof Information

```

5  @pure def sorted(seq: ISZ[Z]): B = {
6      Contract(
7  ✎  Ensures(Res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i)))
8      )
9  ✎  var res: B = true
10 ✎  var k: Z = 1
11 ✎  while (k < seq.size) {
12      Invariant(
13          Modifies(k, res),
14          ✎  k ≥ 1,
15          ✎  k-1 ≥ 0,
16          ✎  k-1 ≤ seq.size,
17          ✎  seq.size ≥ 2 || res = true,
18          ✎  seq.size < 2 v k ≤ seq.size,
19          ✎  seq.size < 2 || res = All(1 until k)(i => seq(i-1) ≤ seq(i))
20      )
21 ✎  Deduce(⊢ (seq.size ≥ 2))
22 ✎  if (seq(k - 1) > seq(k)) {
23 ✎      res = false
24  }
25 ✎  k = k + 1
26  }
27 ✎  Deduce(⊢ (seq.size ≥ 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
28 ✎  Deduce(⊢ (seq.size < 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
29 ✎  return res
30  }
31

```

```

{ // State claims at line 23
  At(res, 0) = T;
  At(k, 0) = 1;
  k < seq.size;
  k ≥ 1;
  k - 1 ≥ 0;
  k - 1 ≤ seq.size;
  seq.size ≥ 2;
  seq.size < 2 v
    k ≤ seq.size;
  ~(seq.size < 2);
  res = V(1 until k)(i => seq(i - 1) ≤ seq(i));
  seq(k - 1) > seq(k)
}

```

Filter claims ...

A Quick Tour of Slang and Logika

Proof Information

```
5  @pure def sorted(seq: ISZ[Z]): B = {
6      Contract(
7  ✎  Ensures(Res = All(1 until seq.size)(i ⇒ seq(i-1) ≤ seq(i)))
8      )
9  ✎  var res: B = true
10 ✎  var k: Z = 1
11 ✎  while (k < seq.size) {
12      Invariant(
13          Modifies(k, res),
14          ✎  k ≥ 1,
15          ✎  k-1 ≥ 0,
16          ✎  k-1 ≤ seq.size,
17          ✎  seq.size ≥ 2 || res = true,
18          ✎  seq.size < 2 v k ≤ seq.size,
19          ✎  seq.size < 2 || res = All(1 until k)(i ⇒ seq(i-1) ≤ seq(i))
20      )
21 ✎  Deducer(⊢ (seq.size ≥ 2))
22 ✎  if (seq(k-1) > seq(k)) {
23 ✎      res = false
24  }
25 ✎  k = k + 1
26  }
27 ✎  Deducer(⊢ (seq.size ≥ 2 || res = All(1 until seq.size)(i ⇒ seq(i-1) ≤ seq(i))))
28 ✎  Deducer(⊢ (seq.size < 2 || res = All(1 until seq.size)(i ⇒ seq(i-1) ≤ seq(i))))
29 ✎  return res
30  }
31
```

```
{ // State claims at line 23
  At(res, 0) = T;
  At(k, 0) = 1;
  k < seq.size;
  k ≥ 1;
  k - 1 ≥ 0;
  k - 1 ≤ seq.size;
  seq.size ≥ 2;
  seq.size < 2 v
    k ≤ seq.size;
  ~(seq.size < 2);
  res = V(1 until k)(i ⇒ seq(i-1) ≤ seq(i));
  seq(k-1) > seq(k)
}
```

Proof information shown to student close to the program text

A Quick Tour of Slang and Logika

Proof Information

```

5  @pure def sorted(seq: ISZ[Z]): B = {
6    Contract(
7  # Ensures(Res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i)))
8    )
9  # var res: B = true
10 # var k: Z = 1
11 # while (k < seq.size) {
12   Invariant(
13     Modifies(k, res),
14     # k ≥ 1,
15     # k-1 ≥ 0,
16     # k-1 ≤ seq.size,
17     # seq.size ≥ 2 || res = true,
18     # seq.size < 2 ∨ k ≤ seq.size,
19     # seq.size < 2 || res = All(1 until k)(i => seq(i-1) ≤ seq(i))
20   )
21 # Deducer(⊢ (seq.size ≥ 2))
22 # if (seq(k - 1) > seq(k)) {
23 #   res = false
24 # }
25 # k = k + 1
26 # }
27 # Deducer(⊢ (seq.size ≥ 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
28 # Deducer(⊢ (seq.size < 2 || res = All(1 until seq.size)(i => seq(i-1) ≤ seq(i))))
29 # return res
30 # }
31

```

```

{ // State claims at line 23
  At(res, 0) = T;
  At(k, 0) = 1;
  k < seq.size;
  k ≥ 1;
  k - 1 ≥ 0;
  k - 1 ≤ seq.size;
  seq.size ≥ 2;
  seq.size < 2 ∨
    k ≤ seq.size;
  ¬(seq.size < 2);
  res = V(1 until k)(i => seq(i - 1) ≤ seq(i));
  seq(k - 1) > seq(k)
}

```

Proof information shown to student
close to the program text

Easy to match program text to formulas
(also large formulas)

A Quick Tour of Slang and Logika

Informal vs Formal

```
1 | // #Sireum #Logika
2 | import org.sireum._
3 |
4 | val m: Z = randomInt();
5 | val n: Z = randomInt()
6 | val z: Z = m + n
7 | // deduce z == m + n (consequence of assignment)
8 | val y: Z = z - n
9 | // deduce z == m + n (old fact)
10 | // deduce y == z - n (consequence of assignment)
11 | // deduce y == m (proof by algebra)
12 | // (y == z - n
13 | // == (m + n) - n
14 | // == m)
15 | val x: Z = z - y
16 | // deduce z == m + n (old fact)
17 | // deduce y == m (old fact)
18 | // deduce x == z - y (consequence of assignment)
19 | // deduce x == n (proof by algebra)
20 | // (x == z - y
21 | // == (m + n) - m
22 | // == n)
23 | assert(x == n & y == m)
24 |
```

Informal proofs in comments useable
without tool support

A Quick Tour of Slang and Logika

Informal vs Formal

```
1 // #Sireum #Logika
2 import org.sireum._
3
4 val m: Z = randomInt();
5 val n: Z = randomInt()
6 val z: Z = m + n
7 // deduce z == m + n (consequence of assignment)
8 val y: Z = z - n
9 // deduce z == m + n (old fact)
10 // deduce y == z - n (consequence of assignment)
11 // deduce y == m (proof by algebra)
12 // (y == z - n
13 // == (m + n) - n
14 // == m)
15 val x: Z = z - y
16 // deduce z == m + n (old fact)
17 // deduce y == m (old fact)
18 // deduce x == z - y (consequence of assignment)
19 // deduce x == n (proof by algebra)
20 // (x == z - y
21 // == (m + n) - m
22 // == n)
23 assert(x == n & y == m)
24
```

Informal proofs in comments useable
without tool support

Also used on white board

A Quick Tour of Slang and Logika

Informal vs Formal

```
1  ▸ // #Sireum #Logika
2  import org.sireum._
3
4  ✨ val m: Z = randomInt();
5  ✨ val n: Z = randomInt()
6  ✨ val z: Z = m + n
7  // deduce z = m + n (consequence of assignment)
8  ✨ val y: Z = z - n
9  // deduce z = m + n (old fact)
10 // deduce y = z - n (consequence of assignment)
11 // deduce y = m (proof by algebra)
12 //           (y = z - n
13 //            = (m + n) - n
14 //            = m)
15 ✨ val x: Z = z - y
16 // deduce z = m + n (old fact)
17 // deduce y = m (old fact)
18 // deduce x = z - y (consequence of assignment)
19 // deduce x = n (proof by algebra)
20 //           (x = z - y
21 //            = (m + n) - m
22 //            = n)
23 ✨ ✨ assert(x = n ∧ y = m)
24
```

Logika can do many proofs fully automatic

✔ Logika Verified
Proof is accepted

A Quick Tour of Slang and Logika

Informal vs Formal

```
1  ▸ // #Sireum #Logika
2  import org.sireum._
3
4  ✨ val m: Z = randomInt();
5  ✨ val n: Z = randomInt()
6  ✨ val z: Z = m + n
7  // deduce z = m + n (consequence of assignment)
8  ✨ val y: Z = z - n
9  // deduce z = m + n (old fact)
10 // deduce y = z - n (consequence of assignment)
11 // deduce y = m (proof by algebra)
12 //           (y = z - n
13 //            = (m + n) - n
14 //            = m)
15 ✨ val x: Z = z - y
16 // deduce z = m + n (old fact)
17 // deduce y = m (old fact)
18 // deduce x = z - y (consequence of assignment)
19 // deduce x = n (proof by algebra)
20 //           (x = z - y
21 //            = (m + n) - m
22 //            = n)
23 ✨ ✨ assert(x = n ∧ y = m)
24
```

Logika can do many proofs fully automatic

Beginning students benefit from this

✔ Logika Verified
Proof is accepted

A Quick Tour of Slang and Logika

Informal vs Formal

```
1  ▸ // #Sireum #Logika
2  import org.sireum._
3
4  ✨ val m: Z = randomInt();
5  ✨ val n: Z = randomInt()
6  ✨ val z: Z = m + n
7  // deduce z = m + n (consequence of assignment)
8  ✨ val y: Z = z - n
9  // deduce z = m + n (old fact)
10 // deduce y = z - n (consequence of assignment)
11 // deduce y = m (proof by algebra)
12 //           (y = z - n
13 //            = (m + n) - n
14 //            = m)
15 ✨ val x: Z = z - y
16 // deduce z = m + n (old fact)
17 // deduce y = m (old fact)
18 // deduce x = z - y (consequence of assignment)
19 // deduce x = n (proof by algebra)
20 //           (x = z - y
21 //            = (m + n) - m
22 //            = n)
23 ✨ ✨ assert(x = n ∧ y = m)
24
```

Logika can do many proofs fully automatic

Beginning students benefit from this

If they know that it can be proved,
Logika confirms or refutes their deductions

✔ Logika Verified
Proof is accepted

A Quick Tour of Slang and Logika

Informal vs Formal

```
1  // #Sireum #Logika
2  import org.sireum._
3
4  val m: Z = randomInt();
5  val n: Z = randomInt()
6  val z: Z = m + n
7  // deduce z = m + n (consequence of assignment)
8  val y: Z = z - n
9  // deduce z = m + n (old fact)
10 // deduce y = z - n (consequence of assignment)
11 // deduce y = m (proof by algebra)
12 //           (y = z - n
13 //            = (m + n) - n
14 //            = m)
15 val x: Z = z - y
16 // deduce z = m + n (old fact)
17 // deduce y = m (old fact)
18 // deduce x = z - y (consequence of assignment)
19 // deduce x = n (proof by algebra)
20 //           (x = z - y
21 //            = (m + n) - m
22 //            = n)
23 assert(x = n ^ y = m)
24
```

Logika can do many proofs fully automatic

Beginning students benefit from this

If they know that it can be proved,
Logika confirms or refutes their deductions

The students can use Logika like a teacher

✓ Logika Verified
Proof is accepted

Context, Approach and Evolution

Use and Significance of Slang and Logika

Feedback

Discussion

Next Steps

Student Feedback

It was nice with a little mini-project to use some of the techniques learned in the course

Student Feedback

It was nice with a little mini-project to use some of the techniques learned in the course

It was really nice to have exercises during the lecture and that [the teacher] walked around to help us if we were struggling with some of the proofs. I really liked that!

Student Feedback

It was nice with a little mini-project to use some of the techniques learned in the course

It was really nice to have exercises during the lecture and that [the teacher] walked around to help us if we were struggling with some of the proofs. I really liked that!

[The teacher] was really good at explaining the subjects and always made sure that the class was understanding the theory

Student Feedback

It was nice with a little mini-project to use some of the techniques learned in the course

It was really nice to have exercises during the lecture and that [the teacher] walked around to help us if we were struggling with some of the proofs. I really liked that!

[The teacher] was really good at explaining the subjects and always made sure that the class was understanding the theory

I am not sure if I am going to use what I have learned

Context, Approach and Evolution

Use and Significance of Slang and Logika

Feedback

Discussion

Next Steps

Discussion

- A good **user-friendly tool** that the students are familiar with is **essential**
- Students look for the **benefit** they get out of a course
- They **don't** have a strong background in maths and logics
- It's better if taught material **does not look like formal methods**
- Concerning proof, **in-class attention** by teacher is required
- Using theorem provers directly **did not work well**
- Notation and methodology should be as close to **programming** as possible
- The students rate this course **very high**: 4.4 out of 5
(but response rate needs to be improved)
- **Despite its title** “Software Correctness” high number of inscriptions
(20 students)
- Lecture materials for the course are **publicly available**
(<https://github.com/santoslab/software-correctness-course-materials>)

Much More Material Available From Kansas State University

A lot of material available online

S 301: Logical Foundations of Programming

Search...

- Propositional Logic Proofs
- Predicate Logic Translations
- Predicate Logic Proofs
- Mathematical Induction
- Intro to Programming Logic
- Functions and Loops
 - 9.1. Functions
 - 9.2. Recursion
 - 9.3. Loops
 - 9.4. Logika Facts
 - 9.5. Summary
- Sequences, Globals, and Termination
- U Auto
- Using Hugo and Hugo Relearn Theme.

CC BY-NC-SA
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Functions and Loops > Loops

```

val x: Z = Z.read()
val y: Z = Z.read()

var sum: Z = 0
var count: Z = 0

//prove the invariant before the loop begins
Deduce(
  1 ( sum == 0 ) by Premise, //from the "sum = 0" assignment
  2 ( count == 0 ) by Premise, //from the "count = 0" assignment
  3 ( sum == count*x ) by Algebra*(1, 2) //proved EXACTLY the loop invariant
)

while (count != y) {
  Invariant(
    Modifies(sum, count),
    sum == count * x
  )

  Deduce(
    1 ( sum == count*x ) by Premise, //the loop invariant holds
    //at the beginning of an iteration
  )

  sum = sum + x

  Deduce(
    1 ( sum == Old(sum) + x ) by Premise, //from "sum = sum + x" assignment
    2 ( Old(sum) == count*x ) by Premise, //loop invariant WAS true, but sum
    3 ( sum == count*x + x ) by Algebra*(1,2) //current knowledge without using
  )

  count = count + 1

  Deduce(
    1 ( count == Old(count)+ 1 ) by Premise, //from "count = count + 1" assignm
    2 ( sum == Old(count)*x + x ) by Premise, //from previous "sum = count*x +
    //but count has changed
    3 ( sum == (count-1)*x + x ) by Algebra*(1,2),
    4 ( sum == count*x - x + x ) by Algebra*(3),
    5 ( sum == count*x ) by Algebra*(4) //loop invariant holds at end of
  )
}

```

Much More Material Available From Kansas State University

S 301: Logical Foundations of Programming

Search...

- Propositional Logic Proofs
- Predicate Logic Translations
- Predicate Logic Proofs
- Mathematical Induction
- Intro to Programming Logic
- Functions and Loops
 - 9.1. Functions
 - 9.2. Recursion
 - 9.3. Loops
 - 9.4. Logika Facts
 - 9.5. Summary
- Sequences, Globals, and Termination
- JUnit Auto
- Using Hugo and Hugo Relearn Theme.
- CC BY-NC-SA
- Work is licensed under a Creative

```
Functions and Loops > Loops

val x: Z = Z.read()
val y: Z = Z.read()

var sum: Z = 0
var count: Z = 0

//prove the invariant before the loop begins
Deduce(
  1 ( sum == 0 ) by Premise,
  2 ( count == 0 ) by Premise,
  3 ( sum == count*x ) by Algebra*(1, 2)
)

while (count != y) {
  Invariant(
    Modifies(sum, count),
    sum == count * x
  )

  Deduce(
    1 ( sum == count*x ) by Premise, //the loop invariant holds
    //at the beginning of an iteration
  )

  sum = sum + x

  Deduce(
    1 ( sum == Old(sum) + x ) by Premise, //from "sum = sum + x" assignment
    2 ( Old(sum) == count*x ) by Premise, //loop invariant WAS true, but sum
    3 ( sum == count*x + x ) by Algebra*(1,2) //current knowledge without using
  )

  count = count + 1

  Deduce(
    1 ( count == Old(count)+ 1 ) by Premise, //from "count = count + 1" assignm
    2 ( sum == Old(count)*x + x ) by Premise, //from previous "sum = count*x +
    //but count has changed
    3 ( sum == (count-1)*x + x ) by Algebra*(1,2),
    4 ( sum == count*x - x + x ) by Algebra*(3),
    5 ( sum == count*x ) by Algebra*(4) //loop invariant holds at end of
  )
}
```

A lot of material available online

E.g.

<https://textbooks.cs.ksu.edu/cis301/>

Much More Material Available From Kansas State University

S 301: Logical Foundations of Programming

Search...

- Propositional Logic Proofs
- Predicate Logic Translations
- Predicate Logic Proofs
- Mathematical Induction
- Intro to Programming Logic
- Functions and Loops
 - 9.1. Functions
 - 9.2. Recursion
 - 9.3. Loops
 - 9.4. Logika Facts
 - 9.5. Summary
- Sequences, Globals, and Termination
- U Auto
- Using Hugo and Hugo Relearn Theme.
- CC BY-NC-SA
- Work is licensed under a Creative

```
Functions and Loops > Loops

val x: Z = Z.read()
val y: Z = Z.read()

var sum: Z = 0
var count: Z = 0

//prove the invariant before the loop begins
Deduce(
  1 ( sum == 0 ) by Premise,
  2 ( count == 0 ) by Premise,
  3 ( sum == count*x ) by Algebra*(1, 2)
)

while (count != y) {
  Invariant(
    Modifies(sum, count),
    sum == count * x
  )

  Deduce(
    1 ( sum == count*x ) by Premise, //the loop invariant holds
    //at the beginning of an iteration
  )

  sum = sum + x

  Deduce(
    1 ( sum == Old(sum) + x ) by Premise, //from "sum = sum + x" assignment
    2 ( Old(sum) == count*x ) by Premise, //loop invariant WAS true, but sum
    3 ( sum == count*x + x ) by Algebra*(1,2) //current knowledge without using
  )

  count = count + 1

  Deduce(
    1 ( count == Old(count)+ 1 ) by Premise, //from "count = count + 1" assignm
    2 ( sum == Old(count)*x + x ) by Premise, //from previous "sum = count*x +
    //but count has changed
    3 ( sum == (count-1)*x + x ) by Algebra*(1,2),
    4 ( sum == count*x - x + x ) by Algebra*(3),
    5 ( sum == count*x ) by Algebra*(4) //loop invariant holds at end of
  )
}
```

A lot of material available online

E.g.

<https://textbooks.cs.ksu.edu/cis301/>

Authors: Robby, John Hatcliff, Julie Thorton

Much More Material Available From Kansas State University

S 301: Logical Foundations of Programming

Search...

- Propositional Logic Proofs
- Predicate Logic Translations
- Predicate Logic Proofs
- Mathematical Induction
- Intro to Programming Logic
- Functions and Loops
 - 9.1. Functions
 - 9.2. Recursion
 - 9.3. Loops
 - 9.4. Logika Facts
 - 9.5. Summary
- Sequences, Globals, and Termination
- U Auto
- Using Hugo and Hugo Relearn Theme.
- CC BY-NC-SA
- Work is licensed under a Creative

```
Functions and Loops > Loops

val x: Z = Z.read()
val y: Z = Z.read()

var sum: Z = 0
var count: Z = 0

//prove the invariant before the loop begins
Deduce(
  1 ( sum == 0 ) by Premise,
  2 ( count == 0 ) by Premise,
  3 ( sum == count*x ) by Algebra*(1, 2)
)

while (count != y) {
  Invariant(
    Modifies(sum, count),
    sum == count * x
  )

  Deduce(
    1 ( sum == count*x ) by Premise,
  )

  sum = sum + x

  Deduce(
    1 ( sum == Old(sum) + x ) by Premise, //from "sum = sum + x" assignment
    2 ( Old(sum) == count*x ) by Premise, //loop invariant WAS true, but sum
    3 ( sum == count*x + x ) by Algebra*(1,2) //current knowledge without using
  )

  count = count + 1

  Deduce(
    1 ( count == Old(count)+ 1 ) by Premise, //from "count = count + 1" assignm
    2 ( sum == Old(count)*x + x ) by Premise, //from previous "sum = count*x +
    //but count has changed
    3 ( sum == (count-1)*x + x ) by Algebra*(1,2),
    4 ( sum == count*x - x + x ) by Algebra*(3),
    5 ( sum == count*x ) by Algebra*(4) //loop invariant holds at end of
  )
}
```

A lot of material **available online**

E.g.

<https://textbooks.cs.ksu.edu/cis301/>

Authors: Robby, John Hatcliff, Julie Thorton

Visit **John's presentation on Logika** later today!

Context, Approach and Evolution

Use and Significance of Slang and Logika

Feedback

Discussion

Next Steps

Next Steps

- Extend the number of **examples**
- Improve support for **self-study**
- improve presentation of **more advanced verification**
- Improve presentation of **proof methodology**
- Rely on discussion and **feedback from students** for improvements
- The course **evolves gradually** – material, tool and students change