Reasoning with Objects

Joseph E. Hollingsworth, IUS
Murali Sitaraman, Clemson

This research is funded in part by NSF grants CCF-1161916 and DUE-1022941.
Some reasoning is implicit

- We reason about programming integers as though they are mathematical integers (subject to bounds, of course)
- We associate mathematical operators (e.g., +) with operations we can do on integers in programs (e.g., +)
- This mathematics and reasoning can be made explicit
Mathematical modeling

- **Type** Integer is modeled by $\mathbb{Z}$;
- **Constraints for all** $i$: Integer,  
  $\text{min}_\text{int} \leq i \leq \text{max}_\text{int}$;
Alternatively...

- **Type** Integer is modeled by $\mathbb{Z}$;
- Let $i$ be an example Integer;
- **Constraints**
  $$\text{min\_int} \leq i \leq \text{max\_int};$$
Initial value specification...

- **Type** Integer is modeled by \( Z \);
  - exemplar \( i \);
- **Constraints**
  - \( \text{min}_{\text{int}} \leq i \leq \text{max}_{\text{int}} \);
- **initialization ensures** \( i = 0 \);
Specification of operations ...

- **Type** Integer is modeled by $\mathbb{Z}$;
  - ...

- specification of operations, e.g., $i++$
  - **Operation** Increment
    - (updates $i$: Integer);
    - **requires** $i < \text{max\_int}$;
    - **ensures** $i = \#i + 1$;
More examples ...

- What is a suitable way to model the state of a light bulb?
  - ...

- ...
More examples ...

- **Type** Light_Bulb_State
  - is modeled by \( B \);
  - exemplar \( b \);
  - Initialization ensures \( b = \text{false} \);

- Exercises: specification of operations Turn_On, Turn_Off, and Is_On
More examples ...

- How would you model the state of a traffic light?
- ...

- Alternative models and discussion
Detailed Example: Grid Positioning

- Model the position of an object on a grid with X and Y coordinates
  - Initially, the position is (0, 0)
  - The grid is bounded
  - Operations allow the position to be moved, left, right, etc.

- Activities involve writing code to position the object at various places and reasoning about the code
“Hands On” Logical Reasoning with Objects

- Google Clemson RESOLVE
- Bookmark: https://www.cs.clemson.edu/resolve/
- Go to Tab: Web IDE
- Click on Link: Reason with Components
Grid Positioning Activities and Reasoning

- Click on Item Components to bring up Finder
- Select Programs in the Finder
- Select the first activity on reasoning with objects and perform the activities; Click on MP_Prove to reason
- To do the above, you need to know Grid Positioning model and operations; see the next slide
Grid Positioning Modeling and Operations

- Click on Item Components to bring up Finder
- Select (reusable) Concepts in the Finder
- Select Grid_Positioning_Template
- Click on Grid_Positioning_Template that appears to the right again
- A realization of this template (code) can be found at Realizations that appears in the above window [Not needed for activities.]
Creating New Activities

- Sign in with an e-mail using the “sign in” link near top right.
- Once you’re signed in, click on Item Components to bring up Finder.
- Right Click on Programs in the Finder to create a new program.
- Right Click on Concepts in the Finder to create a new (reusable) concept.
Data abstraction examples ...

- How would you mathematically model the contents of a stack?
- Is a set model appropriate?
- Why or why not?

- What about a queue?
Data abstraction examples ...

- How would you mathematically model the contents of a stack?
  - Is a set model appropriate?
  - Why or why not?

- What about a queue?
Mathematical Modeling Summary

- To write formal specifications, we need to model the state mathematically.
- Some objects we use in programming, such as Integers and Reals, have implicit models.
- For others, such as stacks, queues, lists, etc., we need to conceive explicit mathematical models.
More “Hands On” Logical Reasoning with Objects

- Google Clemson RESOLVE

- Bookmark: [https://www.cs.clemson.edu/resolve/](https://www.cs.clemson.edu/resolve/)

- Go to Tab: Web IDE

- Click on Link: Reason with Components
Stack and Queue Activities

□ Click on Item Components to bring up Finder
□ Select Programs in the Finder
□ Select relevant activities
□ To do the above, you need to know specifications of Stacks and Queues; see the next slide
Stack Modeling and Operations

- Click on Item Components to bring up Finder
- Select Concepts in the Finder
- Select Integer_Stack_Template (or Stack_Template)
- Click on Integer_Stack_Template (or Stack_Template) that appears to the right again
Example

Specification:

**Operation** Do_Nothing (**restores** S: Stack);

**Goal:** Same as **ensures** S = #S;

**Code:**

**Procedure** Do_Nothing (**restores** S: Stack);

**Var** E: Entry;

Pop(E, S);

Push(E, S);

**end** Do_Nothing;
Mathematical Modeling of Stacks

**Concept** Stack_Template(*type* Entry; Max_Depth: Integer);

```pascal
uses String_Theory;
```

**Type Family** Stack *is modeled by* ...

**Operation** Push...
**Operation** Pop...

...
Mathematical Modeling of Stacks

**Concept** Stack_Template(*type* Entry; Max_Depth: Integer);

**uses** String_Theory;

**Type Family** Stack is modeled by

Str(Entry);

exemplar S;

constraints |S| <= Max_Depth;

initialization ensures S = empty_string;

...
Alternative Specifications of Push

**Operation** Push *(restores E: Entry; updates S: Stack)*;

requires \(|S| < \text{Max\_Depth}\);

ensures \(S = <E> \circ \#S\);

**Operation** Push *(alters E: Entry; updates S: Stack)*;

requires \(|S| < \text{Max\_Depth}\);

ensures \(S = <\#E> \circ \#S\);
Specification of Stack Operations

**Operation** Push *(alters E: Entry; updates S: Stack)*;
  *requires* $|S| < \text{Max\_Depth}$;
  *ensures* $S = \langle \#E \rangle \circ \#S$;

**Operation** Pop *(replaces R: Entry; updates S: Stack)*;
  *requires* $|S| > 0$;
  *ensures* $\#S = \langle R \rangle \circ S$;

**Operation** Depth *(restores S: Stack): Integer*;
  *ensures* $\text{Depth} = |S|$;

...
Alternative Specifications of Pop

**Operation Pop** *(replaces R: Entry; updates S: Stack);*

*requires* $|S| > 0$;

*ensures* $#S = <R> \circ S$;

---

**Operation Pop** *(replaces R: Entry; updates S: Stack);*

*requires* $|S| > 0$;

*ensures* $R = \text{First}(#S)$ and $S = \text{Rest}(#S)$;
Example

Specification:

**Operation** Do_Nothing (**restores** S: Stack);

Goal: Same as **ensures** S = #S;

Code:

**Procedure** Do_Nothing (**restores** S: Stack);

**Var** E: Entry;

Pop(E, S);

Push(E, S);

**end** Do_Nothing;
Establishing ensures clause of Do_Nothing

**Given**: $S = \#S$ and $|S| \leq \text{Max\_Depth}$;

Pop(E, S);
Push(E, S);

**Goal**: $S = \#S$;
Discussion

- Is the code Correct? If not, fix it
- Important Idea: The reasoning can be done mechanically
- Principles of reasoning about all objects and operations are the same
  - Need mathematical specifications
- VC generation and automated verification demo
More Mathematical Reasoning

- Create the example using the web interface, generate VCs, and prove them
- For recursive implementations
  - Recursive calls are handled just as any other call
  - Need to show termination using a programmer-supplied decreasing “metric”
- For iterative implementations, invariants and decreasing metrics are used