Direct Reasoning

Greg Kulczycki

Contributions

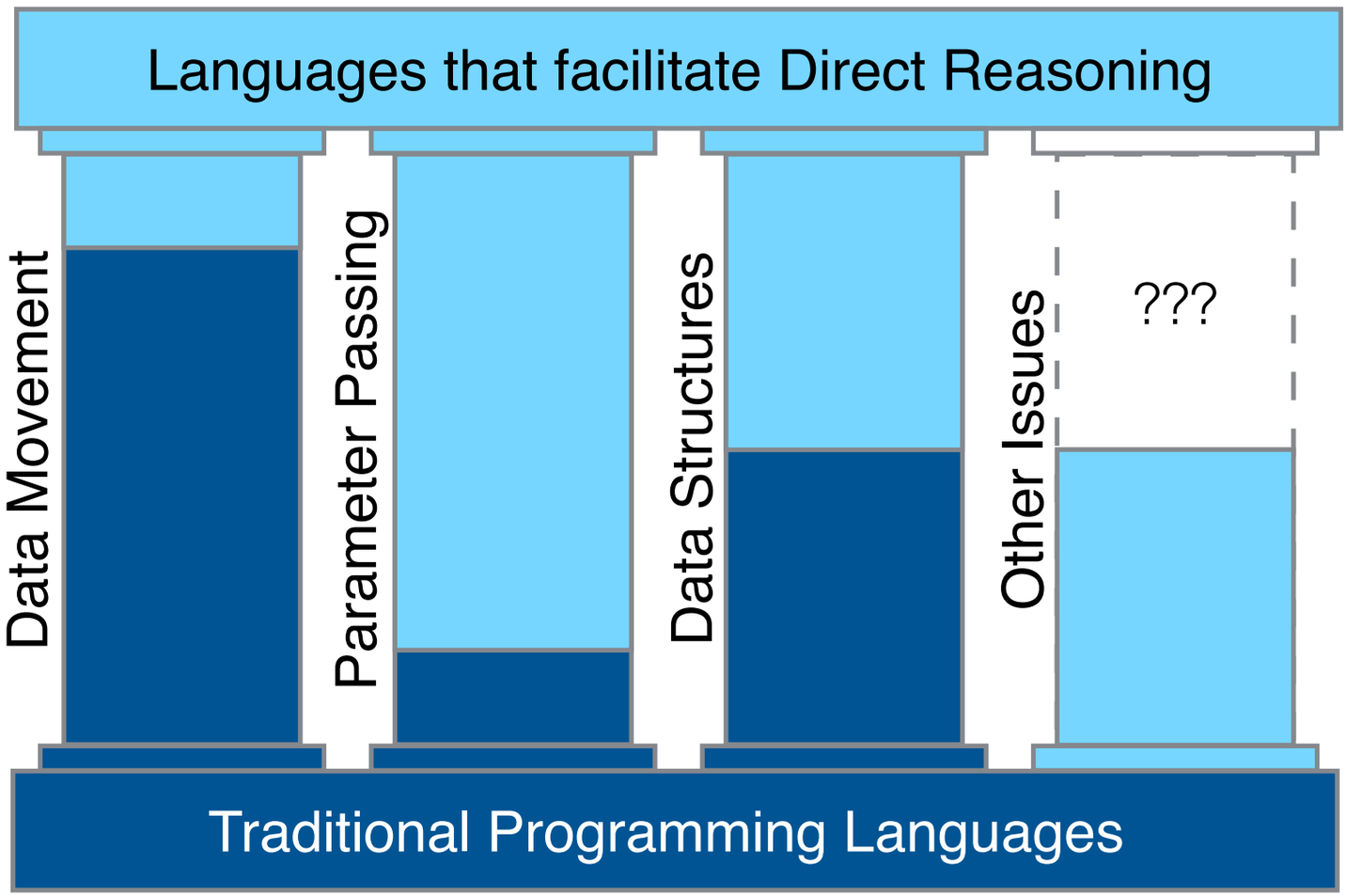
- Introduces notion of *direct reasoning* – where references can be ignored in reasoning; explains its importance to software engineering

- Provides *alternative* to call-by-reference *parameter passing*. No deep copying; works even when arguments are repeated as parameters.

- Introduces *proof rules* for alternative data assignments and a parameter passing proof rule usable with any data assignment.

- Presents a formal specification of a component that captures reference behavior – *Location Linking Template*.

- Applies direct reasoning design principles to Java. *DirectJava* – conceptually direct language that looks like Java.


Languages that facilitate Direct Reasoning
Data Movement
Parameter Passing
Data Structures
Other Issues
???
Traditional Programming Languages

Benefits of References

- Permit constant-time data assignment
- Permit constant-time parameter passing
- Allow implementation of linked data structures
- Can be used to implement object identity
- Useful for certain design patterns

Problems with References

- Traditional reference variables require programmer to keep track of both reference and object value for sound reasoning
- Copying of references causes aliasing, which complicates reasoning of mutable objects
- Introduce memory management problems
- Not persistent

Example

State 1	stack1 = < "A", "B" > stack2 = < >
`stack2 = stack1;`	
State 2	stack1 = < "A", "B" > stack2 = < "A", "B" >
`stack1.pop();`	
State 3	stack1 = < "B" > stack2 = ??

Example

State 1	stack1 = @47 stack2 = @53	@47: < "A", "B" > @53: < >
`stack2 = stack1;`		
State 2	stack1 = @47 stack2 = @47	@47: < "A", "B" > @53: < >
`stack1.pop();`		
State 3	stack1 = @47 stack2 = @47	@47: < "B" > @53: < >

TransferTop Specification

- transferTop(Stack<E> s1, Stack<E> s2)
  - requires s1 ≠ empty-string
  - ensures s1 = all-but-first(s1)
  - ensures s2 = [first(s1)] + s2

TransferTop Implementation: Normal Arguments

State 1	s1 = < "A", "B", "C" > s2 = < "D", "E", "F" >
`String str = s1.pop();`	
State 2	s1 = < "B", "C" > s2 = < "D", "E", "F" > str = "A"
`s2.push();`	
State 3	s1 = < "B", "C" > s2 = < "A", "D", "E", "F" > str = "A"

TransferTop Implementation: Repeated Arguments

State 1	s1 = @10 s2 = @10	@10: < "A", "B", "C" >
`String str = s1.pop();`		
State 2	s1 = @10 s2 = @10 str = "A"	@10: < "B", "C" >
`s2.push();`		
State 3	s1 = @10 s2 = @10 str = "A"	@10: < "A", "B", "C" >

Definitions (example)

Definition 3.5 – A direct view of a variable v is a conceptualization of v that does not involve references

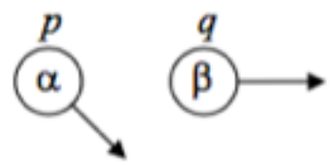
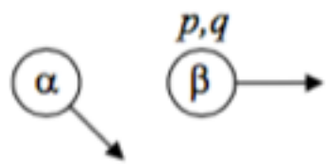
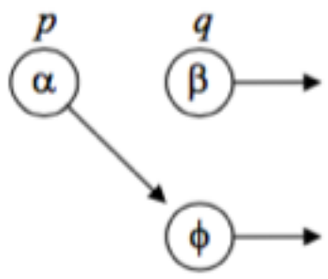
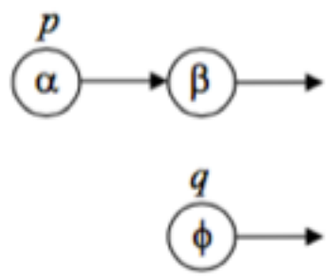
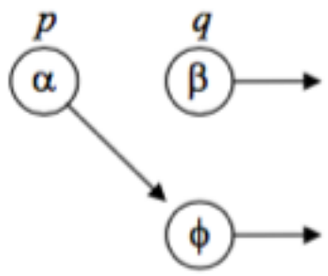
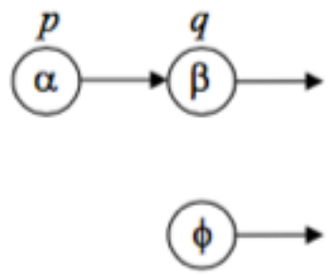
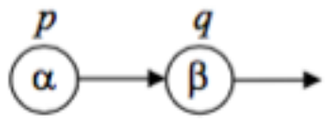
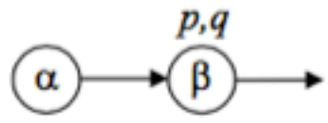
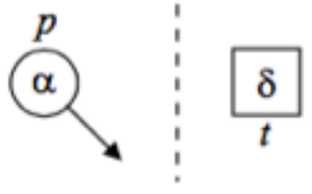
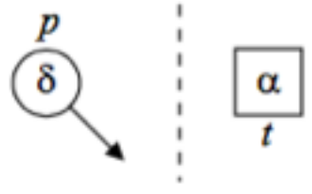
Parameter Passing Proof Rule

We conclude that the procedure call rule using clearing transfer parameter passing and given below is sound and relatively complete.

Context / assertive_code; T1 $\%a_x$; T2 $\%b_y$; $\%a_x \leftarrow\!\circ\, a$; $\%b_y \leftarrow\!\circ\, b$;
confirm pre[x $\leftarrow \%a_x$, y $\leftarrow \%b_y$] **and**
$\forall ?a_x$: T1, $\forall ?b_y$: T2,
post[#x $\leftarrow \%a_x$, x $\leftarrow ?a_x$, #y $\leftarrow \%b_y$, y $\leftarrow ?b_y$] $\Longrightarrow$
Q' [a $\leftarrow ?a_x$][b $\leftarrow ?b_y$];

Context / assertive_code p(a, b); **confirm** Q;

Location Linking Template

Action	Procedure Call	Before	After
p, relocate to *q*'s location.	Relocate(p,q);		
p, redirect your first link toward *q*'s location while *q* jumps to the link's original target.	Swap_Locations(p,1,q);		
p, redirect your first link toward *q*'s location.	Redirect_Link(p,1,q);		
p, follow your first link.	Follow_Link(p,1);		
p, exchange the information at your location with messenger *t*'s information.	Swap_Contents(p,s);		

Direct Java

- Looks like Java
- Swapping for data assignment
- “Clearing transfer” parameter passing
- Location linking component
- Thoughts on design patterns