LangLang Specification

version 0.1

Rehno Lindeque

February 7, 2010

Contents

1 Introduction

This document is intended as a specification for implementers to write implementations against. For language design and rationalle see [1] and [2].

2 Overview

3 General structure

3.1 Source code structure

3.2 Build system structure

3.3 Syntactic Issues

4 Semantics

4.1 Definitions

4.2 Recursive definitions

4.3 Selection queries

4.4 Mutation queries

4.5 Stateful queries

5 Pragmatics

5.1 Processes, External calls, Non-determinism

5.2 Distributed processes

5.3 Live updates

5.4 Proofs

6 Issues

Issue 1.1 – Should nested contexts overide parent contexts or only add information?

Status: Open

Discussion (2010-02-06): Suppose 2 nested contexts, a and b are defined with a simple relation x in each as illustrated bellow.

$$a \rightarrow \left[\begin{array}{l} x \rightarrow 4 \\ b \rightarrow [x \rightarrow 5] \end{array} \right]$$

$a : b : x : 5$ { This will always be true. }
$a : b : x : 4$ { But does x inherrit the relation in $a : x$ as well? }

Example 1: Nested domains with similar variables

A user may expect nesting to override values defined in a parent context, however it seems desirable to have the ability to extend sets from parent contexts.

Futhermore, consider cases in which we expect a relation to be inherited from a parent context.

$$a \rightarrow \begin{bmatrix} x \rightarrow 4 \\ b \rightarrow [x] \end{bmatrix}$$

$a : b : x : 4 \quad \{ \text{ True? } \}$

Example 2: Automatic implication inheritance

This becomes more complicated in a case such as this.

$$a \rightarrow \begin{bmatrix} x \rightarrow 4 \\ b \rightarrow \begin{bmatrix} x \rightarrow 5 \\ c \rightarrow x \end{bmatrix} \end{bmatrix}$$

$a : b : c : x : 5 \quad \{ \text{ True? } \}$
$a : b : c : x : 4 \quad \{ \text{ True? } \}$

Example 3: Automatic implication inheritance

Resolved (2010-02-06 to 2010-02-07): Let us go back to the logical background of contexts. If we say that the symbol $\rightarrow$ means 'implies' then $x \rightarrow 4$ and $b \implies x$ in the same context will tell us denotationaly that $b \rightarrow 4$. (Furthermore the type system will tell us that b implies that 4 is implied by x. I.e. $b \rightarrow {}_{x\rightarrow}4$.) Now, suppose one additional rule $c \rightarrow [x \rightarrow 5]$ is added to the same context. Then c implies that x implies 5. Does x imply both 4 and 5 in the context of c?

Consider a query definition $i \rightarrow b.x$. Translated into english i implies whatever is implied by the predicate $b.x$ (I.e. "x exists in b"). Because if x did not exist in b then we would not expect i to hold the value 4. The fact that we select x from b seems to imply that we are only looking for the specific x that satisfies the type ${}_{b\rightarrow}x$. Once the selection returns ${}_{b\rightarrow}x$ we can look up the symbol's codomain lazily if any exists. In order to select all x we could construct i as $i \rightarrow [x \ b.x]$, hence this method is still general.

However, suppose that we selected x from yet another definition in the contex: $d \rightarrow x$. Now the query $j \rightarrow d.x$ gives us the typed result ${}_{d\rightarrow}x$. However this symbol has no codomains. Our definition of an arrow (the symbol $\rightarrow$) is based on boxing and unboxing. An arrow defines a *labeled box which is automatically unboxed by the compiler in a lazy manner. Hence if no arrow exists, then the compiler will look up one in a parent context. Thus,* ${}_{d\rightarrow}x$ *inherits the relations of* x *in the parent context and* $j : d : x : 4$ *holds. To create a boxed 'empty set' that does not inherit from a parent context we simply use the notation* $x \rightarrow []$.

Discussion (2010-02-07): *How can we extend a symbol in the parent context with additional relations in the child context?*
(See "Unboxing is Evaluation")

Resolved (2010-02-07): *Perhaps it is possible to use the same 'preceding domain' syntax for parent selection as we use for recursion. Take the example below.*

$x \rightarrow 4$
$a \rightarrow [x \rightarrow [5\ @ : x]]$
$a : x : 5 \quad \{ \text{True} \}$
$a : x : 4 \quad \{ \text{True} \}$

Example 4: Explicit inheritance

Issue 1.2 – Perhaps define operators .. and :: as 'deep' selection?

Status: *Open*

References: *Issue 1.1*

Discussion (2010-??-??):

List of Examples

List of Tables

List of Figures

References

[1] Rehno Lindeque. A foundation for programming. 20??

[2] Rehno Lindeque. Rationalle for the programming language, langlang. 20??