Aslan Oztreves

CS-360

Krzysztof Nowak

**Summary of The Language PCF**

PCF stands for “programming with computable functions”. The language PCF is an extension of the simply-typed lambda calculus with booleans, natural numbers, and recursion. It was first introduced by Dana Scott as a simple programming language on which to try out techniques for reasoning about programs. Although PCF is not intended as a “real world” programming language, many real programming languages can be regarded as extensions of PCF as syntactic variants, and many of the reasoning techniques developed for PCF also apply to more complicated languages.

**11.1 Syntax and typing rules**

PCF types are simple types over two base types bool and nat .

A , B ::= bool | nat | A → B | A × B | 1

**11.2 Axiomatic equivalence**

The axiomatic equivalence of PCF is based on the βη-equivalence of the simply- typed lambda calculus.

* All the β- and η-axioms of the simply-typed lambda calculus (page 62)
* One congruence or ξ-rule for each term constructor.

**11.3 Operational semantics**

The operational semantics of PCF is commonly given in two different styles: the *small-step* or *shallow* style, and the *big-step* or *deep* style.

Small-step is closer to the notion of β-reduction that we considered for the simply-typed lambda calculus. Most important difference between an operational semantics and the notion of β-reduction is that operational semantics is going to be *deterministic*, which means, each term can be reduced in at most one way. In other words, it will always be uniquely specified which redex to reduce next.

**11.4 Big-step semantics**

In the small-step semantics, if M ->\* V, we say that M evaluates to V. Note that by determinacy, for every M, there exists at most one V such that M ->\* V. It is also possible to axiomatize the relation “M evaluates to V” directly. This is known as the big-step semantics. Here, we write M ⇓ V if M evaluates to V.

**11.5 Operational equivalence**

Informally, two terms M and N will be called operationally equivalent if M and N are interchangeable as part of any larger program, without changing the observable behavior of the program. This notion of equivalence is also often called observational equivalence.

**11.6 Operational approximation**

As a refinement of operational equivalence, we can also define a notion of operational approximation: We say that M *operationally approximates* N, in symbols M [op N, if for all closed and closing contexts C [−] of observable type and all values V,

C[M]⇓V ⇒C[N]⇓V

**11.7 Discussion of operational equivalence**

Operational equivalence is a very useful concept for reasoning about programs, and particularly for reasoning about program fragments. If M and N are operationally equivalent, then we know that we can replace M by N in any program without affecting its behavior.

**11.8 Operational equivalence AND parallel OR**

We say that a term POR implements the *parallel or* function if it has the following behavior:

POR T P → T, for all P

POR N T → T, for all N

POR F F → F.

We can conclude this summary by saying PCF has been the very bases of a lot of modern languages we use today.