CS 430/530
Formal Semantics

Zhong Shao

Yale University
Department of Computer Science

Coq Basics; Inductive Definitions

September 4, 2013

Language vs. Logic

- A language has “syntax” and “semantics”
- A “logic” is also a language
  - There is a lot more about this … “Curry-Howard correspondence”
- A programming language has
  - “computation” terms and values
  - often with “executable” semantics
- A logic has
  - “computation” terms and values (with slow “executable” semantics)
  - predicates and assertions (about computation terms & values)
  - inference rules & proofs on why an assertion is true

The Big Picture

Object Language:

Predicate Logic [Reynolds Chapter 1]

↓ semantic "interpretations"

Meta Language:

Set Theory [Reynolds Appendix]

Object Language:

Imp [Reynolds Chapter 2]

↓ semantic "interpretations"

Meta Language:

Set Theory [Reynolds Appendix]

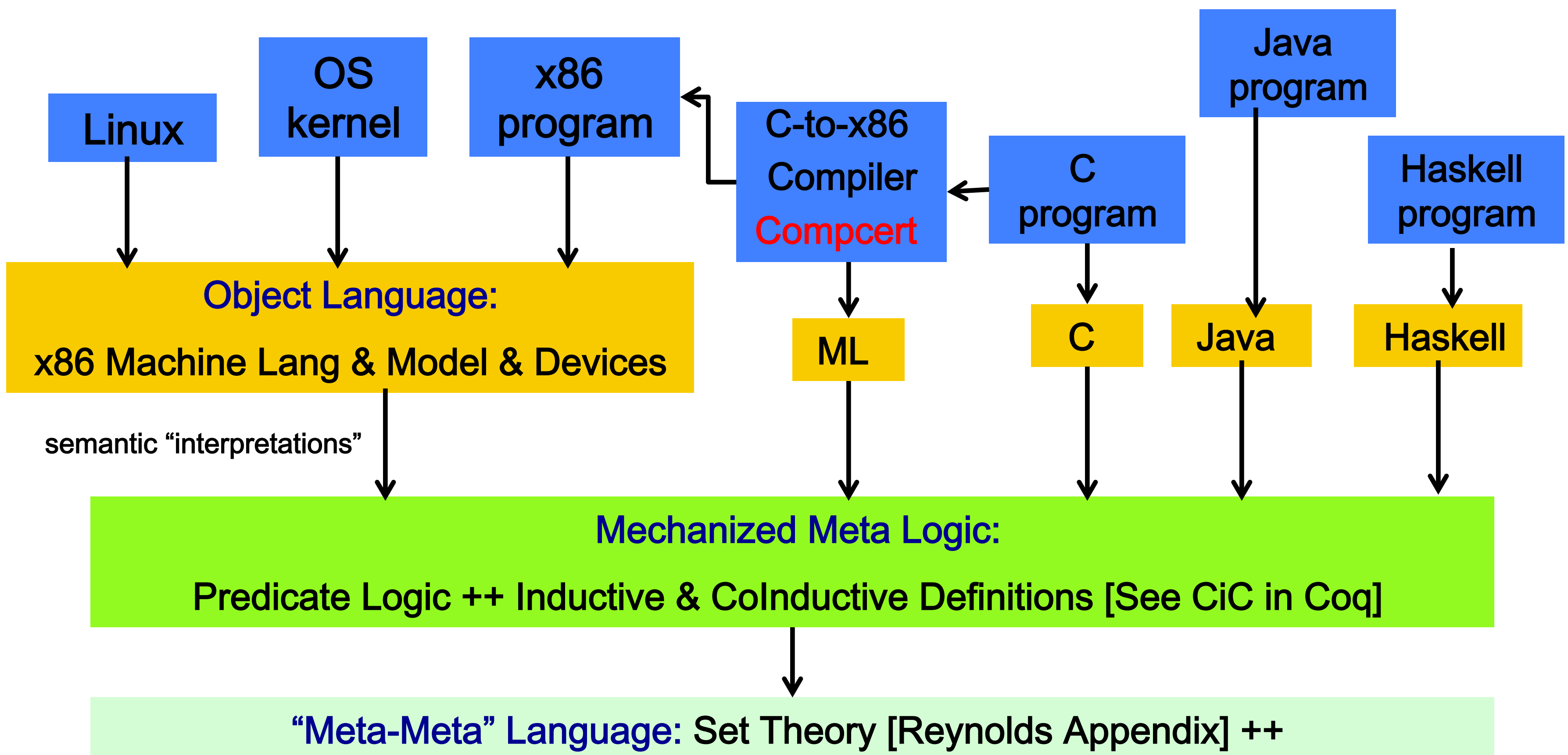
Formal semantics is always about studying the meanings of an object language in a meta language!

Like a compiler or an interpreter.

The Big Picture (cont'd)

Developing the world's most general programming language is hard!

Developing a rich mechanized meta logic to bootstrap the "world" is more feasible



What makes a good “Meta Logic”?

A good meta-logic should be simple & expressive. It has:

- “computation” terms and values (with slow “executable” semantics)
- predicates and assertions (about computation terms & values)
- inference rules & proofs on why an assertion is true

plus a way to introduce user-defined “terms” and “predicates”

- inductive data types & recursive functions
- inductive predicates & inductive proofs

plus a way to reason about blackbox or infinite objects

- coinductive data types (e.g., objects), predicates, and proofs

Inductive Data Types

1.2 Abstract Syntax Trees

An *abstract syntax tree*, or *ast* for short, is an ordered tree whose leaves are *variables*, and whose interior nodes are *operators* whose *arguments* are its children. Abstract syntax trees are classified into a variety of *sorts* corresponding to different forms of syntax. A *variable* is an *unknown*, or *indeterminate*, standing for an unspecified, or generic, piece of syntax of a specified sort. Ast's may be combined by an *operator*, which has both a sort and an *arity*, a finite sequence of sorts specifying the number and sorts of its arguments. An operator of sort s and arity $s_1, \ldots, s_n$ combines $n \geq 0$ ast's of sort $s_1, \ldots, s_n$, respectively, into a compound ast of sort s. As a matter of terminology, a *nullary* operator is one that takes no arguments, a *unary* operator takes one, a *binary* operator two, and so forth.

AST Examples

For example, consider a simple language of expressions built from numbers, addition, and multiplication. The abstract syntax of such a language would consist of a single sort, Expr, and three operators that generate the forms of expression: $\texttt{num}[n]$ is a nullary operator of sort Expr whenever $n \in \mathbb{N}$; plus and times are binary operators of sort Expr whose arguments are both of sort Expr. The expression $2 + (3 \times x)$, which involves a variable, x, would be represented by the ast

$$\texttt{plus(num[2];times(num[3];}x\texttt{))}$$

of sort Expr, under the assumption that x is also of this sort.[1]

Formal Definition of AST

Let $\mathcal{S}$ be a finite set of sorts. Let $\{ O_s \}_{s \in \mathcal{S}}$ be an $\mathcal{S}$-indexed family of *operators*, o, of sort s with arity $\mathsf{ar}(o) = (s_1, \ldots, s_n)$. Let $\{ \mathcal{X}_s \}_{s \in \mathcal{S}}$ be an $\mathcal{S}$-indexed family of *variables*, x, of sort s. The family $\mathcal{A}[\mathcal{X}] = \{ \mathcal{A}[\mathcal{X}]_s \}_{s \in \mathcal{S}}$ of ast's of sort s is defined as follows:

1. A variable of sort s is an ast of sort s: if $x \in \mathcal{X}_s$, then $x \in \mathcal{A}[\mathcal{X}]_s$.
2. Operators combine ast's: if o is an operator of sort s such that $\mathsf{ar}(o) = (s_1, \ldots, s_n)$, and if $a_1 \in \mathcal{A}[\mathcal{X}]_{s_1}, \ldots, a_n \in \mathcal{A}[\mathcal{X}]_{s_n}$, then $o(a_1; \ldots; a_n) \in \mathcal{A}[\mathcal{X}]_s$.

It follows from this definition that the principle of *structural induction* may be used to prove that some property, $\mathcal{P}$, holds of every ast. To show $\mathcal{P}(a)$ holds for every $a \in \mathcal{A}[\mathcal{X}]$, it is enough to show:

1. If $x \in \mathcal{X}_s$, then $\mathcal{P}_s(x)$.
2. If $o \in \mathcal{O}_s$ and $\mathsf{ar}(o) = (s_1, \ldots, s_n)$, then if $a_1 \in \mathcal{P}_{s_1}$ and $\ldots$ and $a_n \in \mathcal{P}_{s_n}$, then $o(a_1; \ldots; a_n) \in \mathcal{P}_s$.