이름이 같은 함수들의 안전성을 Coq으로 증명하기

김 지 응

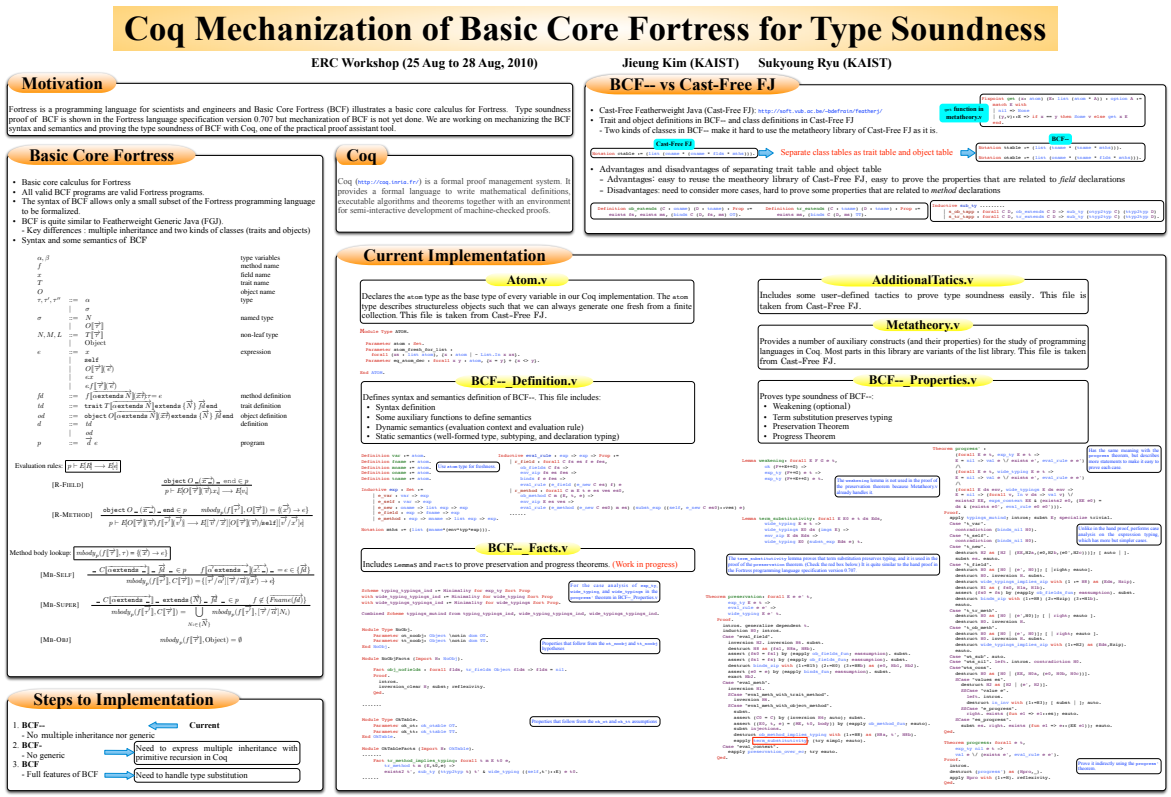
카이스트 프로그래밍 언어 연구실

2011년 1월 7일

이전 발표 내용

Fortress 언어의 핵심 부분(Featherweight Basic Core Fortress: FBCF)에 대한 타입 안전성을 Coq으로 증명

- ROSAEC-2010-010



Coq Mechanization of Basic Core Fortress for Type Soundness

ERC Workshop (25 Aug to 28 Aug, 2010)

Jieung Kim (KAIST) Sukyoung Ryu (KAIST)

Motivation

Fortress is a programming language for scientists and engineers and Basic Core Fortress (BCF) illustrates a basic core calculus for Fortress. Type soundness proof of BCF is shown in the Fortress language specification version 0.707 but mechanization of BCF is not yet done. We are working on mechanizing the BCF syntax and semantics and proving the type soundness of BCF with Coq, one of the practical proof assistant tool.

Basic Core Fortress

- Basic core calculus for Fortress
- All valid BCF programs are valid Fortress programs.
- The syntax of BCF allows only a small subset of the Fortress programming language to be formalized.
- BCF is quite similar to Featherweight Generic Java (FGJ).
  - Key differences : multiple inheritance and two kinds of classes (traits and objects)
- Syntax and some semantics of BCF

Coq

Coq (http://coq.inria.fr/) is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs.

BCF-- vs Cast-Free FJ

- Cast-Free Featherweight Java (Cast-Free FJ): http://sofl.vub.ac.be/~bdefrain/feather/
- Trait and object definitions in BCF-- and class definitions in Cast-Free FJ
  - Two kinds of classes in BCF-- make it hard to use the metatheory library of Cast-Free FJ as it is.

Separate class tables as trait table and object table

- Advantages and disadvantages of separating trait table and object table
  - Advantages: easy to reuse the meatheory library of Cast-Free FJ, easy to prove the properties that are related to *field* declarations
  - Disadvantages: need to consider more cases, hard to prove some properties that are related to *method* declarations

Current Implementation

Atom.v

Declares the `atom` type as the base type of every variable in our Coq implementation. The `atom` type describes structureless objects such that we can always generate one fresh from a finite collection. This file is taken from Cast-Free FJ.

AdditionalTatics.v

Includes some user-defined tactics to prove type soundness easily. This file is taken from Cast-Free FJ.

Metatheory.v

Provides a number of auxiliary constructs (and their properties) for the study of programming languages in Coq. Most parts in this library are variants of the list library. This file is taken from Cast-Free FJ.

BCF--_Definition.v

Defines syntax and semantics definition of BCF--. This file includes:

- Syntax definition
- Some auxiliary functions to define semantics
- Dynamic semantics (evaluation context and evaluation rule)
- Static semantics (well-formed type, subtyping, and declaration typing)

BCF--_Properties.v

Proves type soundness of BCF--:

- Weakening (optional)
- Term substitution preserves typing
- Preservation Theorem
- Progress Theorem

BCF--_Facts.v

Includes `Lemma`s and `Fact`s to prove preservation and progress theorems. (Work in progress)

Steps to Implementation

1. **BCF--** ← Current
   - No multiple inheritance nor generic
2. **BCF-**
   - No generic
   - Need to express multiple inheritance with primitive recursion in Coq
3. **BCF**
   - Full features of BCF
   - Need to handle type substitution

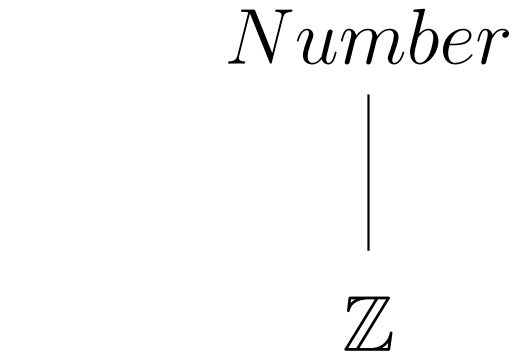
What Is Next?

1. Multiple inheritance
2. Generic types
3. Overloading

Why Overloading?

1. No change in the syntax of FBCF
2. Easy to add or remove rules for valid overloading

Overloading Example



$$f(x : Number,\ y : \mathbb{Z})$$
$$f(x : \mathbb{Z},\ y : Number)$$

Overloading Example

$$f(x : Number,\ y : \mathbb{Z})$$
$$f(x : \mathbb{Z},\ y : Number)$$

$$\boxed{f(x : \mathbb{Z},\ y : \mathbb{Z})}$$

Current Progress

1. Define the syntax and semantics of FBCF with overloading
2. Prove its type soundness by hand (work in progress)
3. Implement it using Coq (work in progress)