Interfaces in Extended ML

Don Sannella

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh
http://homepages.inf.ed.ac.uk/dts

Interfaces in Standard ML

```
signature SIG =
  sig
    type t
    val f : t -> t
  end

structure S : SIG =
  struct
    type t = int
    fun f(x) = x
  end
```

Signatures mediate module interconnections.

Compatibility can be checked (automatically) at compile time = separate compilation.

Interfaces in Extended ML

```
signature SIG =
  sig
    type t
    val f : t -> t
    axiom forall x => f(f(x)) = x
    axiom forall x => exists y => f(y) = x
  end

structure S : SIG =
  struct
    type t = int
    fun f(x) = x
  end
```

From separate compilation (SML) to separate verification.

Checking compatibility requires proof.

Extended ML in an arbitrary institution

Institution (Burstall/Goguen JACM 1992): a particular way of defining a logical system

Requires formal definitions of:

- What is allowed in a signature (type system)
- The syntax and semantics of axioms

Example: first order equational logic

Example: a simple programming language (a definition in a program is a degenerate case of axiom)

What is essential about Extended ML is the module system and how signatures with axioms relate to program modules

- not the stuff that is inside the modules
- not the choice of axiom syntax

This is just a way of decomposing the design of the framework

Multiple institutions

Different institutions may be appropriate for different stages of development

Different institutions for specifying/developing different modules of a multi-paradigm system

Institution semi-morphism INS -> INS' for relating two institutions (INS "richer", INS' "poorer")

Used to make sense of situations in which multiple institutions are involved (e.g. interconnecting different kinds of modules)

Lots of theory has been developed in this context.

It might be useful in connection with concepts.