



Research Consortium in Speckled Computing

Type-Oriented Programming for Specknets

Andy Koppe

University of Edinburgh

andy.koppe@ed.ac.uk



NAPIER UNIVERSITY
EDINBURGH



Motivation

- Embedded systems such as Specks are traditionally programmed in low-level languages such as C or even assembler
- Investigating possibilities of programming Specks in a safer, higher-level language
- Apply and adapt language to network setting

Type-Oriented Programming

- type-oriented as in emphasising the importance of types in programming
- types help to ensure program safety and correctness
- a strong type system catches many errors that can otherwise lead to crashes or incorrect program behaviour
- static type checking finds errors at compile time and thus reduces the need for program testing

Challenges

- higher-level languages incur a performance penalty that must be minimised
- any type system can only approximate programs behaviour and therefore disallows some programs that would execute correctly
 - C solves this by allowing the programmer to circumvent the type system, e.g. through casts and unions
 - the other approach is to make the type system more expressive

TOP language

- class-based type system
- variant type parameters checked with a kind system (i.e. second-level types)
- first-class functions and closures
- data structures immutable by default
- multiple dispatch (“multi-methods”)
- implementation-side type inference
 - i.e. type annotations only in interfaces

Class-based type system

- Classes provide templates for types and objects
 - e.g.: `! :Pair[t<](left:t,right:t);`
- Types are created by filling in parameters
 - e.g.: `Pair[Int]`
- Objects are created by filling in the fields
 - e.g.: `Pair(2,3)`

Subtyping

- Subtyping means that an object of a type can be used in all place where the supertype is required
 - e.g.:
 - ! :Person(name:String, birthday:Date);
 - ! :Student <: Person (course:String);

Methods & multiple dispatch

- methods can have multiple implementations
 - send (Device,Packet) : ();
 - send (Radio,LightPacket) : () = ...
 - send (Radio,SoundPacket) : () = ...
 - send (Laser,LightPacket) : () = ...
 - send (Laser,SoundPacket) : () = ...
- when calling the method, the most suitable implementation is found
- crucial: methods can later be extended with new implementations for new classes

Mutable data

- data structures are immutable by default
 - resulting lack of side effects makes it easier to ensure the correctness of programs
- mutable data support through special class
 - provided for situations, where immutable data structures are less efficient or more difficult to work with

Variant type parameters and kinds

- Variant type parameters allow subtyping in type parameter positions
 - e.g.:
 - ! List[elem<]
 - ! Action[arg>];
 - ! Register[type=];
 - List[Student] < List[Person]
 - Action[Person] < Action[Student]
 - but not: Register[Student] < Register[Person] or vice versa
- Kind system ensures correct parameter use

First-class functions and closures

- allow algorithms to be passed as data
- replace control structures like C's while, if, or for
- control structures are normal methods, so programmers can implement their own

```
if (Bool) (true:{r}) (false:{r}) : r;
```

```
if (True) (true:{r}) (false:{r}) : r = true@();
```

```
if (False) (true:{r}) (false:{r}) : r = false@();
```

Type inference

- only method interfaces have to be explicitly typed
- types in method implementations are all inferred
 - reduces the need for tedious type annotations
 - Java: `Person p = new Person("Smith");`
 - TOP: `p = Person("Smith");`

Module system

- modules organise a program into a hierarchical structure
- programs are translated and checked on a module-by-module basis
- visibility declarations keep program components separated

Code diffusion

- code diffusion allows new functionality to be spread through a network on demand
- when a Speck sees a message containing a class it does not know, it automatically asks the network for the definition of that class
- similarly, when a method is called on a new class, its implementation will be requested from the network
- as a result, functionality will be diffused as needed

Implementation

- Simple compiler with little checking
- Full typechecking close to completion
- Interpreter running on ProSpeckzIII
- Interpreter embedded in Ryan's simulator

Work to be done

- finish compiler
- code diffusion
- good examples

