Types in Modula-3

Amer Diwan

Why bother with Modula-3?

• A small and clean, yet usable object-oriented language
• Language design goal: the entire language definition should fit in 50 pages
• Buzzword compliant—statically and strongly typed, objects, exceptions, threads, garbage collection, modules, generics, ...
Outline

• Modula-3’s notion of types
  – Type equality in Modula-3
  – Subtyping rules
  – Assignment rules (based on subtyping rules)
  – Information hiding in Modula-3

Type equality

• Modula-3 uses structural equivalence
• But structural equality can be “over-ridden” if needed
Subtyping of pointer types

• NULL <: REF T <: REFANY

Subtyping of fixed arrays

• ARRAY I OF T <: ARRAY J OF T
  if NUMBER(I) = NUMBER(j)
Subtyping of object types

- NULL <: T OBJECT ... END <: T OBJECT ... END <: REFANY

Examples

- T1 = OBJECT i: INTEGER; END;
  T2 = OBJECT i: INTEGER; END;
  ST1 = T1 OBJECT j: INTEGER; END;
  ST2 = T2 OBJECT j: INTEGER; END;
  T1 <: T2?
  ST1 <: T1?
  ST2 <: T2?
  ST1 <: T2?
  What happens with branding?
(Simplified) Assignment rules

• Type T is assignable to Type U if
  – T <: U
  – T and U are ordinal types with at least one member in common
  – U <: T and T is an array type or reference type
• Why the exception in the third case?
• Note the implicit safe casts!

Run-time checks

• S <: T;
  VAR s: S; t: T;
  t := s;
• What checks does this need?
• What representation does this need?
Opaque types

• The information hiding mechanism based on subtyping
• TYPE T <: U
  
  U = OBJECT i: INTEGER; END;
  
  T, an opaque type, is some subtype of U
• REVEAL T = U OBJECT j: INTEGER; END;
  
  T is “revealed”: must be consistent with its opaque declaration

Revelations

• Revelations can be incremental
  TYPE T <: U
  
  U = OBJECT i: INTEGER; END;
  
  V = U OBJECT ch: CHAR; END;
• REVEAL T <: V;
  REVEAL T = V OBJECT j: INTEGER; END;
• Can reveal different views to different clients (trusted, etc.).
An example of using opaque types

- INTERFACE Counter;
  TYPE T <: Public;
  Public = OBJECT METHODS next(): INTEGER; END;
  END Counter
- INTERFACE CounterFriends IMPORT Counter;
  REVEAL Counter.T <: U;
  TYPE U = Counter.Public OBJECT last_value: INTEGER; END;
  END CounterFriends
- MODULE Counter EXPORTS Counter, CounterFriends;
  REVEAL T = U OBJECT otherstate: INTEGER; END;
  END Counter.

Continuing with example

- MODULE TrustedClient; IMPORT Counter, CounterFriends;
  BEGIN
  END TrustedClient
- MODULE OtherClient; IMPORT Counter;
  BEGIN
  END OtherClient
Unsafe parts of Modula-3

- Unsafe operations are restricted to modules especially marked as unsafe
  - Explicit deallocation: Untraced references may be deallocated only unsafe modules
  - Unchecked type casts: Called LOOPHOLE!
  - ...

What about other languages?

- Java
  - Similar to Modula-3, BUT rather than specifying subtyping rules, it specifies conversions
  - Different information hiding model
  - Interfaces! (more soon)
- C++/C
  - Unsafe: everything can be converted to everything else using casts
  - Otherwise not too dissimilar to Java
Next topic: Object orientation

• Reading: