Execution Model

Designing the “object” program and preparing to create it

Looking all the way forward ...

• There could be many steps between a type-checked AST and object code
  – Generation of intermediate representation, improvement (optimization), register allocation, more improvement ...
  – (or not)

• But we need a clear idea of where we are going (at least in outline)
Designing the Abstract Machine

• (Mostly) hardware-independent decisions
  – What does an activation record look like?
  – What does memory look like? What’s in the heap? ...

• Defer Hardware-dependent decisions
  – How many registers do we have? How do we allocate & reclaim heap? Which operations are library subroutine calls? ...
Contour Model of Memory

Activation record for one procedure; has control point (code pointer), local memory, non-local references

Object code. Usually boring, unless we have dynamic loading.

Heap. What invariants does it maintain? Can it have references to contours? To code?

Important Properties of Activation Records

• Activation protocol:
  – Strictly LIFO (stack-like) call/return?
    • Anything remembered from one call to the next?
    • Co-routines (e.g., generators) or threads?
    • Non-local goto (e.g., exceptions)? How?
  – Argument passing by value, ref, copy in-out?
  – How is run-time scope established?
    • Non-local variable references? “This” object?
The Activation Stack (or?)

- Simple call/return follows a stack discipline
  - Easy, efficient to implement; usually well-supported by hardware
- Look for deviations from stack discipline
  - E.g., references from calling context to called context (closures)

Typical Stack of Activation Records (a.k.a. “stack frames”)

- Allocate activation record when method/procedure called
  - Includes return address, pointer to previous stack frame and/or saved registers
- Simply abandon when procedure returns
  - Reuse space on next call
Variations on LIFO Call/Return

• Threads (as in Java, Ada, ...
  – Need a stack per thread, at least
• Generators (co-routines)
  – a la Icon, and more recently Python; what
    Java iterators would do if they could
• Exceptions / catch-throw
• Closures

(Aside: Cool doesn’t have these, but
they are worth knowing about.)

Coroutines / Generators

• What’s wrong with iterators in Java?
  – From the caller’s point of view, not much
    • Clumsy syntax, but standardized
  – From the iterator writer’s side, plenty
    • A natural traversal method must be “inverted”,
      storing the “where I left off” marker in the
      iterator data structure
• Alternative: “yield” instead of “return”
Generators

while (p != null) {
    yield p.item;
    p = p.next;
}

What’s in a Frame?

A context by any other name would run as sweet.

- Leaving off physical layout for now ...
- We need:
  - Calling context for the return (control point, any state we might clobber)
  - Parameters (references? values?)
    - Including implicit parameters, e.g., “this”
  - Storage for local variables
    - Pointers or actual values?
Arguments

• By reference: Pointers
  – Not in Java, C, or Cool (sort of)
• By value
  – But in Java, C, and Cool, the “value” of an object is a reference (not so in Pascal, Ada)
• Later: Implementation choice
  – Push on stack? Pass in registers? Who saves?

Local Variables

• In C
  – both primitives and structures “live” in the stack; no restriction on pointers
• In Java
  – Primitive types (int, bool) live on stack, reference to objects live on stack, but objects always live in the heap
  – NO program-controlled pointers to the stack
• In Cool: Like Java, but everything is an object
Activation Record Design

Local storage in C: frame may contain complex and simple objects, pointers to and from heap without restriction.

Local storage in Cool: frame contains only references to objects in heap. No user-controlled pointers to stack are possible.

Interpreter or Compiler?

• Usually some of each
  – Even interpreters do some translation, for efficiency
  – Even compilers use library routines to interpret some features, like built-in operations in Cool

• Activation record design applies to both
  – Except parts may be implicit in interpreter, e.g., interpreter AR stack