Languages with Context

A Block Structured Language

SE424

Languages with Contexts

- In any language, the context of a phrase influences its meaning.
- In p rogramming languages, contexts attribute meanings to identifiers.
- Does the store establish the context?
  begin integer X; integer Y Y:=0 X:=Y Y:=1 X:=Y+1 end
- But the store constantly changes within the block!
- Declarations establish block context.
- Commands operate within that context.

SE424

Languages with Contexts

```
begin integer X X:=0
    begin real X X:=1.5 end
    X:=X+1
end
```

- Outer X denotes integer object.
- Inner X denotes real object.
- X is the name used for both objects.
- Scope rules are needed to resolve ambiguities.
- Meaning of an identifier can't be just its storable value!

SE424

Contexts

- “Objects" are computer store locations.
- The meaning of an identifier is the location bound to it.
- A context is the set of identifier/ store location pairs accessible at a textual position.
- Each position in a program resides within a unique context.
- Context of a phrase can be determined without running the program.

SE424

Environments

Mathematical value that models context.

- Environment establishes context for syntactic phrase
  - Resolution of ambiguities concerning meaning of identifiers.
- As many environment values as distinct program contexts.
  - Multiple environments are maintained during program evaluation.
- Environment is a static object.
  - Phrase uses the same environment each time it is evaluated.

SE424

Environment Models

- Simple model: store = environment
  - Only one environment.
  - Used in previous program examples.
- Complex model: store plus environments.
  - One store
  - Multiple environments

SE424

Compiler's Symbol Table

- An example of an environment
- Used for translation of source program into compiled code.
- One entry for each identifier in the program
  - Data type.
  - Mode of usage(variable,constant,parameter).
  - Relative location in run-time store.
- Resolution of name conflicts
  - Different symbol table for each block.
  - Build table as single stack
    - Incremented and decremented upon block entry and exit.
- Compile-time object (Pascal, C, C++),
- Run-time object (Lisp, Smalltalk).

Static and Dynamic Semantics

- Static semantics
  - Part of semantics definition that use environment to resolve context questions.
  - Type-checking, scope resolution, storage calculations.
- Dynamic semantics
  - "Real" production of meanings.
  - Code generation and execution.

No clear separation!

Evaluation Functions

- **Environments used as additional argument**

  C: Command → Environment → Store → Store⊥
- **Environment domain**
  - Environment = Identifier → Denotable value

Language Features

1. Declarations.
2. Block structures.
3. Scoping mechanisms.
4. Compound data structures.

A Block-Structured Language

- See Figures 7.1 and 7.2 \017
- Composition of commands C
- [[C1;C2]]= λe(C[[C2]]e)o(C[[C1]]e)
- Both C1 and C2 are evaluated in e.
- C1 may contain local declarations.
- Environments created within C1 do not affect C2.
- Static scoping
  - Context of phrase solely determined by its textual position.
  - Identifier declared within block only referenced by commands within that block.
  - Straight-forward management of storage locations.

Strongly Typed Languages

- Environment processing can proceed independently of store processing.
- i.e. P[[P]] can be simplified without value of initial base location l and initial store s. Result neither contains occurrences of environment arguments nor checking of denotable and expressible value tags.
- Simplifications correspond to declaration and type-checking actions in a compiler.

Example See Figure 7.3

λ l.(λ s. return (update l (one plus two) s))
! (check (fix (λ f. λ s.
    ((access l s) equals zero →
        (λ s. return(update (next-locn l)
        (access l s) s)) ! (check f)
    [] return
) s)))
  ! (check (λ s. return (update l one s)))
(f ! g := g o f)
Resembles series of machine code instructions parameterized on the store's base address l .

Stack-Managed Storage

- Store of block-structured languages always used in a stack-like fashion
  - Locations are bound to identifiers sequentially using 'nextlocn'.
  - Location bound to identifier in local block are freed for re-use when block is exited.
- Storage reuse automatically in C[[C1;C2]]
  - Locations bound to identifiers in [[C 1]] are re- served by environment built from e for C [[C 1]] .
  - C [[C 2]] uses original e (and original location marker) effectively deallocating these locations.
- Significant characteristic of block-structured languages!

Stack-Managed Storage

- Make storage calculations explicit in Store algebra
  - Store = (Location → Storable-value) x Location {
  - First component is data space of the stack
  - Second component indicates “top of stack"
  - 'allocate-locn' becomes the run-time version of 'reserve-locn'
- Environment domain is freed from storage management
  - Environment = Id → Denotable-value
  - 'reserve-locn' is dropped.
- Processing of declarations requires store as well as environment.

Stack-Managed Store

- Declarations
  - D: Declaration → Environment → Store → (Environment x Poststore)
  - D [[va r I]] = λ e. λ s. let (l ,p) = (allocate-locn s) in ((update env[[I]] inLocation (l) e), p)
  - D [[D 1 ; D 2]] = λ e. λ s. let (e’,p) = (D [[D1]] e s) in (check D [[D 2]] e’)(p)

- Blocks
  - K [[begin D;C end]] = λ e. λ s. let l = mark-locn in
    let (e’,p) = D[[D]] e s in
    let p’= (check (C [[C]] e’))(p) in
    (check (deallocate-locn s l))(p 0)
- Environment becomes run-time object because binding of location values to identifiers depends on run-time store.

The Meaning of Identifiers

- Assignment X := X+1
  - Left-hand side value of X is location,
  - Right-hand side value is storable value associated to location.
- Context problem occurs even at primitive command level!
- Variable identifier denotes pair of values
  - I[[I]] = (L-value, R-value)
  - L-value is kept in environment.
  - R-value is kept in store.

The Meaning of Identifiers

- Valuation I : Id → Environment → Store → (Location x Storable-value)
  - L : Id → Environment
  - Store → Location.
  - R : Id → Environment → Store → Storable value.
  - L[[I]] = accessenv [[I]]
  - R[[I]] = access o accessenv[[I]]
  - Semantic equations with variables
    - E[[I]] = R[[I]]
    - C [[I:=E]] = λ e. λ s. return(update(L [[I]] e) (E [[E]] e s) s)

The Meaning of Identifiers

- Other view: R-value = Function(L- value)
  - “True" meaning is L-value.
  - “Coercion" on right-hand side of assignment
- Coercion = dereferencing.
  - J : Id → Environment → Denotable-value
  - J [[I]] = λ e. (accessenv [[I]] e)
  - C [[I:=E]] = λ e. λ s. return (update (J [[I]] e) (E [[E]] e s) s)
  - E [[I]] = λ e. λ s. access (J [[I]] e)s
- Some system languages require explicit dereferencing operator (X=@X+1)
  - E [[I]] = λ e. λ s. in Location (J [[I]] e)
  - E [[@E]] = λ e. λ s. cases (E [[E]] e s) of

  is Location (l) →(access l s) [] . . . end