Orthogonality of language features: The ALGOL 68 experience

“If you call me by name, it’s “Veert,” but if you call me by value, it’s “Worth.”
— Niklaus Wirth, when asked how to pronounce his name

“What’s in a name? That which we call a rose by any other name would smell as sweet.”
— Juliet

Contributions to programming

New features that exceed those of ALGOL 60:
— Constant declarations
— User-defined data structures (e.g., records, unions)
— Dynamic memory allocation
— Expressions on left-hand side of assignment operator
— Array slices
— Coercions
— Assignment statements are expressions that return values

Heavily influenced the design of the C language

ALGOL 68 design goals

Orthogonality of semantic machinery
— Motivated by ambiguities arising from feature composition in ALGOL 60
— Define a small number of semantic concepts that compose with one another in meaningful ways

Formal definition of syntax and semantics

Static type (mode) checking

Key distinctions

Distinction between syntactic entities (phrases) and semantic entities (values and actions)
— Action is a process that constructs and manipulates values
— Phrase is the piece of text a programmer writes
— Phrases are elaborated to perform actions

Document organized around semantic entities, in sharp contrast with ALGOL 60 report

Key semantic distinctions

Categories of entities in the language specification
— Hardware representations (left unspecified)
— Internal objects (opaque, semantically meaningful)
— External objects (things the programmer can “see” or manipulate)

Example: Values vs. expressions
— Value: Internal object, represented by a bit pattern, which is operated upon when program is elaborated
— Expression: Phrase (i.e., an external object) that yields a value

What’s in a name?

A value?
A set of values?
A memory location?
A set of memory locations?

ALGOL 68 makes fine distinctions between these roles, which were not distinct in ALGOL 60
Values and identifiers

Consider the ALGOL 60 declaration:

```algori
real x;
```

This declaration serves two purposes:
- To reserve a memory location that can store values of type `real` and
- To associate the identifier `x` with that location

In ALGOL 68, these purposes are seen as orthogonal and are treated separately
- Memory locations are hardware representations of names, which
  are values that refer to other values
- Identifiers associated with values via identity declarations

Creation and use of names

Names created by expressions called generators:
- `loc μ` creates a local name that refers to values of mode `μ`
- `heap μ` creates a global name that refers to values of mode `μ`
- local names correspond to locations in automatic storage
- global names correspond to locations allocated off the heap

Assignment makes a name refer to another value:
- `N := E`
  - `N` is expression of mode `ref μ` and `E` an expression of mode `μ`
- E.g. `loc real := 3.2`
- Note: name refers to new value; identifier that possesses the
  name does not then possess the new value

Identity declarations

Define the value possessed by an identifier

Syntactically:
- `μ s = E`
  - where `μ` is a mode, `s` an identifier, and `E` an expression
  of mode `μ`
- E.g., `real pi = 3.1415927`

Note: Not a variable declaration!
- E.g., assignment `pi := 3.14` is not meaningful
- More powerful, as the value could be of any mode, including a procedure

Example

Identity declaration:

```algori
ref real x = loc real
```

is roughly equivalent to the ALGOL 60 declaration:

```algori
real x
```

Question: What is the ALGOL 68 equivalent of:

```algori
real x := 2.4
```

Coercions

**Defn:** Language-defined transformation of the mode of an expression into another mode.
- Indicated by the context of an expression
- Usually specifies an action for transforming a value of
  former mode into a value of the latter

Examples:
- **Dereferencing** an expression of mode `ref μ` in a
  context that expects mode `μ`
  - `x := y` where `x` and `y` of mode `ref real`
- **Proceduring** an expression in an identity declaration
  ```algori
  proc real p = 3.14 + x
  ```

Construction/application of routines

Programmer constructs routine by writing suitable phrase

```algori
preceded by list of formal parameters
```

- E.g., `(bool, bool) bool : if a then true else b fi`
- Denotation (as above) only way to construct a procedure

Application:
- When routine is called, parameters are elaborated before the body
  of the routine is elaborated.
- Actual parameters bound to formal parameters using identity
  declarations, which are introduced to precede the body of the
  routine prior to elaboration.
- Advantage: Much cleaner semantics of procedure calls than we saw
  for ALGOL 60
Procedure calls in detail

Procedure call is phrase of the form:
- \( P(E_1, E_2, \ldots , E_n) \)
  - where \( P \) is of mode
    - \( \text{proc} \left( \mu_1, \mu_2, \ldots , \mu_n \right) \mu_0 \)
    - \( \text{proc} \left( \mu_1, \mu_2, \ldots , \mu_n \right) \)
  - and each \( E_i \) is an expression of mode \( \mu_i \)

Value yielded by \( P \) is a denotation, such as:
- \( \langle \mu_1, \mu_2, \ldots , \mu_n \rangle : E \)

Call specifies an action which:
- Transforms routine into the phrase:
- \( \langle \mu_1, \mu_2, \ldots , \mu_n \rangle : E \)

Example of procedure call

Consider the program snippet:
- \( \text{real } x := 30.5; \)
- \( \text{real } y := 20.2; \)
- \( \text{proc (proc) } f = (\text{proc } a) : x := 2 \cdot y; a \)
- \( x := f (\text{any}) \)

Statement with procedure call elaborated as
- \( x := (\text{proc } a = x + y; x := 2 \cdot y; a) \)

When this term is elaborated, it yields value 60.6, not 50.7
- Actual parameter \( (x + y) \) is procedured by identity declaration
  and associated with identifier \( a \)
- Use of \( a \) then deprocedured into an expression of mode \( \text{real} \)

Idiom simulates call-by-name of Algol 60

Discussion

Semantics of procedure calls:
- Awkward in ALGOL 60; clean in ALGOL 68.
- Why?

Importance of coercions:
- Absolutely necessary to make the language usable
- Seem to occur everywhere!
- Is this good or bad?

Orthogonality and usability