The BETA programming language

Bent Bruun Kristensen, Ole Lehrmann Madsen,
Birger Møller-Pedersen Kristen Nygaard
Beta is a language in the SIMULA 67 tradition

Supports object-oriented perspective on programming

Replaces classes, procedures, functions and types by a single abstraction mechanism called the pattern

Objects: instances of patterns
Objects and patterns

Object descriptor have the following form:

( #
D_1; D_2; ...D_n
enter In
do Imp
exit Out
# )

A pattern consists of a named object descriptor:

C : (#...
# )
Evaluations

\[ x + dx \rightarrow x \]

General form

\[ E_1 \rightarrow E_2 \rightarrow ... \rightarrow E_n \]
Control Structures

(if \( E_0 \)
//\( E_1 \) then \( I_1 \)
//\( E_2 \) then \( I_2 \)
//\( E_n \) then \( I_n \)
if
Object kinds and construction modes

3 kinds of objects: system, component and item

- A system object may be executed concurrently with other system objects

- A component object (coroutine) may alternate execution with other components

- An item object is a dependent action sequence contained in a system, component or item
Static Items

$E : \mathcal{C}P$

Inserted Items

$E \rightarrow C \rightarrow A$
Dynamic Items

Dynamic reference:

\[ X \uparrow P \]

\( X \) may refer to a \( P \)-item, a sub-item of \( P \) or NONE

A dynamic \( P \) item may be generated by execution of a “new” imperative: \&\( P \)

A dynamic reference may be given value in the following way:

\&\( P \uplus \rightarrow X \uplus \)
Classification Hierarchies

superpatterns

*Record*: (#*Key* : ©*Integer#ad);
*Person*: *Record* (#*Name* : ©*String#ad)
*Employee*: *Person* (#*Salary* : ©*Integer#ad)
Virtual Patterns

\[ V :< A \]

\( V \) is declared as a virtual pattern with qualifying pattern \( A \)

\( V \) may be bound to any subpattern of \( A \)

The virtual pattern attributes of a pattern \( P \) may be bound in subpatterns of \( P \). A binding of a virtual pattern may have the form of a final binding

\[ V :: A1 \]
Multi-Sequential Execution

- Sequential
- Alternation
- Concurrency
Concurrent execution

\[ Slave_1 : \emptyset \parallel Slave \]
\[ Slave_2 : \emptyset \parallel Slave \]
\[ Master_1 : \emptyset \parallel Master \]

\[ \ldots \]
\[ (\parallel Master_1 \parallel Slave_1 \parallel Slave_2 \parallel) \]
Communication:

Similar to rendezvous in Ada

Sender:

...*Receiver >?M*...

Receiver:

...(#M : @(#... with Sender do ...#)...)... do ... <?M...
The some construct
Compound Objects
Alternating Execution of Components