Jose E. Marchesi

GNU Project

FSCONS 2011

Outline

1. A bit of history
2. The language
3. The World Domination Plan

Three magical decades

- **1950-1960 Discovery and Description**
  - Symbolic assembly languages and macro-assembly languages.
  - Basic concepts.
  - Basic implementation techniques.
  - Fortran, Algol60, Cobol, Lisp.
  - Language = Tool.
- **1961-1969 Elaboration and Analysis**
  - Models and theories. Program correctness.
  - PL/I, Algol 68.
  - Language = Subject of study.
- **1970-1980 Technology**
  - Less abstraction and elaboration.
  - More technology of programming.
  - Complexity barrier.
  - Simplicity.

And then...

- **1981-2011 Decadence... ;)**

The First Generation

- **Fortran**: the practical milestone.
- **Cobol**: data description facilities.
- **Lisp**: simplicity and power.
- **Algol 60**: the conceptual milestone.

Algol 60

- BNF used to describe syntax.
- English used to describe semantics.
- Very uniform structure in the report:

```
1.2. Feature
1.2.1. Syntax
       <BNF description>
1.2.2. Examples
       <Usage examples>
1.2.3. Semantics
       <English descriptions>
```

- Algol 60 Lawyers $\rightarrow$ Algol 60 Theologians.

The Second Generation

Two ways of evolving languages:

- By synthesis: **PL/I**
- By generalization: **Algol 68**

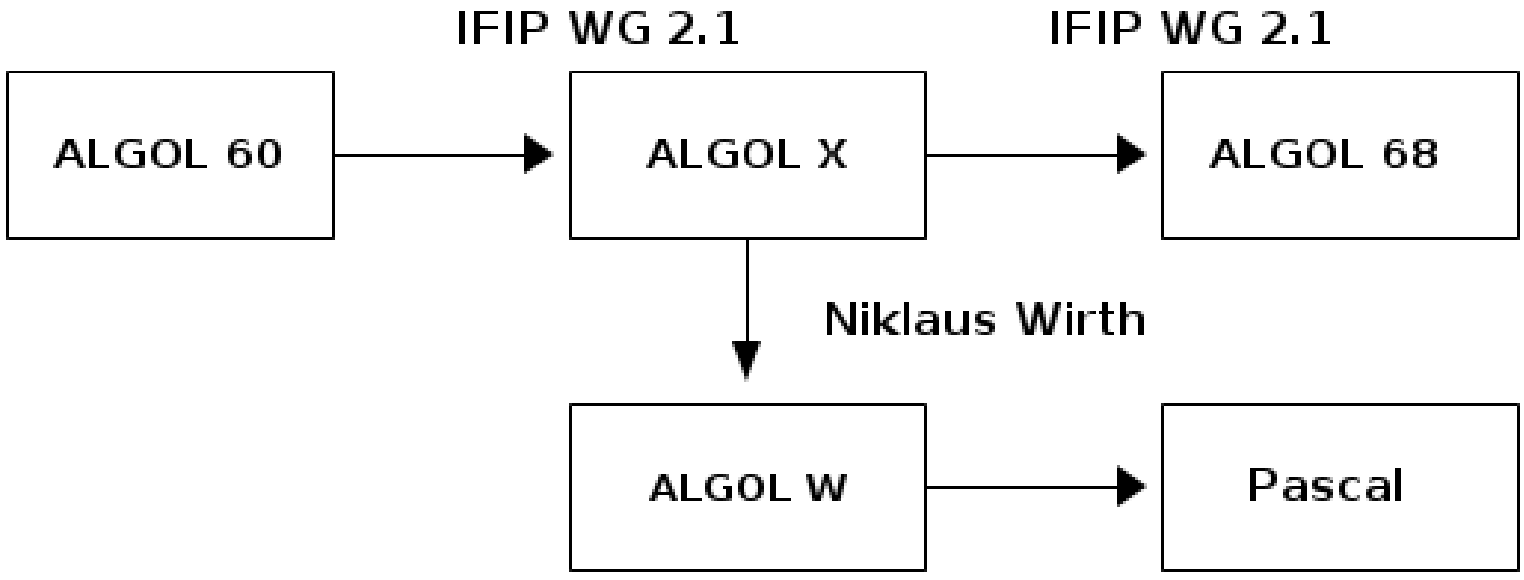
IFIP Working Group 2.1

- **Working group on Algorithmic Languages and Calculi**.
- IFIP: International Federation for Information Processing.
- http://www.cs.uu.nl/wiki/IFIP21/
- 1962 Rome meeting $\Rightarrow$ The Revised Report on Algol 60.
- Support and maintenance of Algol 60.
- Ultimately forked by IFIP Working Group 2.3 on Programming Methodology.

Generalizing Algol 60

- Theologians discovered many problems in Algol 60.
- This leaded to several new languages, and religion wars!
- Aadrian Van Wijngaarden.
- Minority report.
- Dijkstra: "Congratulations, your Master has done it"

Generalizing Algol 60



Algol 68

- Generalization of Algol 60.
- Extreme orthogonality.
- Never widely adopted, because:
  - VW (two-level) grammars.
  - Lack of adequate implementation.
  - Lack of user manuals.
  - New terminology.
  - Too general and flexible?

Orthogonality

- Independent ideas which are developed and applied with generality.
- $N + M$ rules, not $N * M$
- Example:
  - Variables can be passed to procedures.
  - Procedures can be assigned to variables.
  - $\Rightarrow$ Procedures can be passed to procedures.

Some Algol 68 terminology

- Modes (types)
- Moid = mode or indicant.
- Elaboration = run.
- Multiples.
- Go-on symbol: ;
- ... and many more weird stuff...

Hello, world!

```
PROGRAM hello world CONTEXT VOID
USE standard
BEGIN
   print (("Hello world!"))
END
FINISH
```

Stropping

```
INT a var := 10; IF a var > 5 THEN a var := 5 FI

'int' a var := 10; 'if' a var > 5 'then' a var := 5 'fi'

.INT. A VAR := 10; .IF. A VAR > 5 .THEN. A VAR := 5 .FI.
```

Denotations

- Integers: 1, 10, 1e6.
- Real numbers: 3.14, 1.602 10 e-19.
- Characters: “a”, “b”.
- Strings: “foobar”
- Row-displays: (10, 20, 30)
- Structure-displays: (10, “twenty”, 0.10)

Identity declarations

```
INT number of eyes = 2

[]INT list = (1, 2, 3)

[,]INT matrix = ((1, 0, 0),
                 (0, 1, 0),
                 (0, 0, 1))

STRUCT(INT i, STRING name) s = (10, ‘‘a name’’)
```

Names and generators

```
REF INT counter = LOC INT

REF []INT i7 = LOC[1:7]INT

REF FLEX[]INT fn = LOC FLEX[1:0]INT

REF STRUCT(INT i, STRING name) s = HEAP STRUCT(INT,STRING)
```

Assignment

```
REF INT counter = LOC INT := 10;
REF INT another counter = LOC INT := counter
```

References: names storing names

```
REF INT a variable = LOC INT := 10;
REF INT another variable = LOC INT := 20;
REF REF INT a ref = LOC REF INT := a variable;

a ref := a variable         # REF REF INT := REF INT #
a ref := 66                 # REF REF INT := INT #

a ref := another variable # REF REF INT := REF INT #
a ref := 77                 # REF REF INT := INT #
```

Some syntactic sugar

```
REF INT i = LOC INT
REF INT a = HEAP INT := 10
REF REF INT pointer = LOC REF INT

REF[]REAL r = LOC[3]REAL := (1.0, 2.0, 3.0)
```

... can be written as...

```
INT i;
HEAP INT a := 10
REF INT pointer;

[3]REAL r := (1.0, 2.0, 3.0)
```

Scope and range

- Values have scope.
- Identifiers have range.

```
INT j;
BEGIN
   HEAP INT i := 10;
   j := i
END
```

Coercions and Contexts

- Coercions
  - Voiding
  - Rowing
  - Widening
  - Uniting
  - Deproceduring
  - Dereferencing
  - Weaking-dereferencing
- Contexts
  - Strong
  - Firm
  - Meek
  - Weak
  - Soft

Dereferencing

```
REF FOO := REF FOO
```

becomes...

```
REF FOO := FOO
```

Deproceduring

```
PROC FOO
```

is elaborated to...

```
FOO
```

Operators

- Operator names are like MOIDS.
- Unary operators.
- Dyadic operators with priority from 1 to 9.
- Predefined: arithmetic, exponentiation.
- Can be overloaded.

Multiples (Arrays)

- Algol 68 has a very rich support for arrays.
- Multidimensional: `[]INT`, `[,]INT`, `[,,]INT` ...
- Slicing

```
[,]INT arr = ((1, 2, 3),
              (4, 5, 6),
              (7, 8, 9));

arr[1,];    Yields (1, 2, 3)
arr[,1];    Yields (1, 4, 7)
```

- Trimming:

```
[]CHAR quote =
  "There no system but GNU and Linux is one of its kernels";

quote[:5];                  Yields ‘‘There’’
quote[7:8];                 Yields ‘‘no’’
quote[UPB quote - 7:];      Yields ‘‘kernels’’
```

Blocks

- Also known as “Enclosed clauses”.
- Must contain at least one unit.
- Can be nested.
- Two alternative notations:
  - `BEGIN..END`
  - `(..)`
- Examples:

```
BEGIN INT i := 10; print ((i)) END

(INT i := 10; print ((i)))
```

Conditional Clauses

```
IF a THEN
   b
ELIF c THEN
   d
ELSE
   e
FI
```

```
(a | b |: c | d | e)
```

Multiple Conditional Clauses

```
CASE a IN
  u1,
  u2,
  u3
OUT
  u4
ESAC
```

```
(a | u1, u2, u3 | u4)
```

Iterative Clauses

```
FOR a IN 10 TO 20 WHILE cond
DO
  ...
OD
```

Procedures

```
PROC fibonacci = (INT num)INT:
BEGIN
   IF fibonacci < 2 THEN
      num
   ELSE

END

PROC fibonacci = (INT num)INT:
   (n<2 | n | fibonacci(n-1) + fibonacci(n-2))
```

Parallelism!

```
PAR BEGIN
      SEM a semaphore;

      u1;
      u2;
      ...
    END
```

World Domination Plan

1. Write an Emacs mode for Algol 68.
2. Get an Algol 68 compiler which works in modern computers.
3. Integrate that compiler with gcc.
4. Proof of concept: gcc frontend written in Algol 68.
5. Write an Algol 68 frontend for gcc.
6. Assemble a GNU Working Group 2.1.
7. Evolve Algol 68 (and implementation) into GNU Algol.
8. Profit!!

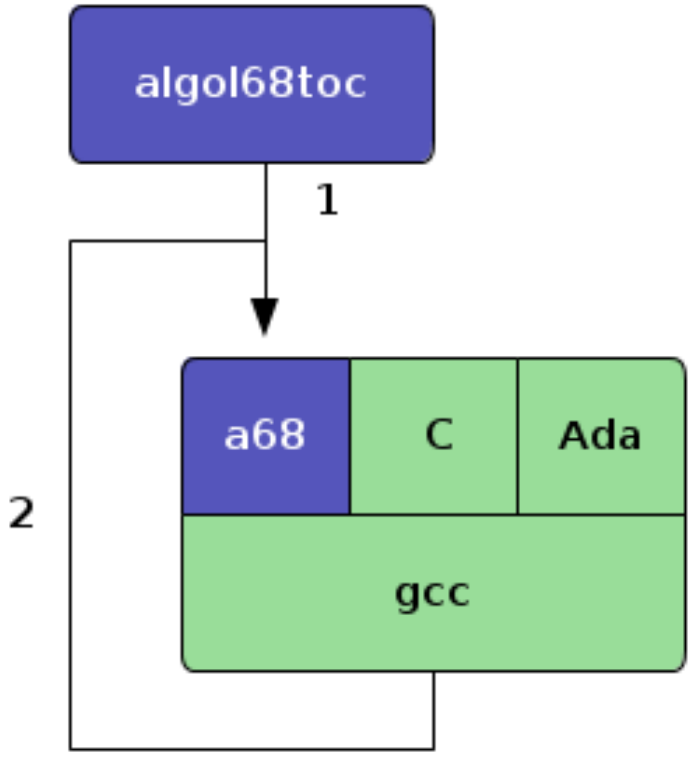
The Algol 68 Emacs mode

- Font locking support.
- Indentation using SMIE.
- Available at http://www.jemarch.net/a68-mode.html

The ctrans compiler

- Free software Algol 68 RS compiler by RSRE.
- ELLA: hardware description language.
- Standard prelude: QAD (quick-and-dirty) by Dr. Sian Leitch.
- Developed during the 1980s and 90s.
- Translates to C, portable.
- Uses the esoteric RS modules system, with restrictions.
- Several restrictions to the language: no PAR, etc.
- Old and funny.
- Supports 64bits, but only since yesterday :D

Bootstrapping the frontend



The Algol 68 gcc frontend

- Currently a Brain  interpreter... ahem..
- `a681.c` entry points.
- `gccaliens.a68` ctrans $\rightarrow$ interface.
- `a68lang.a68` gcc hooks.
- `Make-lang.in` build rules.
- `...`

Evolving the language

- Enumerated values

  ```
  MODE fruit = ENUM (Apple, Orange, Blah)
  ```

- Support for generics.
- Support for _ in mode indicants (a68g extension).
- Support for _ in identifiers (a68g extension).
- Range types.
- Named arguments.

  ```
  say hello (message => ‘‘Hello!’’, indent => 10)
  ```

- Arbitrary precision for `LONGLONG` (gmp).
- Procedure overloading.
- Explicit memory deallocation.

Evolving the language

- Separation between declaration and body.
- Arbitrary precision for `LONGLONG` (gmp).
- Procedure overloading.
- Separation between declaration and body.
- Subtypes.
- Classes... or tagged types?
- Improvements to control structures: DOWNTO, UNTIL.
- Removing stropping?

In the meanwhile... Algol 68 Genie!

- Complete and modern interpreter by Marcel van der Veer.
- GPL.
- Written in C.
- Implements some interesting extensions.
- http://www.xs4all.nl/ jmvdveer/algol.html