Can Programming Be Liberated from the von Neumann Style? (1/3)

Yousun Ko

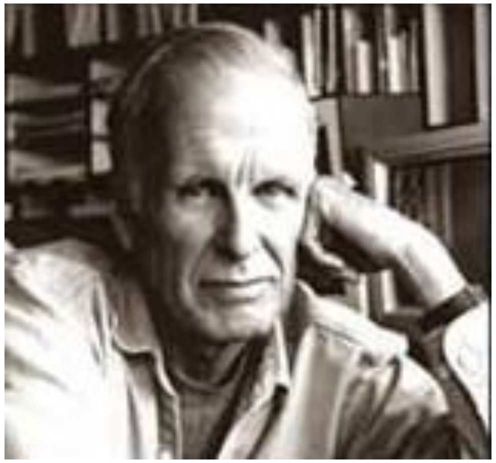
Contents

About the paper and author

- This paper:
  - was Turing Award lecture on 1978 by J. Backus
  - Suggests new concept of programming language, the Function-level programming language
  - has 16 sections
    - Sec. 1~10: Prospect current status regarding von Neumann architecture and languages
    - Sec. 11: Description of Functional Programming System
    - Sec. 12: Algebra of Programs for FP Systems
    - Sec. 13: Formal Systems for Functional Programming
    - Sec. 14: Applicative State Transition System
    - Sec. 15~16: Remarks and summaries

About the paper and author

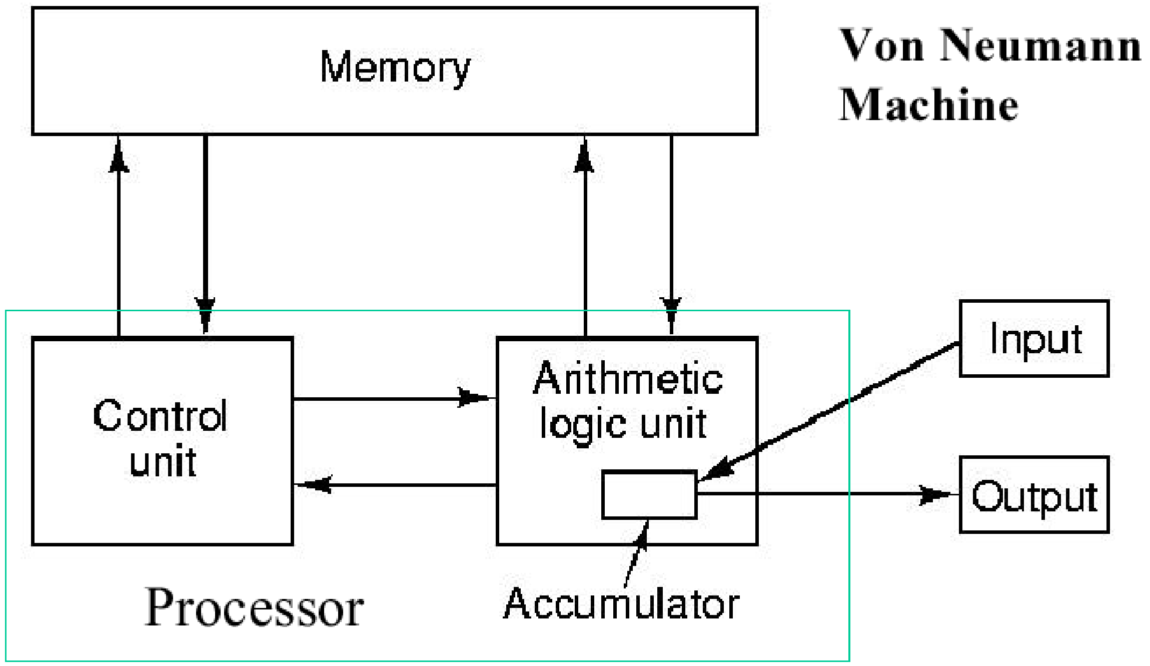
- The Author
  - John Backus



- Developer of the first high level and general-purpose programming language, Fortran
- Developer of Backus-Naur Form(BNF) which was critical contribution in the development of compiler.
- Developer of FL (Function Level)

Prerequisite Knowledge: Von Neumann Architecture

- Implements a universal Turing machine and has a sequential architecture.
- Stored-program computer



Section 1. Conventional Programming Languages

- In short, they are fat and flabby
  - Few languages make programming sufficiently cheaper or more reliable to justify the cost of producing and learning to use them
  - Large increases in size bring only small increases in power
  - Conventional languages create unnecessary confusion in the way we think about programs
- Scott-Strachey approach to denotational semantics
  - Examines the appalling type structure of conventional languages

Section 2.
Models of Computing System

- Every programming language is a model of a computing system that its programs control
- Criteria for Models
  - 1) Foundations
    - How much concise mathematical description is used to the model
  - 2) History sensitivity
    - Does the model include a notion of storage, so that one program can save information that can affect the behavior of a later program?
  - 3) Type of semantics
    - State-transition semantics, reduction semantics, etc.
  - 4) Clarity and conceptual usefulness of programs
    - Are programs of the model clear expressions of a process or computation?
    - Do they embody concepts that help us to formulate and reason about process?

Section 2.
Models of Computing System

- Classification of Models
  - 1) Simple operational models
    - Turing machines, various automata.
    - *Foundations:* concise and useful
    - *History sensitivity:* have storage
    - *Semantics:* state transition with very simple states
    - *Program clarity:* programs unclear and conceptually not helpful

Section 2.
Models of Computing System

- Classification of Models
  - 2) Applicative models
    - Church's lambda calculus, Curry's system of combinators, pure Lisp, functional programming systems
    - *Foundations:* concise and useful
    - *History sensitivity:* no storage
    - *Semantics:* no state
    - *Program clarity:* programs can be clear and conceptually useful

Section 2.
Models of Computing System

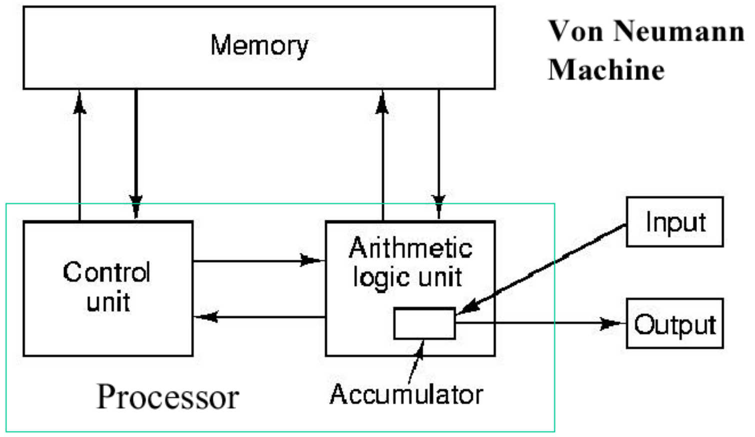
- Classification of Models
  - 3) Von Neumann models
    - Von Neumann computers, conventional programming languages
    - *Foundations:* complex, bulky, not useful
    - *History sensitivity:* have storage
    - *Semantics:* state transition with complex states
    - *Program clarity:* programs can be moderately clear, are not very useful conceptually

Section 3.
Von Neumann Computers

- Intellectual parent of conventional programming languages
- It was an elegant, practical, and unifying idea that simplified an number of engineering and programming problems that existed *60 years ago*

- The word-at-a-time tube is built-in bottleneck
  - Most of tube traffic is due to names of data, operations etc.

- It is a intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand.



Section 4.
Von Neumann Languages

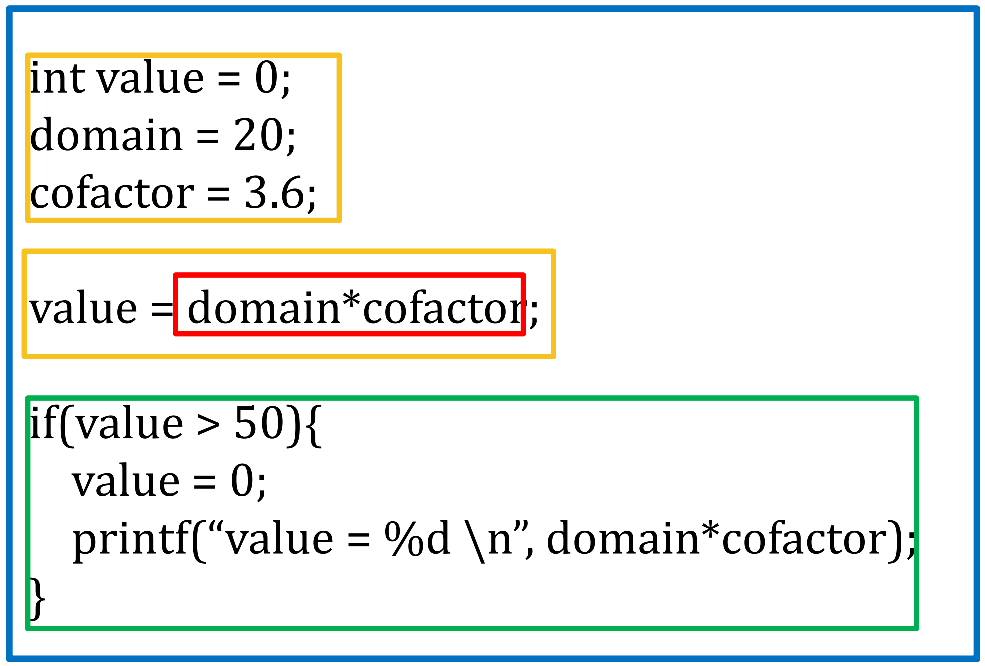
- Basically high level, complex versions of the von Neumann computer

- Use variables to imitate the computer’s storage cells
  - control statements == jump and test instructions
  - assignment statements == fetching, storing, and arithmetic
    - Symbol := is linguistic von Neumann bottleneck!

Section 4.
Von Neumann Languages

- Assignment splits programming into two worlds
  - Orderly world
    - Right sides of assignment statements
    - Has useful algebraic properties (except that those properties are often destroyed by side effects)
    - Most useful computation takes place
  - Disorderly world
    - World of statements
    - All statements of the language exist in order to make assignment statement possible to perform a computation that must be based on this primitive construct
    - Primitive use of loops, subscripts, and branching flow of control → chaotic world

Prerequisite Knowledge: Expressions, Statements, and Blocks



```
int value = 0;
domain = 20;
cofactor = 3.6;

value = domain*cofactor;

if(value > 50){
    value = 0;
    printf(“value = %d \n”, domain*cofactor);
}
```

Section 5. Comparison of von Neumann and Functional Programs

- A von Neumann Program for Inner Products

```
c := 0
for i := 1 step 1 until n do
   c:= c + a[i] * b[i]
```

- Several properties of this program are worth nothing
  - a) Its statements operate on an invisible "state" according to complex rules
  - b) It is not hierarchical
  - c) It is dynamic and repetitive
  - d) It computes word-at-a-time by repetition and by modification
  - e) Part of the data is in the program
  - f) It names its arguments (it requires a procedure declaration)
  - g) Its "housekeeping" operations are represented by symbols in scattered places

Section 5. Comparison of von Neumann and Functional Programs

- A Functional Program for Inner Products

```
Def Innerproduct
        ≡ (Insert +)∘(ApplyToAll *)∘Transpose
```

  - a) It operates only on it's arguments. No hidden states.
  - b) It is hierarchical
  - c) It is static and non-repetitive
  - d) It operates on whole conceptual units, not words
  - e) It incorporates no data. It works for any pair of conformable vectors.
  - f) It does not name its arguments
  - g) It employs "housekeeping" forms and functions that are generally useful in many other programs.

Section 6.
Language Frameworks versus Changeable Parts

- Frameworks
  - Gives overall rules of the system
    - e.g., **for** statement
- Changeable Parts
  - Its existence is anticipated by the framework but whose particular behavior is not specified by it
    - e.g., library functions, user defined procedures
- von Neumann languages always seems to have an immense framework and very limited changeable parts
  - von Neumann language must have a semantics closely coupled to the state
  - Every feature must be built into the state and its transition rules

Section 7.
Changeable Parts and Combining Forms

- Changeable parts in von Neumann languages have so little expressive power
  - “If the designer knew that all those complicated features, which he now builds into the framework, could be added later on as changeable parts, he would not be so eager to build them into the framework”

- Obstacles to the use of combining forms
  - Split between the expression world and the statement world
  - Use of elaborate naming conventions, which are further complicated by the substitution rules required in calling procedures

Next part will cover about:

Functional Programming Systems