

# Introduction to Program Synthesis (SS 25)

## Chapter 1 - Introduction

Dr. rer. nat. Roman Kalkreuth

Chair for AI Methodology (**AIM**), Department of Computer Science,  
RWTH Aachen University, Germany



Center for  
Artificial Intelligence

**RWTH**AACHEN  
UNIVERSITY

# An early approach to PS

## The Fortran I compiler

- ▶ In 1954 one of the first compiler for Fortran (*Formula Translation*) was introduced [Bac+57]
  - ~ First commercial release in 1957
- ▶ Fortran is one of the oldest programming languages
  - ~ Specifically designed for scientific numerical computing
- ▶ Translation from Fortran programs to assembly code for the IBM 704 mainframe

## The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBTT,  
H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†,  
H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT||

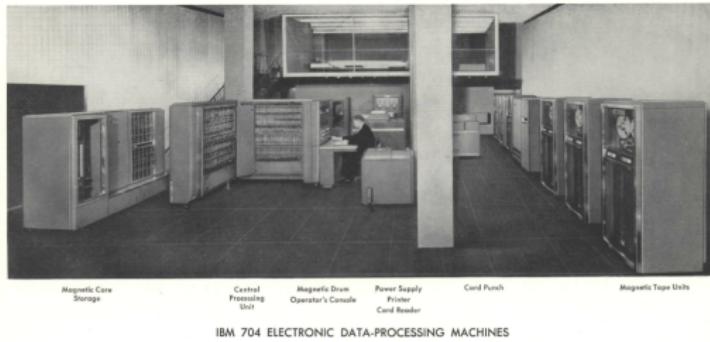
## Fortran: Programming paradigm

- ▶ Fortran → imperative programming language
  - ~~ Imperative paradigm → based on the Turing-machine model
  - ~~ Support for functional programming
- ▶ Use of statements that change the state of a program
- ▶ Fundamental concepts are
  - ~~ Store
  - ~~ Variable
  - ~~ Assignment
  - ~~ Iteration

# An early approach to PS

## The Fortran I compiler

- ▶ IBM 704 mainframe → first mass-produced computer capable of handling floating-point arithmetic
  - ↝ 40,000 instructions per second → up to 12,000 floating-point additions per second
  - ↝ Instruction set size: 102 instructions
- ▶ Magnetic Core Storage Unit →  $4,096 \times 36\text{-bit words}$  (18,432 bytes) of RAM
- ▶ FORTRAN and LISP were the first to be developed for the IBM 704



# An early approach to PS

The Fortran I compiler

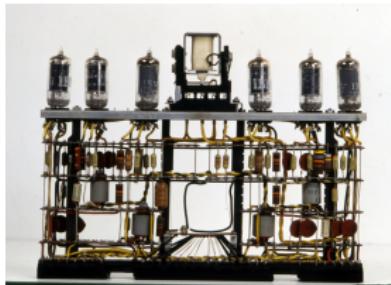
- ▶ Data registers: **Accumulator** (38 bit), **multiplier/quotient register** (36-bit)
- ▶ Three 15-bit **index registers**
- ▶ 15-bit **program counter**



# An early approach to PS

The Fortran I compiler

- ▶ Logic circuitry design → vacuum-tube circuit modules
- ▶ 4000 logic modules ran on +150 Volt and -150 Volt power supply
- ▶ **Von Neumann architecture** → Separation of processing, control and memory unit



## An early approach to PS

The Fortran I compiler

- ▶ The Bernstein Chess Program<sup>12</sup> → first complete chess engine
- ▶ Developed on and designed for the IBM 704 mainframe
- ▶ Capable of defeating an inexperienced human opponent

---

<sup>1</sup> [https://www.chessprogramming.org/The\\_Bernstein\\_Chess\\_Program](https://www.chessprogramming.org/The_Bernstein_Chess_Program)

<sup>2</sup> <https://lichess.org/@/bernstein-2ply>

## An early approach to PS

### The Fortran I compiler

- ▶ Demonstration that it is possible to automatically generate efficient machine code from a high-level language
    - ~~> First known successful attempt in code optimization
  - ▶ Considered as one of the 10 algorithms with the greatest influence in science and engineering in the 20th century
    - ~~> Computing in Science & Engineering Journal (2000) [DS00]
- 
- Metropolis Algorithm for Monte Carlo
  - Simplex Method for Linear Programming
  - Krylov Subspace Iteration Methods
  - The Decompositional Approach to Matrix Computations
  - The Fortran Optimizing Compiler
  - QR Algorithm for Computing Eigenvalues
  - Quicksort Algorithm for Sorting
  - Fast Fourier Transform
  - Integer Relation Detection
  - Fast Multipole Method

# Nostalgic Programming: Fortran

## Historical Sketch (Fortran Standards)

- 1957 • John Backus at IBM is developing a language for *Formula translation* → **Fortran 1**
- 1966 • First ANSI<sup>3</sup> standard for a programming language ever → **Fortran 66**
- 1978 • ANSI standard X3.9-1978 defines **Fortran 77**
- 1992 • ANSI standard X3.198-1992 defines **Fortran 90**
- 1997 • ISO/IEC 1539-1:1997 revises (and extends) Fortran 90 to **Fortran 95**
- 2004 • ISO/IEC 1539-1:2004 introduces **Fortran 2003**
- 2010 • **Fortran 2008** (ISO/IEC 1539-1:2010)
- 2018 • **Fortran 2018** (ISO/IEC TS 29113:2012 & ISO/IEC TS 18508:2015)
- 2023 • **Fortran 2023** (ISO/IEC 1539-1:2023)

<sup>3</sup> American National Standards Institute (<https://www.ansi.org/>)

# Nostalgic Programming: Fortran

## Historical Sketch (Fortran Features)

- 1957 • I/O, DO loops, GOTO, IF statements
- 1958 • Functions
- 1966 • Booleans, portability
- 1977 • Block if/else, strings, I/O revision
- 1990 • Free-form input, parallelism (SIMD), recursion, memory allocation, modules
- 1995 • SIMD parallelism revision
- 2003 • Object-oriented programming, function pointers
- 2008 • Further revision and extension (SIMD and MIMD)
- 2018 • Additional features for parallelism
- 2023 • Correcting errors and omissions from Fortran 18

# Nostalgic Programming: Fortran

## Fun Facts

- ▶ Fortran is older than Professor Hoos
- ▶ Fortran ranks #9 in the  $\pi$  approximation speed comparison benchmark<sup>4</sup>
- ▶ Fortran II compiler for the PDP-8 → 3700 12-bit words of memory
  - ~~ PDP-8 had  $4096 \times 12$ -bit words of memory (6 Kilobytes)
  - ~~ GCC (GNU Compiler Collection) had over 14 million lines of code in 2015

---

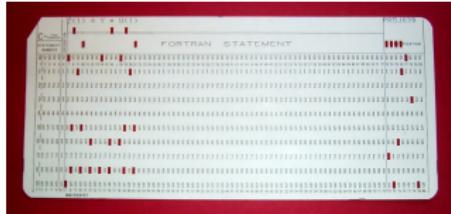
<sup>4</sup> <https://github.com/niklas-heer/speed-comparison>

## Fortran: Real Talk

- ▶ Fortran has survived **70 years** in the computer science era and is still used today
- ▶ Highly powerful compiler design
- ▶ Landmark in **code optimization** and **program synthesis**

# Fortran: Program representation and storage

- ▶ Throughout the 70s, many Fortran programs were stored or directly written on punch cards
- ▶ One card was used to store each line (statement) of Fortran program
- ▶ A stack of cards was necessary to store an entire program



## Fortran: Code Examples

```
! Fortran 90 implementation of a monte-carlo algorithm
! to approximate pi
program main
    implicit none
    integer :: i, samples, count, report
    real(8) :: x, y, d, estimate

    samples = 10000
    count = 0

    do i = 1, nsamples
        call random_number(x)
        call random_number(y)

        r = sqrt(x**2 + y**2)
        if(r < 1d0) count = count + 1

        if(mod(i, report) == 0) then
            estimate = 4d0 * count/dble(i)
            write(*,*) estimate
    end do
```

**Listing:** Implementation of a monte-carlo algorithm that approximates  $\pi$  in FORTRAN

## Fortran: Code Examples

```
! Fortran 90 implementation of the calculation of the euler number
program euler
    implicit none
    integer, parameter :: n = 10
    integer :: s, f
    do i = 1, n
        f = call fact(n)
        s = s + (1 / f)
    end do
end program

integer function fact(n)
    implicit none
    integer :: n, f

    do i = 1, n
        f = f * i
    end do

    fact = f
end function fact
```

Listing: Implementation of the calculation of the euler number in FORTRAN

## An early approach to PS

The Fortran I compiler

- ▶ **23,500** assembly language instructions
- ▶ Development required **18 person-years**
- ▶ Was the first major project in code optimization capable of:
  - ~ Reduction of **runtime**
  - ~ Reduction of **memory occupation**
  - ~ Reduction of **power consumption**

## An early approach to PS

### The Fortran I compiler

- ▶ With the Fortran I compiler various features have been introduced that are used by modern compilers<sup>5</sup>
  - ~ Expression parsing with **redundancy optimization, copy propagation** followed by **dead code elimination**
  - ~ Identification of **permutations of operations** → memory access reduction
  - ~ Elimination of **redundant computations** that resulted from common sub-expressions

---

<sup>5</sup> <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

# An early approach to PS

## The Fortran I compiler

- ▶ Parse and compile arithmetic expressions → major challenge
  - ~ Lack of operator priority → missing precedence or hierarchy
- ▶ The Fortran I compiler expands every operator with parentheses
  - ~ The expression **A + B \* C** would be expanded to  
 $((A)) + ((B) * (C))$
  - ~ Translation scans the resulting expression from left to right
  - ~ Expression according to the operator hierarchical semantics
  - ~ The optimisation then eliminates redundancy

**Listing:** Code after parsing

```
u1=u2+u4; u2=u3; u3=A;  
u4=u5*u6; u5=B; u6=C
```

**Listing:** Code after optimization

```
u1=A+u4; u4=B*C
```

# An early approach to PS

The Fortran I compiler

- a **Dead variable elimination** → removal of statements assigned to values to unused variables
- b **Elimination of common sub-expressions** → search for expressions that have been performed previously
- c **Constant propagation** → removal of calculations that only contain known constants that can be performed directly in the compiler
- d **Code motion** → rearrangement of expressions

**Listing:** Sample program

```
A = B + C
Y = Y + i.
Z = C
Q = (Z + B) * SIN (.7854)
DO I=1,100
P(I) = P(I) * (A + B)
```

**Listing:** Optimised program

```
A = B + C
!
!
Q = A * 0.7071
T = A + B
DO I=1,100
P(I) = P(I) * T
```

← a  
← a  
← b,c  
← d

# An early approach to PS

The Fortran I compiler

- ▶ Collection of information about the **structure of programs**
  - ~ Program was broken down into a set of basic blocks
  - ~ Monte Carlo simulation → finding the **execution frequency** of each block
  - ~ **Frequency statements** → traversing paths and **counting frequencies** of the program flow graph

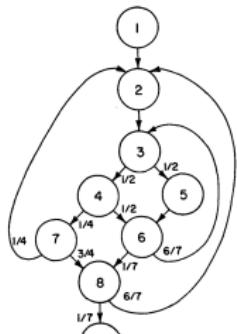


Figure 4. Program Flow Graph with Branch Probabilities

Table 1. Relative Frequencies Obtained by Simulation

BLOCK	RELATIVE FREQUENCY
1	1.0
2	8.2
3	33.0
4	16.5
5	16.5
6	28.9
7	4.1
8	7.1
9	1.0

Figure: Frequency analysis of programs with the Fortran I compiler [Sch73]

# Paper Discussion

## Takeaways

- ▶ Landmarks for program synthesis:
  - ▶ **Decomposition** of programs into **building blocks**
    - ▶ Foundation for assembling new programs in the synthesis process
  - ▶ **Optimisation** of programs with **monte-carlo methods**
    - ▶ Proof-of-concept that was later extended by genetic programming

## References

- [Bac+57] J. W. Backus et al. "The FORTRAN automatic coding system". In: *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*. IRE-AIEE-ACM '57 (Western). Los Angeles, California: Association for Computing Machinery, 1957, pp. 188–198. ISBN: 9781450378611. DOI: [10.1145/1455567.1455599](https://doi.org/10.1145/1455567.1455599). URL: <https://doi.org/10.1145/1455567.1455599>.
- [DS00] Jack Dongarra and Francis Sullivan. "Guest Editors' Introduction: The Top 10 Algorithms". In: *Computing in Science and Engg.* 2.1 (Jan. 2000), pp. 22–23. ISSN: 1521-9615. DOI: [10.1109/MCISE.2000.814652](https://doi.org/10.1109/MCISE.2000.814652). URL: <https://doi.org/10.1109/MCISE.2000.814652>.
- [Sch73] Paul B. Schneck. "A survey of compiler optimization techniques". In: *Proceedings of the ACM Annual Conference*. ACM '73. Atlanta, Georgia, USA: Association for Computing Machinery, 1973, pp. 106–113. ISBN: 9781450374903. DOI: [10.1145/800192.805690](https://doi.org/10.1145/800192.805690). URL: <https://doi.org/10.1145/800192.805690>.