A history of compilers

Peter Sestoft
sestoft@itu.dk

Dansk Datahistorisk Forening
2014-01-23 v 1.0
The speaker

- MSc 1988 computer science and mathematics and PhD 1991, DIKU, Copenhagen University
- KU, DTU, KVL and ITU; and AT&T Bell Labs, Microsoft Research UK, Harvard University
- Programming languages, software development, ...
- Open source software
  - Moscow ML implementation, 1994...
  - C5 Generic Collection Library, with Niels Kokholm, 2006...
Outline

• What is a compiler?
• A history of the history of ...
• Early autoprogramming systems
• The FORTRAN I compiler
• Some Algol compilers
• Lexing and parsing
• Code generation
• Intermediate languages
• Optimization
• Flow analysis
• Type systems
What is a compiler (today)?

C language source program

```
for (int i=0; i<n; i++)
    sum += sqrt(arr[i]);
```

clang

x86 machine code

```
LBB0_1:
    movl    -28(%rbp), %eax         // i
    movl    -4(%rbp), %ecx          // n
    cmpl    %ecx, %eax
    jge     LBB0_4                  // if i >= n, return
    movslq  -28(%rbp), %rax         // i
    movq    -16(%rbp), %rcx         // address of arr[0]
    movsd   (%rcx,%rax,8), %xmm0    // arr[i]
    callq   _sqrt                   // sqrt
    movsd   -24(%rbp), %xmm1        // sum
    addsd   %xmm0, %xmm1            // sum + ...
    movsd   %xmm1, -24(%rbp)         // sum = ...
    movl    -28(%rbp), %eax         // i
    addl    $1, %eax                // i + 1
    movl    %eax, -28(%rbp)         // i = ...
    jmp     LBB0_1                  // loop again
```

From Aho et al
A compiler in context

Source language
- Syntax
- Types, semantics
- Libraries

Intermediate lang.
- Instructions
- Types, semantics
- Libraries

Compiler
- Lexing, parsing
- Checking, types
- Optimization
- Code generation

Dev. environment
- Editors
- Syntax highlighting
- Debuggers
- Linkers

Runtime support
- Memory management
- Libraries
- Debuggers

Hardware
- Instruction set
- Memory hierarchy
- Peripherals

Operating system
- Process management
- Kernel functions

Focus
Conceptual phases of a compiler

From Aho et al
Why "compiler" not "translator"?

• Hopper: A Programmer's Glossary, 1 May 1954:

  **Compile (verb)** - The process of producing from pseudo-code a specific routine for a particular problem by:
  1) decoding elements of information expressed in pseudo-code and segmenting the problem;
  2) selecting or generating the required sub-routines;
  3) transforming the subroutines into specific coding and entering them as elements in the problem routine;
  4) maintaining a record of the subroutines used and their position in the problem routine.

  Compilation decodes the pseudo-code and processes static and dynamic subroutines and generators to produce a specific routine for a problem before computation.

• Hopper's A-2 compiler collected & inlined subroutines
  - In modern terminology: *macro expansion*

• Later use: "algebraic compiler" to mean *translator*
A history of the history of ...

• Bauer: *Historical remarks on compiler construction* (1974)
  – Many important references to early papers
  – USSR addendum by Ershov in 2nd printing (1976)
  – Opening quote:

D. E. KNUTH [81] has observed (in 1962!) that the early history of compiler construction is difficult to assess. Maybe this, or maybe the general unhistorical attitude of our century is responsible for the widespread ignorance about the origins of compiler construction. In addition, the overwhelming lead of the USA in the general de-
Some older histories of ...

- Knuth: *A history of writing compilers* (1962)
  - Few references, names and dates, mostly US:
    A complete bibliography of the compiler literature is hard to give; you may, in fact, find it quite distressing to try to read many of the articles.
  - Knuth later regretted this attitude


- Randell and Russell 1964, par. 1.2 and 1.3

- Rosen: *Programming Systems and Languages*, 1964 and 1972
### Knuth 1977: The early development ...

<table>
<thead>
<tr>
<th>Language</th>
<th>Principal author(s)</th>
<th>Year</th>
<th>Arithmetic</th>
<th>Implementation</th>
<th>Readability</th>
<th>Control structures</th>
<th>Data structures</th>
<th>Machine independence</th>
<th>Impact</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plankalkül</td>
<td>Zuse</td>
<td>1945</td>
<td>X,S,F</td>
<td>F</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>Programming language, hierarchic data</td>
</tr>
<tr>
<td>Flow diagrams</td>
<td>Goldstine &amp; von Neumann</td>
<td>1946</td>
<td>X,S</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>Accepted programming methodology</td>
</tr>
<tr>
<td>Composition</td>
<td>Curry</td>
<td>1948</td>
<td>X</td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td>Code generation algorithm</td>
</tr>
<tr>
<td>Short Code</td>
<td>Mauchly</td>
<td>1950</td>
<td>F</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>High-level language implemented</td>
</tr>
<tr>
<td>Intermediate PL</td>
<td>Burks</td>
<td>1950</td>
<td>?</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>F</td>
<td>Common subexpression notation</td>
</tr>
<tr>
<td>Klammer-ausdrücke</td>
<td>Rutishauser</td>
<td>1951</td>
<td>F</td>
<td>F</td>
<td>B</td>
<td>F</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>Simple code generation, loop expansion</td>
</tr>
<tr>
<td>Formules</td>
<td>Böhm</td>
<td>1951</td>
<td>X</td>
<td>F</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td>Compiler in own language</td>
</tr>
<tr>
<td>AUTOCODE</td>
<td>Glennie</td>
<td>1952</td>
<td>X</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>Useful compiler</td>
</tr>
<tr>
<td>A-2</td>
<td>Hopper</td>
<td>1953</td>
<td>F</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>F</td>
<td>C</td>
<td>B</td>
<td>Macroexpander</td>
</tr>
<tr>
<td>Whirlwind translator</td>
<td>Laning &amp; Zierler</td>
<td>1953</td>
<td>F</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>Constants in formulas, manual for novices</td>
</tr>
<tr>
<td>AUTOCODE</td>
<td>Brooker</td>
<td>1954</td>
<td>X,F</td>
<td>A</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>Clean two-level storage</td>
</tr>
<tr>
<td>IIII-2</td>
<td>Kamynin &amp; Lübimskii</td>
<td>1954</td>
<td>F</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>B</td>
<td>B</td>
<td>D</td>
<td>Code optimization</td>
</tr>
<tr>
<td>III</td>
<td>Ershov</td>
<td>1955</td>
<td>F</td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>Book about a compiler</td>
</tr>
<tr>
<td>BACAIC</td>
<td>Grems &amp; Porter</td>
<td>1955</td>
<td>F</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td>Use on two machines</td>
</tr>
<tr>
<td>Kompiler 2</td>
<td>Elsworthy &amp; Kuhn</td>
<td>1955</td>
<td>S</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td>Scaling aids</td>
</tr>
<tr>
<td>PACT I</td>
<td>Working Committee</td>
<td>1955</td>
<td>X,S</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>Cooperative effort</td>
</tr>
<tr>
<td>ADES</td>
<td>Blum</td>
<td>1956</td>
<td>X,F</td>
<td>D</td>
<td>D</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>F</td>
<td>Declarative language</td>
</tr>
<tr>
<td>IT</td>
<td>Perlis</td>
<td>1956</td>
<td>X,F</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>Successful compiler</td>
</tr>
<tr>
<td>FORTRAN I</td>
<td>Backus</td>
<td>1956</td>
<td>X,F</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>I/O formats, comments, global optimization</td>
</tr>
<tr>
<td>MATH-MATIC</td>
<td>Katz</td>
<td>1956</td>
<td>F</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>Heavy use of English</td>
</tr>
<tr>
<td>Patent 3,047,228</td>
<td>Bauer &amp; Samelson</td>
<td>1957</td>
<td>F</td>
<td>D</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>Formula-controlled computer</td>
</tr>
</tbody>
</table>

X=int, F=float, S=scaled  
A ... F = much ... little
Other substantial secondary sources

- Naur: *The replies to the AB14 questionnaire* – Survey of Algol compiler construction June 1962

- Bromberg: *Survey of programming languages and processors* (March 1963)
### Early interpreters and compilers

<table>
<thead>
<tr>
<th>Language</th>
<th>Machine</th>
<th>Operatic</th>
<th>Developer</th>
<th>Comp. size</th>
<th>Comp spec</th>
<th>Citation 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDSAC</td>
<td>IBM 701</td>
<td>Sep-1950</td>
<td>Wilkes, Wheeler, Gill</td>
<td></td>
<td></td>
<td>Wilkes:1951:ThePreparation</td>
</tr>
<tr>
<td>Speedcode</td>
<td>IBM 701</td>
<td>Sep-1953</td>
<td>Backus</td>
<td></td>
<td></td>
<td>Backus:1954:TheIbm</td>
</tr>
<tr>
<td>A-2</td>
<td>Univac I</td>
<td>Nov-1953</td>
<td>Hopper</td>
<td></td>
<td></td>
<td>Knuth:1977:TheEarly</td>
</tr>
<tr>
<td>Autocode</td>
<td>Mark I</td>
<td>Dec-1955</td>
<td>Brooker</td>
<td></td>
<td></td>
<td>Knuth:1977:TheEarly</td>
</tr>
<tr>
<td>Fortran I</td>
<td>IBM 704</td>
<td>Jun-1957</td>
<td>Backus et al</td>
<td>24000 ins</td>
<td>8 cards/min</td>
<td>Backus:1957:TheFortran</td>
</tr>
<tr>
<td>MAC</td>
<td>Ferranti Mercu</td>
<td>Dec-1957</td>
<td>Dahl</td>
<td></td>
<td></td>
<td>AMS:1958:MathematicalTable</td>
</tr>
<tr>
<td>Runcible</td>
<td>IBM 650</td>
<td>Dec-1958</td>
<td>Knuth</td>
<td></td>
<td></td>
<td>Knuth:1959:Runcible</td>
</tr>
<tr>
<td>Algol 58</td>
<td>Zuse Z22</td>
<td>Dec-1958</td>
<td>Bauer, Samelson</td>
<td></td>
<td></td>
<td>Samelson:1960:SequentialFo</td>
</tr>
<tr>
<td>Nelliac</td>
<td>Univac M-460</td>
<td>Mar-1959</td>
<td>Halstead</td>
<td></td>
<td></td>
<td>Huskey:1959:Nelliac</td>
</tr>
<tr>
<td>Algol 58</td>
<td>B 220</td>
<td>Dec-1959</td>
<td>Barton</td>
<td>3500 ins</td>
<td>500 instr/min</td>
<td>Barton:1961:AnotherNameles</td>
</tr>
<tr>
<td>Lisp 1</td>
<td>IBM 704</td>
<td>Jan-1960</td>
<td>McCarthy</td>
<td></td>
<td></td>
<td>McCarthy:1960:RecursiveFun</td>
</tr>
<tr>
<td>Algol 60</td>
<td>X-1</td>
<td>Jun-1960</td>
<td>Dijkstra</td>
<td>2500 words</td>
<td></td>
<td>Dijkstra:1960:RecursiveProgr</td>
</tr>
<tr>
<td>Algol 58</td>
<td>B 205</td>
<td>Sep-1960</td>
<td>Knuth</td>
<td>4000 words</td>
<td>45 cards/min</td>
<td>Knuth:1960:TheInternals</td>
</tr>
<tr>
<td>Algol 60</td>
<td>CDC 1604</td>
<td>Jun-1961</td>
<td>Irons</td>
<td>800 ins/10000 tbl</td>
<td>300 instr/s</td>
<td>Irons:1961:ASyntax</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Facit EDB</td>
<td>Oct-1961</td>
<td>Dahlstrand</td>
<td></td>
<td></td>
<td>Dahlstrand:2009:Minnen</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Gier</td>
<td>Sep-1962</td>
<td>Jensen, Naur</td>
<td>4000 words</td>
<td>30 instr/s</td>
<td>Naur:1963:TheDesign1</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Whet</td>
<td>Sep-1962</td>
<td>Randell, Russe</td>
<td>3000 words</td>
<td>input limited</td>
<td>Randell:1964:Algol60Implement</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Kidsig</td>
<td>Dec-1962</td>
<td>Hawkins, Huxt</td>
<td>20000 words</td>
<td></td>
<td>Randell:1964:Algol60Implement</td>
</tr>
<tr>
<td>Algol 60</td>
<td>M-20</td>
<td>Jan-1964</td>
<td>Ershov</td>
<td>45000 words</td>
<td></td>
<td>Ershov:1966:Alpha</td>
</tr>
</tbody>
</table>
Knuth's B 205 Algol 58 compiler

• Developed June-Sep 1960, at age 22
• Idiosyncratic documentation:

How does this compiler work? Frankly, it's a miracle if it does.

The style of presentation is adapted from the practice of computer publications in the U.S.S.R.: the flowchart boxes call scanners (routine 0) which operate from A1 and A2. The Rube Goldberg procedure call scanner actually begins by operation

......and that's all there is to this compiler.

• No description of runtime state or its invariants
Problems and solutions

• Lexing and parsing: from text to internal representation
  – Arithmetic operators, precedence, parentheses

• Compilation of expressions
  – To postfix, reverse Polish form

• Storage allocation
  – Runtime state invariants, code to maintain them

• Optimization
  – How to generate efficient code

• Flow analysis
  – How discover program structure and invariants
History: lexing and parsing

- Initially ad hoc
- Table-driven/automata methods
- Regular expressions, context-free grammars
- Finite state automata and pushdown automata
- Knuth LR parsing 1965
- Gries operator grammars 1968
- Lexer and parser generator tools
  - Lex (Lesk 1975) and Yacc (Johnson 1975)
  - LR dominated for a while
  - LL back in fashion: Antlr, Coco/R, parser combinators, packrat parsers
• Historically, too much emphasis on parsing?
  – Because it was formalizable and respectable?
  – But also beautiful relations to complexity and computability …
History: compilation of expressions

- **Rutishauser 1952** (unimpl.)
  - Translating arithmetic expressions to 3-addr code
  - Infix operators, precedence, parentheses
  - Repeated scanning and simplification

- **Böhm 1952** (not impl.)
  - Similar – also at ETH Zürich

- **Fortran I, 1957**
  - Baroque but simple treatment of precedence (Böhm &)
  - Complex, multiple scans, both left-right and right-left

- **Samelson and Bauer 1960**
  - One scan, using a stack ("cellar") at translation time

- **Floyd 1961**
  - One left scan, one right scan, optimized code
History: Compilation techniques

- Single-pass table-driven with stacks
  - Bauer and Samelson for Alcor
  - Dijkstra 1960, Algol for X-1
  - Randell 1962, Whetstone Algol

- Single-pass recursive descent
  - Hoare 1962, one procedure per language construct

- Multi-pass ad hoc
  - Fortran I, 6 passes

- Multi-pass table-driven with stacks
  - Naur 1962 GIER Algol, 9 passes
  - Hawkins 1962 Kidsgrove Algol

- General syntax-directed table-driven
  - Irons 1961 Algol for CDC 1604
Multi-pass compilation is still used

- Good for small-memory systems (eg GIER)
- Still used, now for separation of concerns:

21 internal passes of the Scala compiler (2013)

namerFactory
typerFactory
superAccessors
pickler
refchecks
liftcode
uncurry
tailCalls
explicitOuter
erasure
lambdaLift
constructors
flatten
mixer
cleanup
genicode
inliner
inlineExceptionHandlers
closureElimination
deadCode
genJVM
History: Run-time organization

• Early papers focus on translation
  – Runtime data management is trivial, e.g. Fortran I
• Algol: runtime storage allocation is essential
• Dijkstra: Algol for X-1 (1960)
  – Stack of procedure activation records
  – Display, to access variables of enclosing scopes
• Also focus of Naur's Gier Algol papers

• Design a runtime state structure (invariant)
• Compiler should generate code that
  – Can rely on the runtime state invariant
  – Must preserve the runtime state invariant
History: Target architectures

• EDSAC, IAS machine, BESK
  – No index register, so self-modifying code for arrays
  – Subtle, and makes manual code relocation painful

• Index registers, IBM 704, DASK, ...
  – Simpler and faster code, position-independent
  – IBM 704 at-least-once loops impacts Fortran DO

• Stack machines
  – Conceptual: Samelson, Hamblin, Dijkstra, Barton
  – Hardware: Burroughs B5000, KDF9 (arith, return)
  – Compile to reverse Polish by post-order traversal
  – Java and .NET/CLI intermediate languages

Hamblin:1962:TranslationTo
History: Intermediate languages

- Strong: *The problem of ...*, February 1958
Everything old is new again


Chow:2013:IntermediateRepresentation
**UNCOL is finally coming true**

- **Java Virtual Machine, 1994**
  - Developed as bytecode for Java only
  - Yet used for Scala, Clojure, Jython, Ceylon, JRuby...
  - Runtime system, libraries, on many CPUs and OSs

- **.NET Common Language Infrastructure, 1999**
  - C#, VB.NET, F#, JScript, Eiffel, COBOL, IronPython

- **JavaScript/EcmaScript, 1994**
  - For browser scripting, thus on all user devices
  - Ceylon, Dart, Typescript, F#, ...

- **LLVM = Low-Level Virtual Machine, 2003**
  - Static single assignment compiler-internal form
  - Clang C/C++/Objective-C, Nvidia CUDA, Mono, ...
What does a C# compiler do

C# language source program

```csharp
for (int i=0; i<n; i++)
    sum += sqrt(arr[i]);
```

`.NET/CLI bytecode:
• Stack-based
• Loadtime checks, types, local stack
• Runtime checks, array bounds, null references, ...
• Dynamic compilation to real machine code

.NET/CLI bytecode:
0b stloc.1 // i = 0
0c br.s 1d // goto ld
0e ldloc.0 // sum
0f ldarg.1 // arr
10 ldloc.1 // i
11 ldarg.r8 // arr[i]
12 call Math::Sqrt // sqrt
17 add // sum + ...
18 stloc.0 // sum
19 ldloc.1 // i
1a ldc.i4.1 // 1
1b add // i + 1
1c stloc.1 // i = ...
1d ldloc.1 // i
1e ldarg.0 // n
1f blt.s 0e // if i<n loop

x86 machine code
19 xorl %ebx,%ebx // i = 0
1b jmp 3a // goto 3a
1d leal 0x00(%ebp),%ebp // sum
20 fldl 0xec(%ebp) // array index check
23 cmpl %ebx,0x0c(%edi) // if outofbounds, throw
26 jbe 49 // ...
2c leal 0x10(%edi,%ebx,8),%eax // arr[i]
30 fldl (%eax) // sqrt
32 fsqrt // sqrt
34 faddp %st,%st(1) // sum + ...
36 fstpl 0xec(%ebp) // sum = ...
39 incl %ebx // i++
3a cmpl %esi,%ebx // if i<n, loop
3c jl 20
History: Type checking

• Naur GIER Algol 1965
  – "pseudoevaluation of the expressions"
  – "like a run-time evaluation but works with descriptions of the types and kinds of operands instead of with values"

• Damas and Milner, ML 1982
  – Inference of polymorphic type schemes
  – Generalization and specialization
  – Unification (Robinson) to solve type equations
  – Has influenced Haskell, C#, F#, Scala, ...
Compiler quality attributes

- Produces fast target code
- Produces target code fast
- Checks source code syntax
- Checks source code types and consistency
- Provides precise and clear error messages
- Reports as many errors as possible
- Is itself free of errors
- Does not report spurious errors
- Provides frugal compilation or recompilation or separate compilation
History: Diagnostics

• EDSAC (Wilkes 1951)
  – Tracing and post-mortem dumps

• GIER Algol
  – Comprehensive compiler error messages
  – Not a focus in eg Hoare's Elliot Algol compiler
  – "GIER ... had an excellent compiler-cum-operating system" (Sanders in HiNC 2003)
  – "the very successful Algol compiler ... for the GIER computer" (Randell & Russell 1964)

  – (Presumably also important that it was thoroughly tested, Naur:1963:TheDesign2 page 163)
History: Bootstrapping

- Writing a compiler in the language it compiles
- Runcible 1959
  - Runcible expression compilation chart, in Runcible

- Neliac compiler written in Neliac 1959
  - and self-compilation as correctness check
C compiler optimizations

- Register allocation (sum, i, n, arr)
- Inlining of functions
- Index calculations
- Iteration variable elimination (i)

```c
for (int i=0; i<n; i++)
    sum += sqrt(arr[i]);
```

```x86
LBB0_1:
    movl    -28(%rbp), %eax         // i
    movl    -4(%rbp), %ecx          // n
    cmpl    %ecx, %eax
    jge     LBB0_4                  // if i >= n, return
    movslq  -28(%rbp), %rax         // i
    movq    -16(%rbp), %rcx         // address of arr[0]
    movsd   (%rcx,%rax,8), %xmm0    // arr[i]
    callq   _sqrt                   // sqrt
    movsd   -24(%rbp), %xmm1        // sum
    addsd   %xmm0, %xmm1            // sum + ... 
    movsd   %xmm1, -24(%rbp)        // sum = ...
    movl    -28(%rbp), %eax         // i
    addl    $1, %eax                // i + 1
    movl    %eax, -28(%rbp)         // i = ...
    jmp     LBB0_1                  // loop again

LBB0_2:
    movsd   (%rsi), %xmm1           // *p same as arr[n-i]
    sqrtsd  %xmm1, %xmm1            // sqrt
    addsd   %xmm1, %xmm0            // sum += ...
    addq    $8, %rsi                 // p++
    decq    %rax                    // i--
    jne     LBB0_2                  // if i!=0 loop again
```
History: Optimization

• Fortran I in 1957 has
  – common subexpression elimination
  – fast index computations: reduction in strength
  – clever allocation of index registers
  – constant folding

• Samelson & Bauer
  – algorithm for fast index computations (red. stren.)

• Allen 1969, algorithms for:
  – basic blocks, flow graph of strongly conn. comps.
  – constant folding
  – common subexpression elimination
  – invariant code moving
  – reduction in strength
  – test replacement (for loops after red. strength)
History: Flow analysis

• Fortran I was amazingly ambitious
  – Control flow analysis graph with edge frequencies
  – Monte Carlo simulation at compiletime based on FREQUENCY statements

• Allen, Cocke 1970
  – Control flow graph, dominator, interval, reducible graph, ...
  – Left out of Bauer:1974:HistoricalRemarks

• Cousot and Cousot 1977
  – Abstract interpretation
  – Based on formal semantics
  – Systematic approximation via Galois connections
  – Termination via widening operators
Some topics not covered at all

• Compiler generation, compiler-compilers
• Adaptive compilation
  – just-in-time compilers for Java, .NET/CLI
• Compilation techniques for
  – lazy functional languages: Haskell
  – object-oriented languages: Java, C#
  – dynamic languages: Smalltalk, Javascript (v8)
  – extensible languages: Lua
• Compilation for parallel architectures
• Continuation-based compilation
• …
A thought on hardware and language developments 2010-2020

• First languages were made to resemble machines
• Then Algol made the opposite happen (stacks)
• Today: multicore, manycore, massive parallelism
  – Multiple levels memory and caches
  – Complex memory consistency models
  – Computation is cheap, moving data is expensive
  – Still too much focus on scientific computing? (exascale)
• Are future machines designed with enough input from programming language design?
• What are the language features suitable for parallelism and concurrency?
Interesting reading

• Secondary sources
  – Knuth:1977:TheEarly
  – Bauer:1974:HistoricalRemarks
  – Ershov:1976:Addendum

• Primary sources
  – Backus:1957:TheFortran
  – Samelson:1960:SequentialFormula
  – Dijkstra:1960:RecursiveProgramming
  – Naur:1963:TheDesign1
  – Naur:1965:CheckingOf
  – Randell:1964:Algol60Implementation
## A collection of sources

<table>
<thead>
<tr>
<th>3</th>
<th>Bibtex and link</th>
<th>Author</th>
<th>Title</th>
<th>Notes, message</th>
<th>Category</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Allen;1969:ProgramOptimi</td>
<td>Allen</td>
<td>Program optimization</td>
<td>static analysis</td>
<td>Annual review in automata theory</td>
<td>ACM SIGPLAN Notices 5, 3 (July 1970)</td>
</tr>
<tr>
<td>7</td>
<td>Allen;1976:CodesAndLink</td>
<td>Allen</td>
<td>Control Flow analysis</td>
<td>static analysis</td>
<td>ACM SIGPLAN Notices 5, 3 (July 1970)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ANS;1958:MathematicalTable</td>
<td>American Mathematical Society</td>
<td>Review of Dahl;1957:AutocodingFor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Arden;1969:On Gat</td>
<td>Arden, Graham</td>
<td>On Gat [Generalized Algebraic Translator] and the construction of translators</td>
<td>compilers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Backus;1954:Thietm</td>
<td>Backus</td>
<td>The IBM 701 Spec (Vcomputer)</td>
<td>Apparently no compilation, input is three-address interpreters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Backus;1957:TheFortran</td>
<td>Backus et al</td>
<td>The FORTRAN automatic coding system</td>
<td>About Fortran II, early 1957; only simple arithmetic fun languages</td>
<td>Western computer programe</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Backus;1959:AutomaticPro</td>
<td>Backus</td>
<td>Automatic programming: Prok About Fortran II; adds general procedures and function compilers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Barton;1961:ANew</td>
<td>Barton</td>
<td>A new approach to the function of Algol 60 inspired machine design with stack S and regi hardware</td>
<td>AFIPS Conference Proc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Barton;1961:AnotherName</td>
<td>Barton</td>
<td>Another (nameless) compiler for Some statistics on the Burroughs 220 Algol-58/60 com compilers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Bauer;1957:VerfahrenZur</td>
<td>Bauer, Sameson</td>
<td>Verfahren zur automatischen V Machine (hardware) patent on formula evaluation using</td>
<td>APL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Bauer;1959:SequentieleFol</td>
<td>Bauer, Sameson</td>
<td>Sequentielie formelubersetzung</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Bauer;1974:HistoricalRem</td>
<td>Bauer, Sameson</td>
<td>On Historical remarks on compiler Writing With bibliography (which misses Allen, Cocke)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Bemer;1969:PoliticoSocial</td>
<td>Bemer</td>
<td>A politico-social history of Algol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Bergin;2007:AHistory</td>
<td>Bergin</td>
<td>A history of the history of programming languages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Bateman;1961:AnAlternateRem</td>
<td>Bateman</td>
<td>An alternate form of the UNCOL First appearance of T diagrams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Bromberg;1963:SurveyOf</td>
<td>Bromberg</td>
<td>Survey of programming languages and processors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Bubenko;2005:HistoryOf</td>
<td>Bubenko, Impagliazzo</td>
<td>History of Nordic computer systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Carr;1959:AVisit</td>
<td>Carr et al</td>
<td>A visit to the computation centre: Mentions an automatic programming system in Moscow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Chow;2013:IntermediateRe</td>
<td>Chow</td>
<td>Intermediate representation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Conway;1958:ProposalFor</td>
<td>Conway</td>
<td>Proposal for an UNCOL</td>
<td>A concrete proposal for the UNCOL concept suggested</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Dahl;1959:AutoEncoding</td>
<td>Dahl</td>
<td>A plea for multiprogramming</td>
<td>concurrency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Dahl;1957:AutocodingFor</td>
<td>Dahl</td>
<td>Autocoding for the Ferranti Mercury Computer (MAC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Dahl;1966:Simula</td>
<td>Dahl, Nygaard</td>
<td>Simula, an Algol-based simulation language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Dahl;1970:CommonBase</td>
<td>Dahl, Myhrhaug, Nygaard</td>
<td>Common Base Language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Dahl;2002:2TheBirth</td>
<td>Dahl</td>
<td>The birth of object orientation: the Simula languages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Dahlstrand;2009:Minnen</td>
<td>Dahlstrand</td>
<td>Ingerman Dahlstrands Minnen: Wrote Algol 60 compiler for BEKS and Facit; see pages compilers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Damas;1982:PrincipalType</td>
<td>Damas, Milner</td>
<td>Principal type-schemes for functional programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Dijkstra;1960:RecursivePr</td>
<td>Dijkstra</td>
<td>Recursive programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Dijkstra;1961:Algol60Tang</td>
<td>Dijkstra</td>
<td>Algol 6 translation. An Algol 6 Compiler and runtime organization, basically stack of a compiler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Dijkstra;1963:MakingA</td>
<td>Dijkstra</td>
<td>Making a translator for Algol 6; slightly generalized presentation of Dijkstra;1961:AnAlgol60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Duncan;1962:Implementation</td>
<td>Duncan</td>
<td>Implementation of Algol for the English Electric KDF9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Endres;2013:EarlyLanguage</td>
<td>Endres</td>
<td>Early language and compiler developments at IBM Europe: a personal retrospective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Ershov;1958:OnProgramm</td>
<td>Ershov</td>
<td>On programming of arithmetic subroutines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Ershov;1959:AutomaticPro</td>
<td>Ershov</td>
<td>Automatic programming in the Soviet Union</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Ershov;1959:ProgrammingPro</td>
<td>Ershov</td>
<td>Programming programme for the BESM computer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Ershov;1959:TheWork</td>
<td>Ershov</td>
<td>The work of the computing centre of the Academy of Sciences of the USSR in the field of automatic programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Ershov;1966:Alpha</td>
<td>Ershov</td>
<td>Alpha, an automatic programming system of high efficiency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Ershov;1976:Addendum</td>
<td>Ershov</td>
<td>Addendum</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Jeg har ikke kunnet finde ...

- Information om en Algol-oversætter til SMIL af Ekman og Robertson, Lund
- Information om en Algol-oversætter til Ferranti Mercury fra NDRE = Forsvarets Forskningsinstitut, Oslo, fornemlig O.-J. Dahl