GNAT:

An Ada-9X compiler for everyone

The Gnat Project

New York University

June 1995

GNAT

The GNU NYU Ada Translator

- A high-quality Ada9X compiler.
- Part of the GCC compiler system
- Multiple targets and cross-compilers.
- Distributed under the guidelines of the Free Software Foundation
- Full availability of sources
- Latest version 2.6 just released.
- Expected completion: 2d quarter, 1995

The Free Software Foundation

- A not-for-profit organization dedicated to the production of quality software to be distributed freely with source code.
- Created by Richard Stallman
- Main products: GCC,GDB, Emacs, utilities
- Goal: GNU, a UNIX-compatible operating system

FSF Policies

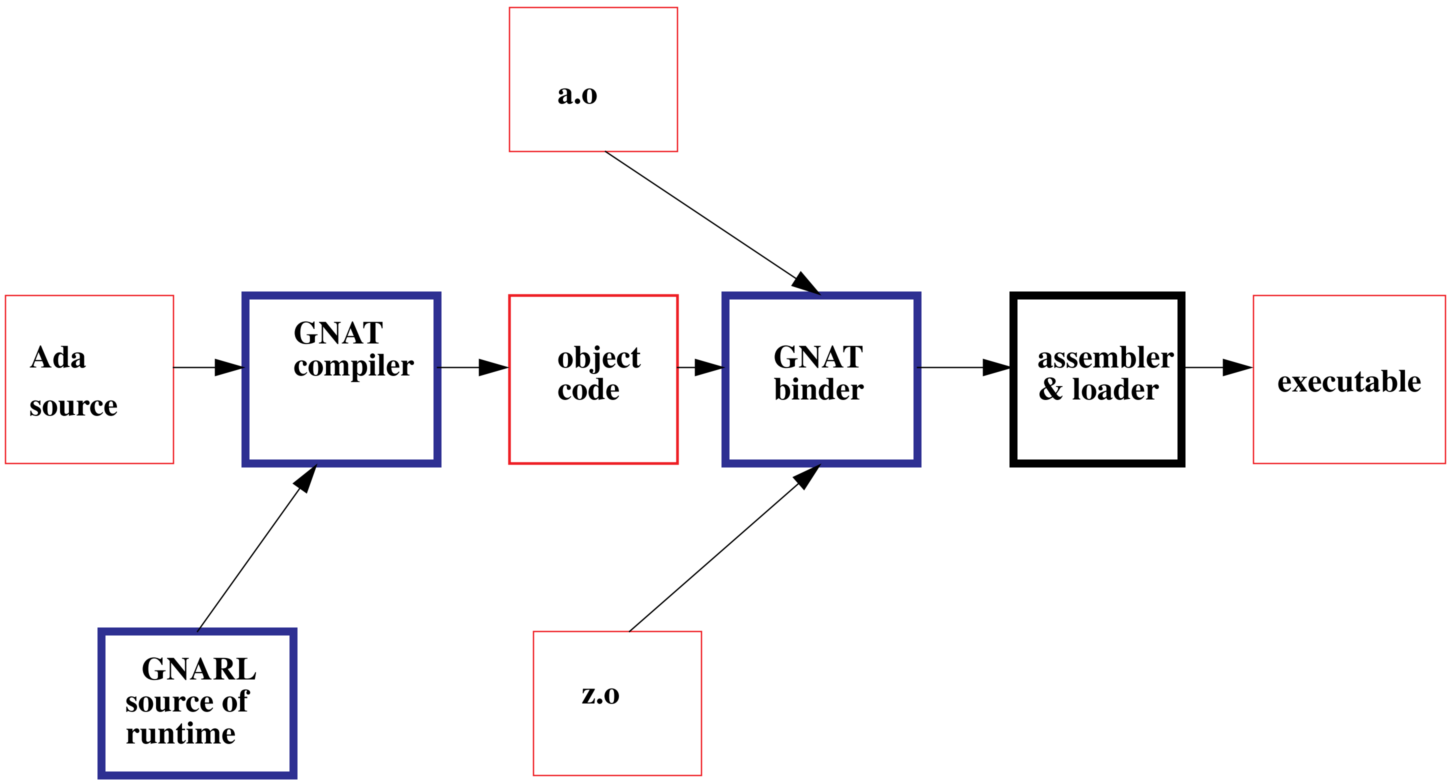
- Embodied in two software licensing mechanisms: FSF software is not in the public domain.
- GPL : GNU public licence.
- Complete freedom to copy and modify. However, if modified version is distributed, sources with their modification must be made available on request.
- GLPL : GNU library public license. Used for RTL components that can be incorporated into proprietary products without having them fall under the GPL license. For example, use of Atan2 does not apply GPL to a program compiled with GCC

Anatomy of a GNAT

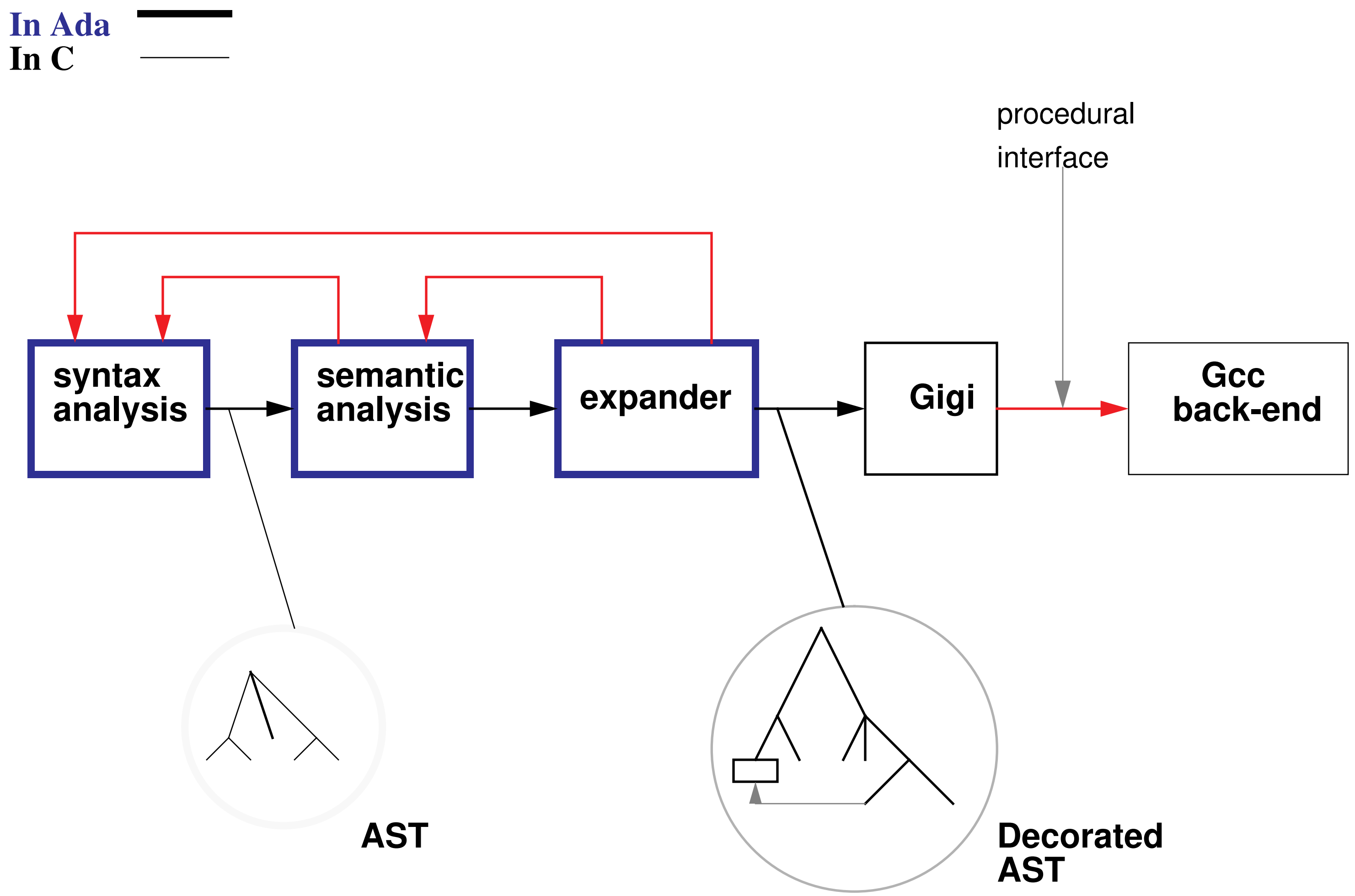
- The structure of GCC
- The GNAT front-end
- Gigi
- Library Management
- Binder
- The runtime
- An improved GCC

The Structure of GCC

- A multi-language retargettable compiler.
- Currently 30 hardware targets,180 configurations.
- Front-ends for C, C++, Objective C, Pascal, Modula-3, Fortran77, others in preparation.
- Code generator mostly machine-independent, uses RTL description of target architecture.
- Extensive optimization, 17 passes.
- Excellent code quality for both CISC and RISC architectures (68K, i86, RIOS, Alpha, 88K,...)
- Common code-generator eases multi-language support.
- > 400,000 lines of C.



The GNAT system



The Phases of the GNAT Compiler

Syntax Analysis

- Hand-coded top-down parser.
- No complex buffering: source file fully in memory.
- Sophisticated error recovery:
  - **Keyword spelling correction**
  - **Recognize indentation**
  - **Excellent scope recovery (";" vs. "is")**
  - **Internal style rules enforced.**
- Fastest portion of compiler
- 22,000 lines: one parent unit, 13 subunits.
- scanner: 3,000 lines. Tightly coded for keyword recognition, and skipping blanks and comments.

Why not LALR (k)?

- Clarity: corresponds exactly to grammar of Reference Manual.
- Performance: an order of magnitude better than table driven.
- Error recovery: always superior in top-down parsing. Heuristics cannot be encoded easily in table-driven methods. Special cases are constantly added to the system, at users' request.

Semantic Analysis

- Name resolution
- Type and overload resolution
- Dispatching and controlling arguments.
- Static expression evaluation.
- Legality checks: currently 770 error messages (still incomplete)
- 54,000 lines of code in 44 files.

Recursion solves everything!

- Need to analyze context clause: call the parser to process package declaration,call the semantics to analyze it, and return to processing of main program.
- Need to process stub: call parser to analyze subunit, call semantics, insert in place of stub, continue with main program.
- Need to perform generic expansion: call parser to process body of enclosing unit, call semantics to analyze it, perform in-line expansion.

Expansion

- The semantics of Ada are more complex than those of C. Higher-level constructs must be expanded to ease translation.

    - **Aggregates**
    - **Tasking / protected types constructs**
    - **Logical operations on boolean arrays**
    - **Equality and comparisons on composite types**
    - **Discriminant references in expressions**
    - **Initialization procedures.**
    - **Distribution annex.**

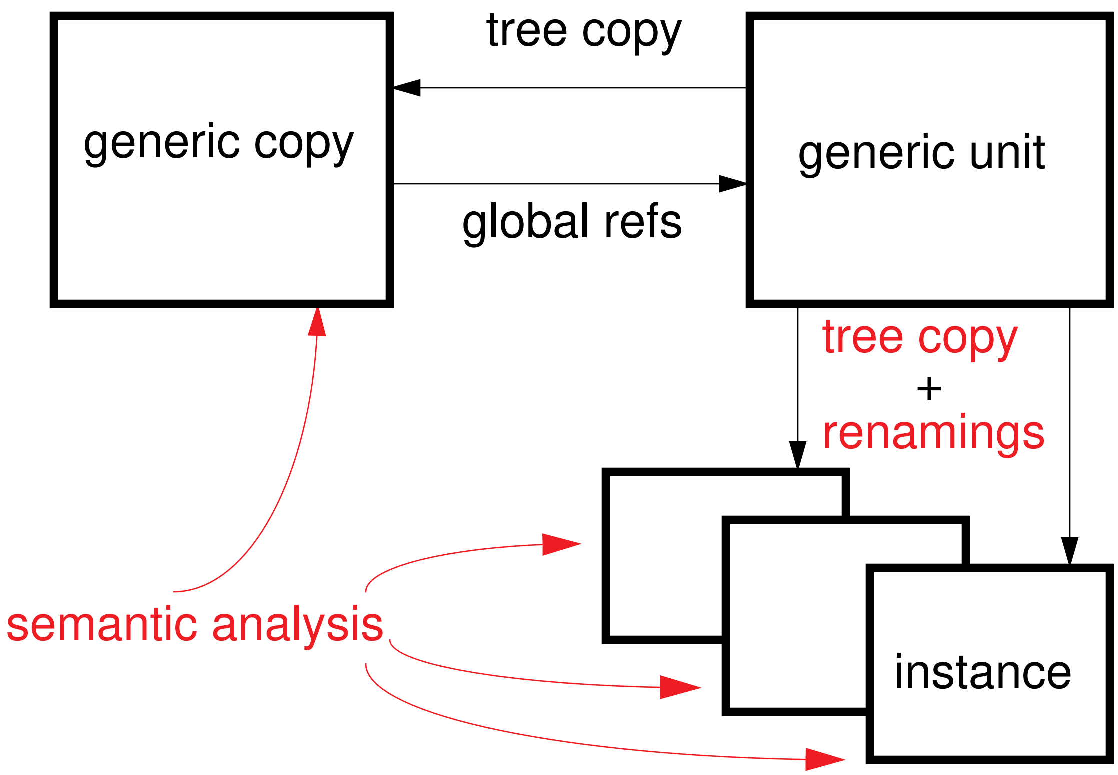
- Build tree fragments, call semantics recursively on them

- A good hacker can write LISP in any language!

- 35,000 lines in 45 files.

Generic Analysis and Instantiation

- Standard model: macro expansion.
- Need to perform limited semantic analysis, and inhibit tree expansion.
- Repeat most semantic analysis at instantiation time, except name resolution



GIGI : GNAT to GCC

- Impratical to use GCC data structure in the front-end, and vice-versa.
- Gigi traverses GNAT tree fragments and calls tree-building procedures in GCC.
- Code generation invoked on each tree fragment at once: there is no full tree representation of program in GCC.
- Gigi needs some knowledge of Ada semantics, and accesses syntactic and semantic information modules.
- Written in C (15,000 lines) to simplify interface with back-end,but calls common syntax and semantic information packages.

Library Management

- Compilation rules are source-based
- No centralized library structure, no intermediate representations for semantic analysis.
- Object files only depend on source files, never on other object files
- Dependency information (time-stamps of semantic dependences) is embedded in object file.

Advantages of a source-based library

- No required order of compilation
- No complex library structures: cost of "recompilation" of package declarations is not significant.
- No accidental obsolescence (familiar mishap: accidental compilation removes dependents from library, even when source did not change).
- Consistent with typical use of make tools: familiar to programmers of other persuasions, eases multilanguage programming.
- Consistent with GNU philosophy: a program is its sources.
- More complex inlining is possible.
- Library management in GNAT is almost invisible: < 2000 lines to read/write dependency information.

File Organization

- One compilation unit per file.
- Uniform naming conventions:
  - test.ads denotes a specification
  - test.adb denotes a body
  - test.ali denotes library information
  - test-subtest.adb denotes a subunit or a child unit.
- Environment variables for paths to predefined units, eg. system and family, and for multiple libraries (e.g. multi-partition vs. single partition versions of categorized packages)..

The Binder

- Build main program and code to invoke all elaboration procedures in proper order.
- Verify semantics of library management
- Flexible enforcement:
  - full: recompile object if any source on which it depends is more recent.
  - permissive: assemble available objects, ignore time stamps.
- Basis for a sophisticated AdaMake tool.
- Allows foreign language main program.

The Runtime

- Portable tasking support, written in Ada.
- Two interfaces:
- GNARL (GNU Ada Run-time Library) between compiler and RTS.
- GNULL (GNU low-level library) between RTS and underlying OS.
- Based on POSIX threads.
- Minimal assembly glue.
- Source-based compilation allows inlining of RTS routines without forcing knowledge of RTS into compiler.
- User (if well-informed) can replace run-time modules.

GCC learns some Ada

- Some aspects of Ada semantics must be known to the back-end
- The GCC approach: if it’s good for more than one language it belongs in the code generator.
- Discriminants and record layout (in Gigi)
- Constraint-checking. (in Gigi)
- Trapping arithmetic (in machine description tables)
- Exception management (in RTS, with stack management)
- GCC is becoming more strongly-typed.

GNAT Status

- Current release version: 2.06
- Ports to SUN workstations: SunOS^{*+}, Sun/Solaris^{*+}
- ports to DEC/Alpha: OSF2.0 - 3.0+, DEC/MIPS
- ports to SGI: IRIX 4, IRIX 5.4^{+}
- port to HP: HPUX
- port to RS6000: AIX^{+}
- others: AmigaDOS, 1750A (cross), DG/Avion.
- ports to i86 machines: DOS/Windows^{+}, Windows95, OS/2^{*+}, NT^{+}, Solaris86^{+}, Nextstep, FreeBSD, NetBSD*, SCO Unix, Linux^{+}
- * indicates support for tasking
- $^{+}$ indicates use by GNAT team.

GNAT Maintenance Today

- Several thousand users
- Total problem reports received: 6000
- 80% immediately dealt with:
  - pilot error
  - already fixed
  - immediately fixed
- about 1400 filed bugs
  - 1200 fixed
  - 200 open (6/9/95)

Status of Language Annexes

- **Interface to other languages:** available for C and FORTRAN.
- **Systems programming:** ongoing collaboration with FSU team.
- **Real-time systems:** ongoing collaboration with FSU team.
- **Distributed systems:** PVM-based prototype developed by ENST-Bretagne. Ada version developed at ENST-Paris. Configuration tool being developed at Texas A & M.
- **Information systems**: collaboration with MITRE.
- **Numerics:** collaboration with ANL and GE.
- **Safety and security**

The GNAT Team

- **At New York University:** Bernard Banner, Cyrille Comar, Robert Dewar, Sam Figueroa, Richard Kenner (GCC), Brett Porter, Ed Schonberg, Gail Schenker.
- **At Florida State University** : Ted Giering, Ted Baker, Dong-Ik Oh, Chris Bray (tasking, protected objects)
- **at Telecom Paris**: Franco Gasperoni, Laurent Pautet, Patrick Bazire (cross-reference tools, distribution, gnatmake, Emacs tools)
- **at Telecom Bretagne:** Yvon Kermarrec, Laurent Nana (distribution)
- **Elsewhere**: Doug Rupp (Washington), Michael Feldman and Chuck McCann (the other Washington) Paul Hilfinger (Berkeley), Jean-Pierre Rosen (Paris), Jon Squires (GE) many others to whom thanks are due.
- **your name here**.............
-

A long-lived GNAT

- Current developement supported by Ada9X project office and US Air Force, contract ending June 30,1995.
- Commitment to free distribution of system with sources.
- Need maintenance organization to provide user support, ports to new architectures, hand-holding for new users, maintenance vis-a-vis of interpretations of future ARG, etc.
- **Industrial users need to know product is maintained.**
- Commercial Entity: Ada Core Technologies
- Software remains free, support is not.
- ACT provides maintenance contracts to OEMs and to individuals. Other maintenance provided by LABTEK (NT) and SGI (IRIX).

GNAT in the NYU Classroom

- In CS2: using Feldman's Ada95 text. Other sections still use Pascal. Evaluation pending (good students loved it!).

- In Programming Languages courses: graduate and undergraduate. Ada is the one mainstream language studied in depth. Also used when discussing concurrency.

- In Compiler construction: sources of GNAT make it possible for students to modify an industrial compiler (language extensions, optimizations, borrowings from C++, CLOS, etc).

- In Robotics: device drivers for miniature vision system (cross-compiled to 68881 controller). Device drivers for DNA sequencing equipement.

Points of Contact

- addresses:
  - **dewar@cs.nyu.edu**
  - **schonberg@cs.nyu.edu**
  - **gnat-report@cs.nyu.edu**
- To obtain system:
  - **ftp to cs.nyu.edu, directory pub/gnat**
- sources always available
- binaries for SPARC-SunOS and Solaris, for i86-OS/2 - DOS - NT, Linux, Alpha/OSF, RS6000, SGI, pointers to others.
- ports elsewhere to Amiga-DOS, 1750A, etc.