CodeBoost

A Framework for C++ Transformation

Otto Skrove Bagge, Bergen

joint work with

Magne Haveraaen, Bergen

Eelco Visser, Utrecht

12th Nordic Workshop on Programming Theory
Bergen, October 2000

http://www.ii.uib.no/~nwpt00/

Source-to-source Transformation

- Useful for
  - Optimisation
  - Instrumentation
  - Language extension / prototyping
  - Refactoring

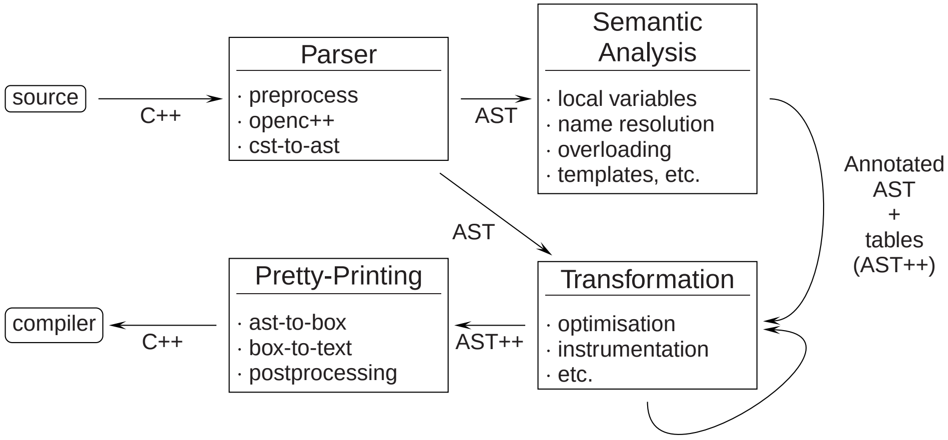
Requirements

- Parse and pretty-print simplified C++
- Type and other semantic information
- Extensible: experiment with transformations
- Preserve semantics
- Cascading transformations
- High-level specification of transformations

CodeBoost provides

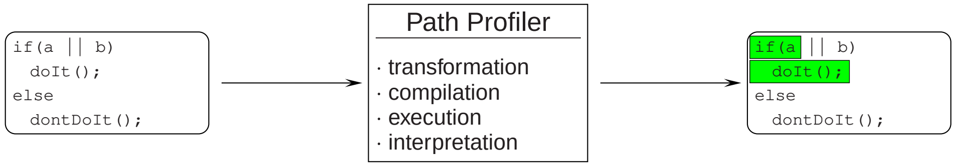
- syntactic analysis: C++ input
- semantic analysis: typing, name resolution, overloading, ...
- pretty-printer: C++ output
- library of transformations: type matching, inlining, renaming, ...

The Transformation Process



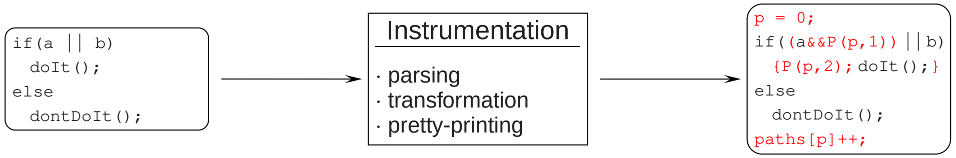
Example Application: Instrumentation

- Adding instrumentation code to obtain information for
  - Coverage Analysis—statements, conditions, paths
  - Performance Profiling—statements, loops, functions
  - Call graphs—function calls
  - Debugging

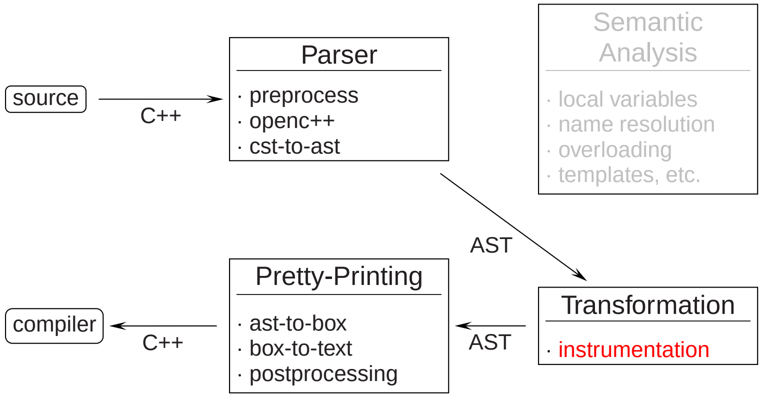


Example Application: Instrumentation
Path Profiling

- Requires
  - Numbering each possible execution path in a function
  - Keeping track of which paths have been executed
  - Shading the source code to display a path or subpath
- Implemented by
  - A single plug-in for both instrumentation and shading



Example Application: Instrumentation
The Instrumentation Process



Example Application: Sophus

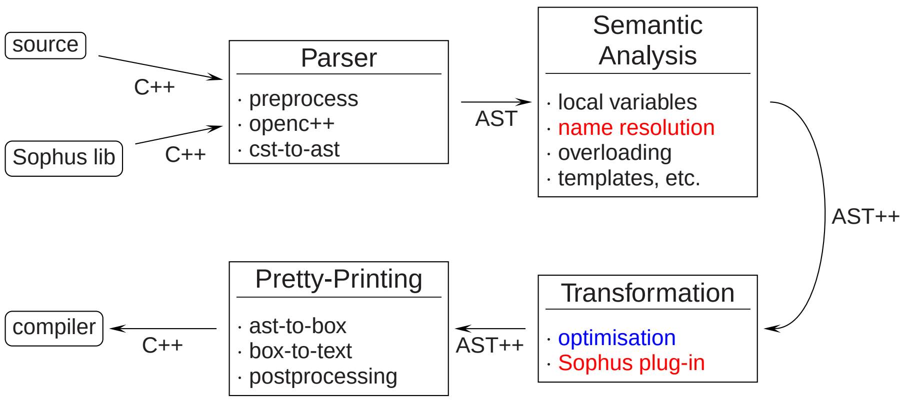
- High-level abstractions for HPC
- C++ code close to mathematical style
- Current compilers can't optimise
- A Sophus-specific optimiser is needed

Example Application: Sophus Optimiser

- Requires
  - syntactic and semantic analysis
  - inlining
  - loop fusion optimisation
- Implemented by
  - modifications to the semantic analyser
  - transformation plug-in: mutification

Example Application: Sophus

The Optimisation Process



Example Application: Sophus Optimiser

Adding a new Transformation

- Requirements
  - For user-defined operators and functions, apply *mutification*:

```
x = x + e;   →  x += e;
x = f(x,e);  →  x.uf(e);
```

  - Declarations for `+=` and `uf()` are given in Sophus library
- Implemented by
  - The semantic analyser is modified to add declarations for `+` and `f()`, and a table mapping them back to real methods
  - A new transformation plug-in is added to perform the rewriting

Example Application: Sophus Optimiser
Stratego Implementation of Mutification

```
module mutification
imports lib codeboost
rules
  asgnop: Infix(Op("=", _), x, Infix(o, x, y)) ->
          Infix(Op(s, o'), x, y)
        where <lookup-mutating-op> o => Op(s, o')

  asgnfun: Infix(Op("=", _), x, App(f, a)) ->
           App(Fun(Member(o, Name(f'), t), s), a')
        where <lookup-mutating-fun> (f, x, a) => Fun(f', s)
strategies
  mutificate = bottomup(try(asgnop + asgnfun))
  main = cbio(mutificate)
```


Ongoing Work

- Language coverage
  - Template expansion
  - Other language features are added as they are needed
- Transformation library
  - Inlining
  - Loop fusion
- Sophus optimisations
  - Alternating traversal of data arrays
  - Selective multi-level inline strategies
  - Application of domain specific algebraic laws

Conclusion

- CodeBoost is an open framework for C++ transformation, and provides
  - basic transformation infrastructure: parsing, semantic analysis, pretty-printing
  - generic transformation library
  - a basis for plug-in transformations
- CodeBoost is Open Source, and can be freely modified and extended
- For more information: http://www.stratego-language.org/codeboost/

CodeBoost Implementation

- Preprocessing is done using Perl and a standard C++ preprocessor
- Parsing is done by OpenC++
- Everything else is written in Stratego, a language for specifying transformations based on rewriting strategies
- Intermediate output is in human-readable ATerm format
- Plug-ins can be written in any language, but Stratego is very convenient

Existing Technology

- Perl
  - only supports lexical rewriting
- ASF+SDF
  - traversals are complicated
  - semantic analysis is difficult
- OpenC++
  - type-oriented transformations
  - low level definition of transformations
  - currently doesn't support cascading transformations