Cetus for C, C++, and Java

LCPC 04 Mini Workshop of Compiler Research Infrastructures

http://www.ece.purdue.edu/ParaMount/Cetus

In this tutorial

- Why we created Cetus and what it is
- Architecture of Cetus
- Capabilities

Why Cetus?

- Wanted source-to-source C, C++, Java compilers
  - Polaris only works on Fortran 77
  - GCC not source-to-source
  - SUIF is for C; must extend IR class hierarchy for C++ and Java; last major update was 2001
- Wanted a compiler written in a modern language
  - Polaris and SUIF use old dialects of C++ (pre-standard templates)
- Best alternative was to write our own

Cetus is useful for...

- Program analysis at the source level
- Source-code instrumentation
- Transform source code into a “normalized” form for use with other programs or scripts
- But not if
  - you want to do back-end compiler work
  - other infrastructures are more appropriate for that

What is Cetus?

- Cetus proper
  - Written in Java
  - C parser (Antlr)
  - Intermediate representation (10K+ lines; stable)
  - Passes (1.5K+ lines; growing)
  - Parse-tree walker & disambiguator (discussed later)
  - Available for download
    - license similar to Perl
  - Written by 3 grad students, part-time, 2 years

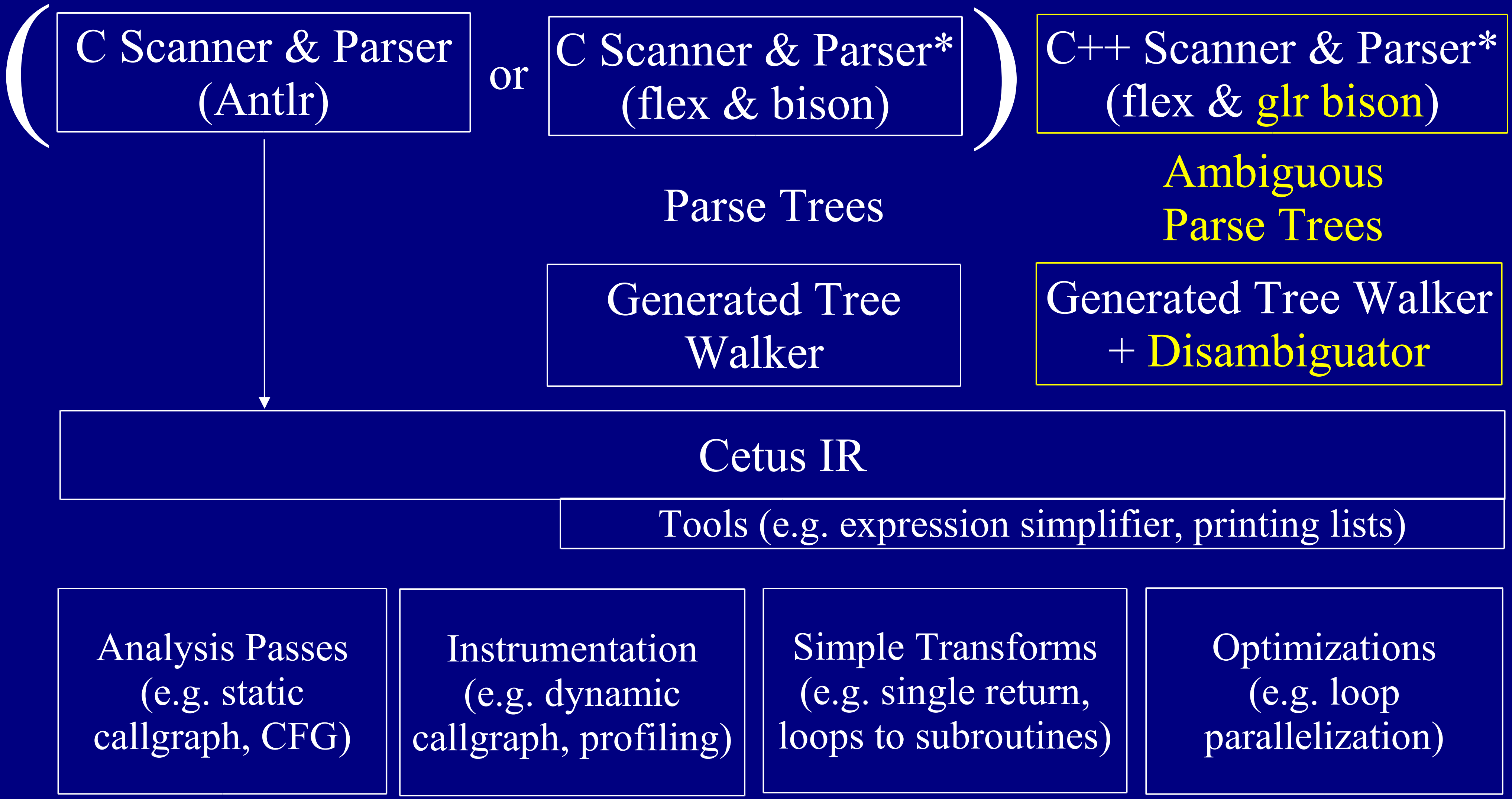
What is Cetus? (continued)

- Separate, useable with Cetus or other programs
  - C (Bison) & C++ (GLR-Bison) parsers
  - Written in C++
  - Creates parse trees for Cetus to read
  - Works fine separately; still integrating with Cetus
  - Not yet available for download
    - uses GNU code, license GPL
  - Written by me in about a month

Running Cetus

- export CLASSPATH=cetus.jar:antlr.jar
- java cetus.exec.Driver -antlr [other options] *.c
- Cetus uses an existing preprocessor (e.g. cpp)
  - output still contains #include directives
  - macros remain expanded
- Cetus output goes in a subdirectory
  - source files have the same name as input files
  - not pretty-printed (use indent or astyle)
  - some passes generate graphviz-compatible graphs

Architecture



(C Scanner & Parser (Antlr) or C Scanner & Parser* (flex & bison))

C++ Scanner & Parser* (flex & glr bison)

Parse Trees

Ambiguous Parse Trees

Generated Tree Walker

Generated Tree Walker + Disambiguator

Cetus IR

Tools (e.g. expression simplifier, printing lists)

Analysis Passes (e.g. static callgraph, CFG)

Instrumentation (e.g. dynamic callgraph, profiling)

Simple Transforms (e.g. single return, loops to subroutines)

Optimizations (e.g. loop parallelization)

* indicates a separate program

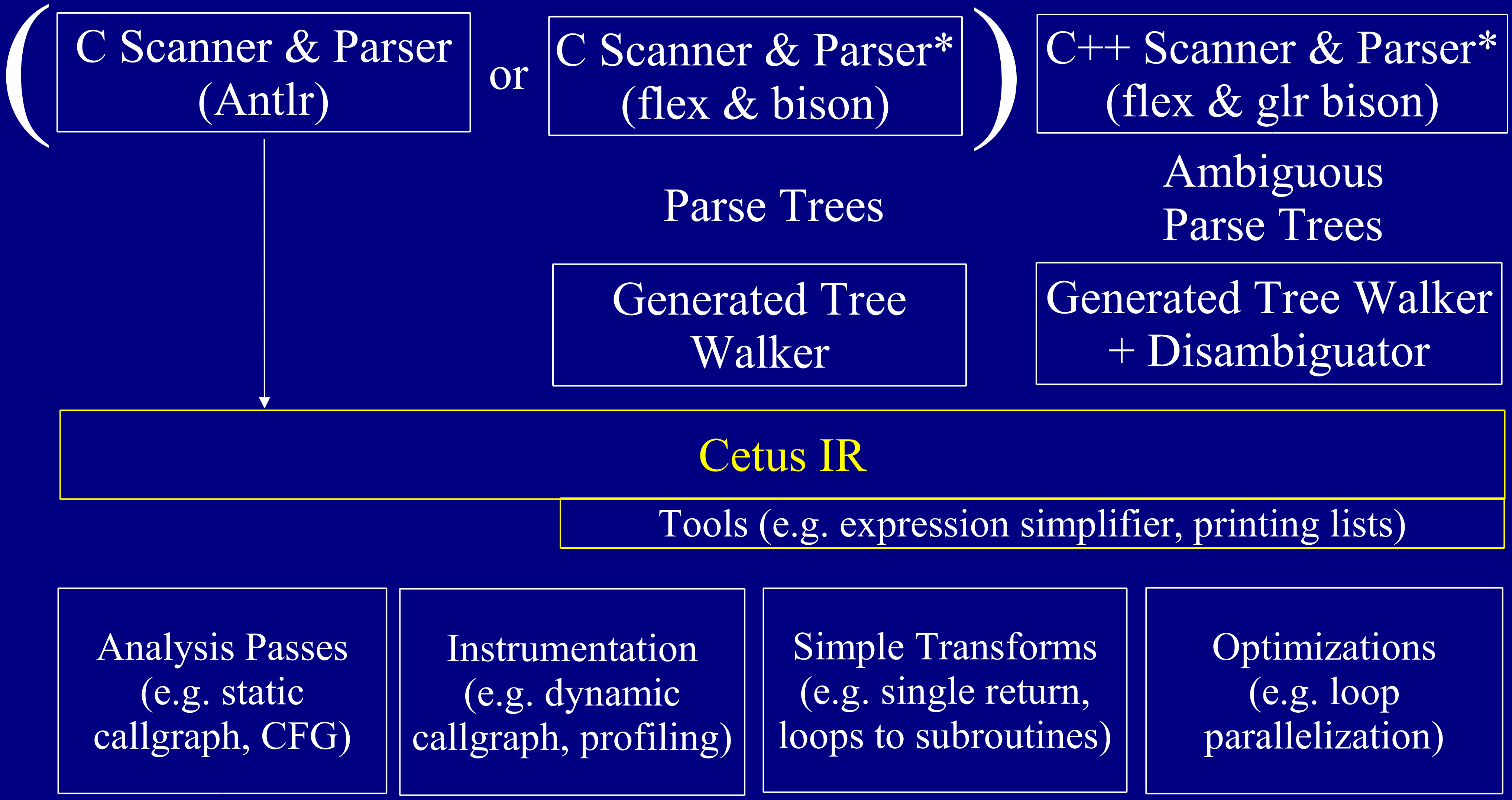
Parsing C++

- Would like to use the actual grammar
  - not compatible with Antlr or yacc/bison without a lot of rewriting (e.g. gcc < 3.4)
  - don't want to write a custom parser (e.g. gcc >= 3.4)
- Bison has recently acquired a GLR (generalized LR) parsing mode
  - accepts unmodified grammar
  - can be used to separate syntax from semantics
  - but generates ambiguous parse trees

Parsing C++ (cont.)

- Cetus approach
  - use glr-bison to read the program and write its parse tree to a file
  - parse tree contains “ambiguity” nodes where only one of the child trees is correct
  - Cetus reads the parse tree and runs a “tree walker” on it to generate IR while resolving ambiguities

Architecture



(C Scanner & Parser (Antlr) or C Scanner & Parser* (flex & bison))

C++ Scanner & Parser* (flex & glr bison)

Parse Trees

Ambiguous Parse Trees

Generated Tree Walker

Generated Tree Walker + Disambiguator

Cetus IR

Tools (e.g. expression simplifier, printing lists)

Analysis Passes (e.g. static callgraph, CFG)

Instrumentation (e.g. dynamic callgraph, profiling)

Simple Transforms (e.g. single return, loops to subroutines)

Optimizations (e.g. loop parallelization)

* indicates a separate program

Cetus High-Level IR

- Basic design principles and consequences
  - must be able to reproduce the source code

    => IR models language
  - should prevent mistakes by pass writers

    => invariants enforced on entry to IR methods
  - support interprocedural analysis

    => all source files represented in IR simultaneously
  - should be simple and compact

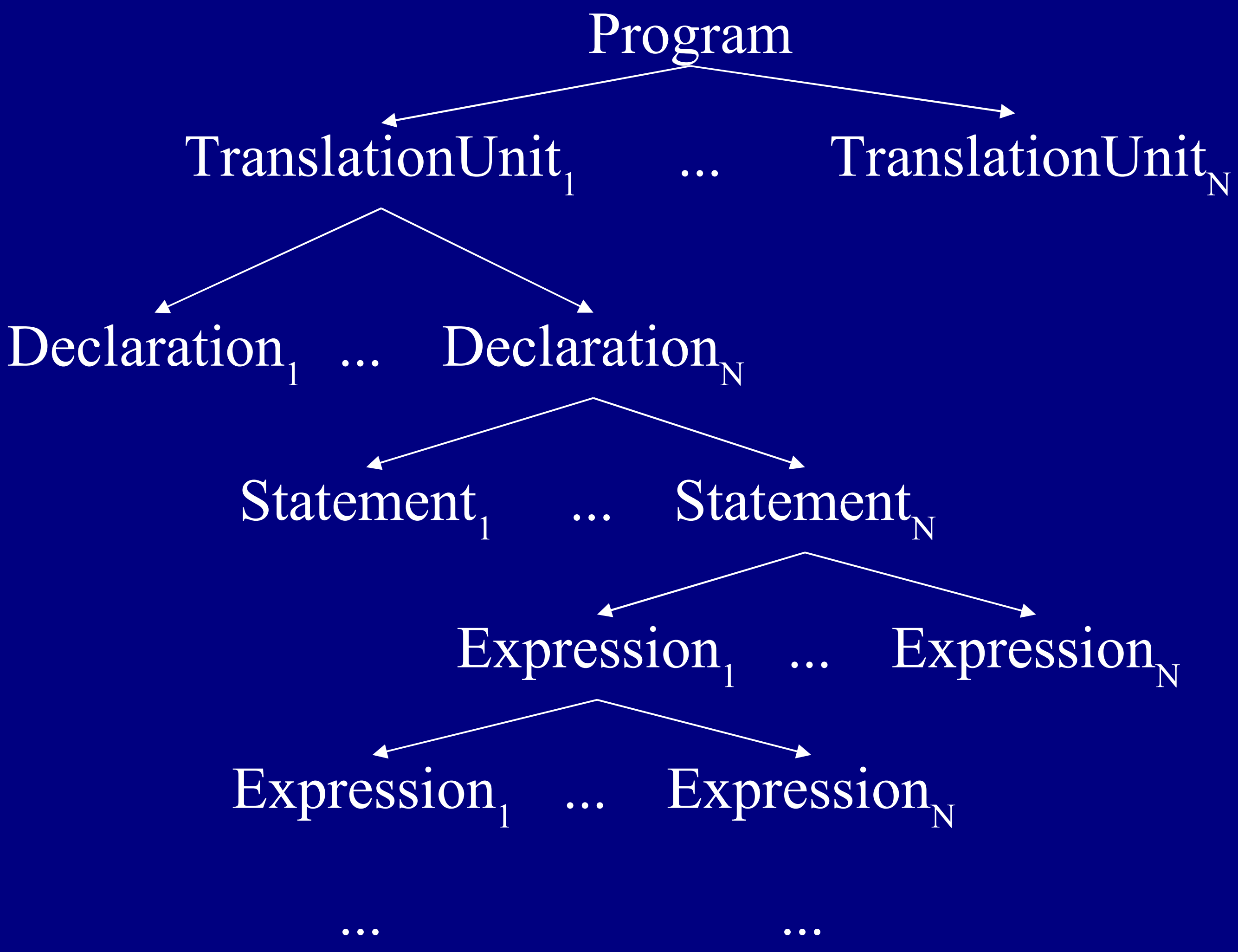
    => shallow class hierarchy for IR (at most 3 levels deep)

Major Parts of IR Class Hierarchy

- Program
- TranslationUnit
- Declaration
  - Annotation
  - Procedure
  - VariableDeclaration
  - …
- Statement
  - ForLoop
  - WhileLoop
  - …
- Expression
  - BinaryExpression
  - FunctionCall
  - …

- IRIterator
  - BreadthFirstIterator
  - DepthFirstIterator
  - FlatIterator

IR Tree != Class Hierarchy Tree



Program
TranslationUnit$_1$ … TranslationUnit$_N$
Declaration$_1$ … Declaration$_N$
Statement$_1$ … Statement$_N$
Expression$_1$ … Expression$_N$
Expression$_1$ … Expression$_N$
… …

Iterating Over IR Tree

- Iterators provided for Breadth, Depth, and Flat (single-level) search order
- Work like normal Java Iterators, except
  - next(Class c) returns the next object of Class c
  - next(Set s) returns the next object of a Class in Set s
  - pruneOn(Class c) forces the iterator to skip everything beneath objects of Class c

Iteration Examples

```
/* Look for loops in a procedure.  Assumes proc is a Procedure
   object. */

BreadthFirstIterator iter = new BreadthFirstIterator(proc);
try {
  while (true)
  {
    Loop loop = (Loop)iter.next(Loop.class);
    // Do something with the loop
  }
} catch (NoSuchElementException e) {
}
```

Iteration Examples (cont.)

```
/* Look for procedures in a program.  Assumes prog is a Program
   object.  Does not look for procedures within procedures. */

BreadthFirstIterator iter = new BreadthFirstIterator(prog);
iter.pruneOn(Procedure.class);

try {
  while (true)
  {
    Procedure proc = (Procedure)iter.next(Procedure.class);
    // Do something with the procedure
  }
} catch (NoSuchElementException e) {
}
```

Symbol Table Management

- Some IR classes implement SymbolTable interface
  - provides addDeclaration, findSymbol, etc.
- Adding (removing) a declaration adds (removes) symbols automatically
- Symbol table maps an IDExpression onto the Declaration that put it in the table
  - mapping is one-to-one if SingleDeclarator pass is run
  - use findSymbol twice then == to see if same symbol

Symbol Table (cont.)

- Searching a SymbolTable searches its parent tables if the symbol is not found
  - parent table not necessarily parent on IR tree
  - can have multiple parent tables (C++ multiple inheritence)
  - but only one IR-tree parent (syntactically enclosing parent)

Error Detection

- IR methods throw exceptions:
  - DuplicateSymbolException
    - if a name collision occurs in the symbol table
  - NotAChildException
    - if an IR object should be a child of another, but isn't
  - NotAnOrphanException
    - if an IR object should not be a child of another, but is

Customized Printing

- Problem: Same IR classes for different languages
  - e.g. ClassDeclaration for C++ and Java
  - C++ class terminates with a ';' and Java classes don't
  - What should the print method do?
- Solutions
  - additional classes or flags to indicate language
  - customized printing <-- Cetus uses this
- Why stop with a few classes?

Customized Printing (cont.)

- Most classes have a static Method class_print_method member
  - set to a default print method in static init block
  - constructor initializes a non-static object_print_method member to class_print_method
  - print(OutputStream stream) invokes object_print_method with this and stream as args
- Class has static setClassPrintMethod(Method)
- Also non-static setPrintMethod(Method)

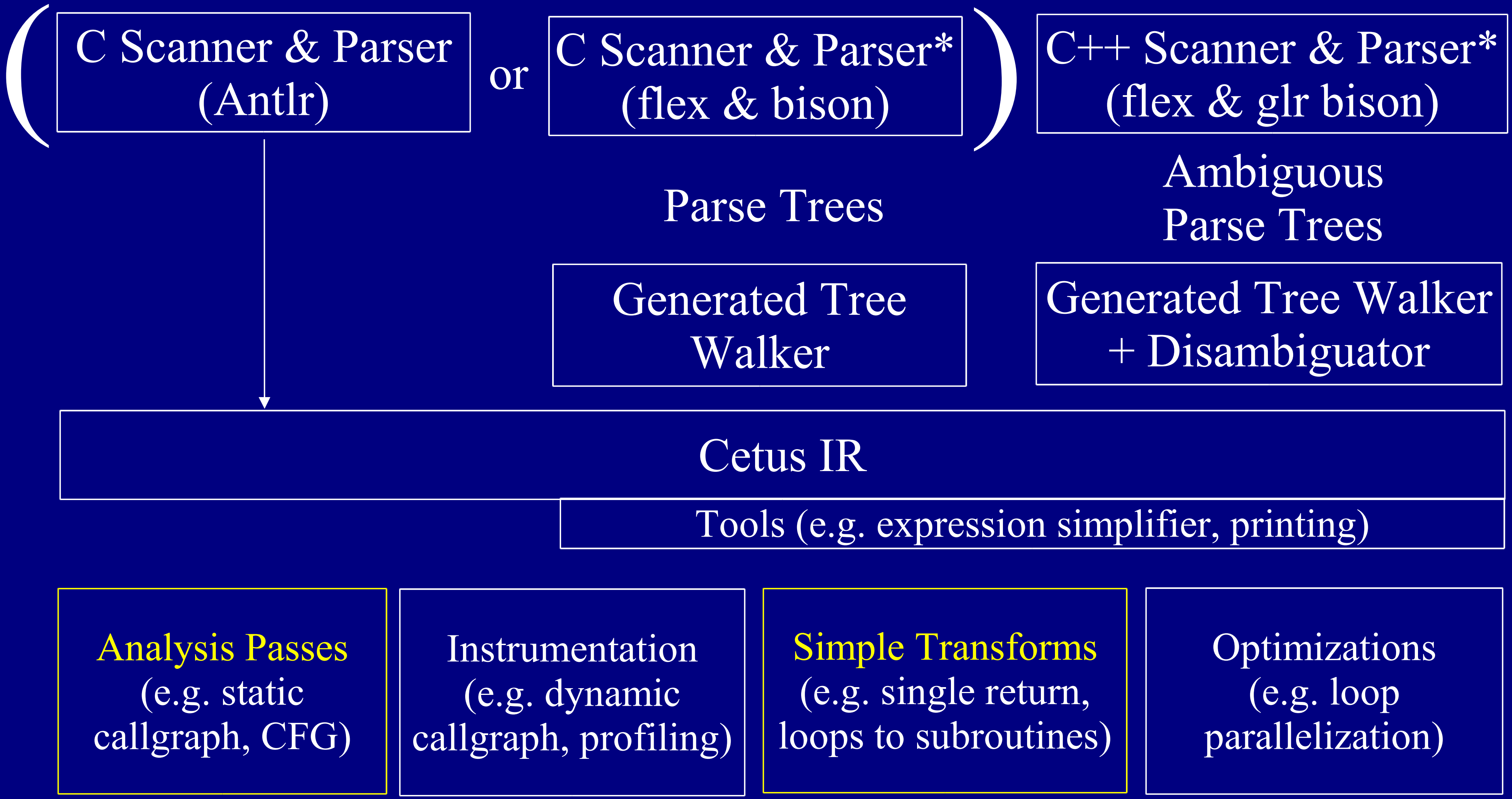
Customized Printing (cont.)

- Benefits
  - can change printing for all instances of an IR class
    - quick way to add simple instrumentation
  - can change printing for a particular instance
    - i.e. we may wish to print a parallel loop differently
  - can set print method to null to hide code in output
- Costs
  - one static and one non-static variable
  - slower printing (not usually a big deal)
  - toString() kept consistent by printing to a buffer
    - but not often used on large parts of the tree

Annotations

- Subclass of Declaration
  - can appear in IR tree anywhere a declaration can
- Stores either
  - a single String
  - a Map of String keys onto String values
- Printable as
  - //-style comment, /**/ comment, pragma, raw text
- Facilitates instrumentation & information exchange among passes

Architecture



(C Scanner & Parser (Antlr) or C Scanner & Parser* (flex & bison))

C++ Scanner & Parser* (flex & glr bison)

Parse Trees

Ambiguous Parse Trees

Generated Tree Walker

Generated Tree Walker + Disambiguator

Cetus IR

Tools (e.g. expression simplifier, printing)

Analysis Passes (e.g. static callgraph, CFG)

Instrumentation (e.g. dynamic callgraph, profiling)

Simple Transforms (e.g. single return, loops to subroutines)

Optimizations (e.g. loop parallelization)

* indicates a separate program

Analysis Passes

- Call Graph
  - creates a static call graph for the program
- Control Flow Graph
  - creates a basic-block graph of a procedure
- Basic Use and Def set computation
  - lists values used and defined within a region

Transformation Passes

- Single Call
  - afterwards each statement contains at most one call
- Single Declarator
  - afterwards each declaration contains at most one declarator
- Single Return
  - afterwards each procedure contains at most one return
- Loops to Subroutines
  - extracts loops out into separate subroutines

Work in Progress

- Improved data flow analysis
- Pointer alias analysis
- Finish integrating C++ front end with Cetus
- Java front end

Cetus Used in Research

- At least 4 current projects at Purdue
- Pin Zhou, Wei Liu, Long Fei, Shan Lu, Feng Qin, Yuanyuan Zhou, Sam Midkiff and Josep Torrellas, AccMon: Automatically Detecting Memory-Related Bugs via Program Counter-based Invariants, to appear in Proc. of the 37th Annual IEEE/ACM International Symposium on Micro-architecture (MICRO 04), December 2004
- Sang-Ik Lee, Troy A. Johnson and Rudolf Eigenmann, Cetus - An Extensible Compiler Infrastructure for Source-to-Source Transformation, Proc. of the Workshop on Languages and Compilers for Parallel Computing (LCPC 03), October 2003.
- Seung-Jai Min, Ayon Basumallik and Rudolf Eigenmann, Supporting Realistic OpenMP Applications on a Commodity Cluster of Workstations, International Workshop on OpenMP Applications and Tools, WOMPAT 2003, Toronto, Canada, June 26-27, 2003.

Obtaining Cetus

- http://www.ece.purdue.edu/ParaMount/Cetus
- Only the C version is available for now
- First release was Aug 11; third was Sep 15
  - releases typically once or twice per month
- Contributions welcomed
  - new passes
  - bug fixes
  - suggestions