LNCS 3602

Rudolf Eigenmann
Zhiyuan Li
Samuel P. Midkiff (Eds.)

Languages and Compilers for High Performance Computing

17th International Workshop, LCPC 2004
West Lafayette, IN, USA, September 2004
Revised Selected Papers

Springer

Rudolf Eigenmann Zhiyuan Li
Samuel P. Midkiff (Eds.)

Languages and Compilers for High Performance Computing

17th International Workshop, LCPC 2004
West Lafayette, IN, USA, September 22-24, 2004
Revised Selected Papers

Springer

Volume Editors

Rudolf Eigenmann
Samuel P. Midkiff
Purdue University, School of Electrical and Computer Engineering
West Lafayette, IN 47906, USA
E-mail: {eigenman, smidkiff}@ecn.purdue.edu

Zhiyuan Li
Purdue University, Department of Computer Sciences
West Lafayette, IN 47906, USA
E-mail: li@cs.purdue.edu

Library of Congress Control Number: 2005929713

CR Subject Classification (1998): D.3, D.1.3, F.1.2, B.2.1, C.2.4, C.2, E.1, D.4

ISSN 0302-9743
ISBN-10 3-540-28009-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28009-5 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11532378 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3602

Preface

The 17th International Workshop on Languages and Compilers for High Performance Computing was hosted by Purdue University in September 2004 on Purdue campus in West Lafayette, Indiana, USA. The workshop is an annual international forum for leading research groups to present their current research activities and the latest results, covering languages, compiler techniques, runtime environments, and compiler-related performance evaluation for parallel and high-performance computing. Eighty-six researchers from Canada, France, Japan, Korea, P. R. China, Spain, Taiwan and the United States attended the workshop.

A new feature of LCPC 2004 was its mini-workshop on Research-Compiler Infrastructures. Representatives from four projects, namely Cetus, LLVM, ORC and Trimaran, gave a 90-minute long presentation each. In addition, 29 research papers were presented at the workshop. These papers were reviewed by the program committee. External reviewers were used as needed. The authors received additional comments during the workshop. The revisions after the workshop are now assembled into these final proceedings.

A panel session was organized by Samuel Midkiff on the question of "What is Good Compiler Research – Theory, Practice or Complexity?" The workshop also had the honor and pleasure to have two keynote speakers, Peter Kogge of the University of Notre Dame and David Kuck of Intel Inc., both pioneers in high performance computing. Peter Kogge gave a presentation titled "Architectures and Execution Models: How New Technologies May Affect How Languages Play on Future HPC Systems". David Kuck presented Intel's vision and roadmap for parallel and distributed solutions.

The workshop was sponsored by the National Science Foundation and by International Business Machines Corporation. Their generous contribution is greatly appreciated. We wish to acknowledge Purdue's Office for Continuing Education and Conferences, Thomas L. Robertson in particular, for their assistance in organizing the workshop. Eighteen graduate students affiliated with Purdue's Advanced Computer Systems Laboratory (ACSL) volunteered their time to assist in the workshop's operations. Our special thanks go to the LCPC 2004 program committee and the nameless external reviewers for their efforts in reviewing the submissions. Advice and suggestions from both the steering committee and the program committee have helped the smooth preparation of the workshop. Finally, we wish to thank all the authors and participants for their contribution and lively discussions which made the workshop a success.

May 2005 — Rudolf Eigenmann, Zhiyuan Li, Samuel P. Midkiff

Organization

Committees

Program Co-chairs:	Rudolf Eigenmann (Purdue University, USA) Zhiyuan Li (Purdue University, USA) Samuel P. Midkiff (Purdue University, USA)
Program Committee:	Nancy Amato (Texas A&M University, USA) Rudolf Eigenmann (Purdue University, USA) Zhiyuan Li (Purdue University, USA) Sam Midkiff (Purdue University USA) Bill Pugh (University of Maryland, USA) J. Ramanujam (Louisiana State University, USA) Lawrence Rauchwerger (Texas A&M University, USA) P. Sadayappan (Ohio State University, USA) Bjarne Stroustrup (Texas A&M University, USA) Chau-Wen Tseng (University of Maryland, USA)
Conference Co-chairs:	Rudolf Eigenmann (Purdue University, USA) Zhiyuan Li (Purdue University, USA) Samuel P. Midkiff (Purdue University, USA)
Steering Committee:	Utpal Banerjee (Intel Corporation, USA) Alex Nicolau (University of California, Irvine, USA) David Gelernter (Yale University, USA) David Padua (University of Illinois at Urbana-Champaign, USA)

Sponsors

National Science Foundation, USA
International Business Machines Corporation

Lecture Notes in Computer Science

For information about Vols. 1–3513

please contact your bookseller or Springer

Vol. 3633: C.B. Medeiros, M. Egenhofer, E. Bertino (Eds.), Advances in Spatial and Temporal Databases. XIII, 433 pages. 2005.

Vol. 3632: R. Nieuwenhuis (Ed.), Automated Deduction - CADE-20. XIII, 459 pages. 2005. (Subseries LNAI).

Vol. 3626: B. Ganter, G. Stumme, R. Wille (Eds.), Formal Concept Analysis. X, 349 pages. 2005. (Subseries LNAI).

Vol. 3621: V. Shoup (Ed.), Advances in Cryptology – CRYPTO 2005. XI, 568 pages. 2005.

Vol. 3619: X. Lu, W. Zhao (Eds.), Networking and Mobile Computing. XXIV, 1299 pages. 2005.

Vol. 3615: B. Ludäscher, L. Raschid (Eds.), Data Integration in the Life Sciences. XII, 344 pages. 2005. (Subseries LNBI).

Vol. 3608: F. Dehne, A. López-Ortiz, J.-R. Sack (Eds.), Algorithms and Data Structures. XIV, 446 pages. 2005.

Vol. 3607: J.-D. Zucker, L. Saitta (Eds.), Abstraction, Reformulation and Approximation. XII, 376 pages. 2005. (Subseries LNAI).

Vol. 3602: R. Eigenmann, Z. Li, S.P. Midkiff (Eds.), Languages and Compilers for High Performance Computing. IX, 486 pages. 2005.

Vol. 3598: H. Murakami, H. Nakashima, H. Tokuda, M. Yasumura, Ubiquitous Computing Systems. XIII, 275 pages. 2005.

Vol. 3597: S. Shimojo, S. Ichii, T.W. Ling, K.-H. Song (Eds.), Web and Communication Technologies and Internet-Related Social Issues - HSI 2005. XIX, 368 pages. 2005.

Vol. 3596: F. Dau, M.-L. Mugnier, G. Stumme (Eds.), Conceptual Structures: Common Semantics for Sharing Knowledge. XI, 467 pages. 2005. (Subseries LNAI).

Vol. 3595: L. Wang (Ed.), Computing and Combinatorics. XVI, 995 pages. 2005.

Vol. 3594: J.C. Setubal, S. Verjovski-Almeida (Eds.), Advances in Bioinformatics and Computational Biology. XIV, 258 pages. 2005. (Subseries LNBI).

Vol. 3587: P. Perner, A. Imiya (Eds.), Machine Learning and Data Mining in Pattern Recognition. XVII, 695 pages. 2005. (Subseries LNAI).

Vol. 3586: A.P. Black (Ed.), ECOOP 2005 - Object-Oriented Programming. XVII, 631 pages. 2005.

Vol. 3584: X. Li, S. Wang, Z.Y. Dong (Eds.), Advanced Data Mining and Applications. XIX, 835 pages. 2005. (Subseries LNAI).

Vol. 3583: R.W. H. Lau, Q. Li, R. Cheung, W. Liu (Eds.), Advances in Web-Based Learning – ICWL 2005. XIV, 420 pages. 2005.

Vol. 3582: J. Fitzgerald, I.J. Hayes, A. Tarlecki (Eds.), FM 2005: Formal Methods. XIV, 558 pages. 2005.

Vol. 3581: S. Miksch, J. Hunter, E. Keravnou (Eds.), Artificial Intelligence in Medicine. XVII, 547 pages. 2005. (Subseries LNAI).

Vol. 3580: L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), Automata, Languages and Programming. XXV, 1477 pages. 2005.

Vol. 3579: D. Lowe, M. Gaedke (Eds.), Web Engineering. XXII, 633 pages. 2005.

Vol. 3578: M. Gallagher, J. Hogan, F. Maire (Eds.), Intelligent Data Engineering and Automated Learning - IDEAL 2005. XVI, 599 pages. 2005.

Vol. 3577: R. Falcone, S. Barber, J. Sabater-Mir, M.P. Singh (Eds.), Trusting Agents for Trusting Electronic Societies. VIII, 235 pages. 2005. (Subseries LNAI).

Vol. 3576: K. Etessami, S.K. Rajamani (Eds.), Computer Aided Verification. XV, 564 pages. 2005.

Vol. 3575: S. Wermter, G. Palm, M. Elshaw (Eds.), Biomimetic Neural Learning for Intelligent Robots. IX, 383 pages. 2005. (Subseries LNAI).

Vol. 3574: C. Boyd, J.M. González Nieto (Eds.), Information Security and Privacy. XIII, 586 pages. 2005.

Vol. 3573: S. Etalle (Ed.), Logic Based Program Synthesis and Transformation. VIII, 279 pages. 2005.

Vol. 3572: C. De Felice, A. Restivo (Eds.), Developments in Language Theory. XI, 409 pages. 2005.

Vol. 3571: L. Godo (Ed.), Symbolic and Quantitative Approaches to Reasoning with Uncertainty. XVI, 1028 pages. 2005. (Subseries LNAI).

Vol. 3570: A. S. Patrick, M. Yung (Eds.), Financial Cryptography and Data Security. XII, 376 pages. 2005.

Vol. 3569: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing. XII, 492 pages. 2005.

Vol. 3568: W.-K. Leow, M.S. Lew, T.-S. Chua, W.-Y. Ma, L. Chaisorn, E.M. Bakker (Eds.), Image and Video Retrieval. XVII, 672 pages. 2005.

Vol. 3567: M. Jackson, D. Nelson, S. Stirk (Eds.), Database: Enterprise, Skills and Innovation. XII, 185 pages. 2005.

Vol. 3566: J.-P. Banâtre, P. Fradet, J.-L. Giavitto, O. Michel (Eds.), Unconventional Programming Paradigms. XI, 367 pages. 2005.

Vol. 3565: G.E. Christensen, M. Sonka (Eds.), Information Processing in Medical Imaging. XXI, 777 pages. 2005.

Vol. 3564: N. Eisinger, J. Małuszyński (Eds.), Reasoning Web. IX, 319 pages. 2005.

Vol. 3562: J. Mira, J.R. Álvarez (Eds.), Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Part II. XXIV, 636 pages. 2005.

Vol. 3561: J. Mira, J.R. Álvarez (Eds.), Mechanisms, Symbols, and Models Underlying Cognition, Part I. XXIV, 532 pages. 2005.

Vol. 3560: V.K. Prasanna, S. Iyengar, P.G. Spirakis, M. Welsh (Eds.), Distributed Computing in Sensor Systems. XV, 423 pages. 2005.

Vol. 3559: P. Auer, R. Meir (Eds.), Learning Theory. XI, 692 pages. 2005. (Subseries LNAI).

Vol. 3558: V. Torra, Y. Narukawa, S. Miyamoto (Eds.), Modeling Decisions for Artificial Intelligence. XII, 470 pages. 2005. (Subseries LNAI).

Vol. 3557: H. Gilbert, H. Handschuh (Eds.), Fast Software Encryption. XI, 443 pages. 2005.

Vol. 3556: H. Baumeister, M. Marchesi, M. Holcombe (Eds.), Extreme Programming and Agile Processes in Software Engineering. XIV, 332 pages. 2005.

Vol. 3555: T. Vardanega, A.J. Wellings (Eds.), Reliable Software Technology – Ada-Europe 2005. XV, 273 pages. 2005.

Vol. 3554: A. Dey, B. Kokinov, D. Leake, R. Turner (Eds.), Modeling and Using Context. XIV, 572 pages. 2005. (Subseries LNAI).

Vol. 3553: T.D. Hämäläinen, A.D. Pimentel, J. Takala, S. Vassiliadis (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation. XV, 476 pages. 2005.

Vol. 3552: H. de Meer, N. Bhatti (Eds.), Quality of Service – IWQoS 2005. XVIII, 400 pages. 2005.

Vol. 3551: T. Härder, W. Lehner (Eds.), Data Management in a Connected World. XIX, 371 pages. 2005.

Vol. 3548: K. Julisch, C. Kruegel (Eds.), Intrusion and Malware Detection and Vulnerability Assessment. X, 241 pages. 2005.

Vol. 3547: F. Bomarius, S. Komi-Sirviö (Eds.), Product Focused Software Process Improvement. XIII, 588 pages. 2005.

Vol. 3546: T. Kanade, A. Jain, N.K. Ratha (Eds.), Audio- and Video-Based Biometric Person Authentication. XX, 1134 pages. 2005.

Vol. 3544: T. Higashino (Ed.), Principles of Distributed Systems. XII, 460 pages. 2005.

Vol. 3543: L. Kutvonen, N. Alonistioti (Eds.), Distributed Applications and Interoperable Systems. XI, 235 pages. 2005.

Vol. 3542: H.H. Hoos, D.G. Mitchell (Eds.), Theory and Applications of Satisfiability Testing. XIII, 393 pages. 2005.

Vol. 3541: N.C. Oza, R. Polikar, J. Kittler, F. Roli (Eds.), Multiple Classifier Systems. XII, 430 pages. 2005.

Vol. 3540: H. Kalviainen, J. Parkkinen, A. Kaarna (Eds.), Image Analysis. XXII, 1270 pages. 2005.

Vol. 3539: K. Morik, J.-F. Boulicaut, A. Siebes (Eds.), Local Pattern Detection. XI, 233 pages. 2005. (Subseries LNAI).

Vol. 3538: L. Ardissono, P. Brna, A. Mitrovic (Eds.), User Modeling 2005. XVI, 533 pages. 2005. (Subseries LNAI).

Vol. 3537: A. Apostolico, M. Crochemore, K. Park (Eds.), Combinatorial Pattern Matching. XI, 444 pages. 2005.

Vol. 3536: G. Ciardo, P. Darondeau (Eds.), Applications and Theory of Petri Nets 2005. XI, 470 pages. 2005.

Vol. 3535: M. Steffen, G. Zavattaro (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 323 pages. 2005.

Vol. 3534: S. Spaccapietra, E. Zimányi (Eds.), Journal on Data Semantics III. XI, 213 pages. 2005.

Vol. 3533: M. Ali, F. Esposito (Eds.), Innovations in Applied Artificial Intelligence. XX, 858 pages. 2005. (Subseries LNAI).

Vol. 3532: A. Gómez-Pérez, J. Euzenat (Eds.), The Semantic Web: Research and Applications. XV, 728 pages. 2005.

Vol. 3531: J. Ioannidis, A. Keromytis, M. Yung (Eds.), Applied Cryptography and Network Security. XI, 530 pages. 2005.

Vol. 3530: A. Prinz, R. Reed, J. Reed (Eds.), SDL 2005: Model Driven. XI, 361 pages. 2005.

Vol. 3528: P.S. Szczepaniak, J. Kacprzyk, A. Niewiadomski (Eds.), Advances in Web Intelligence. XVII, 513 pages. 2005. (Subseries LNAI).

Vol. 3527: R. Morrison, F. Oquendo (Eds.), Software Architecture. XII, 263 pages. 2005.

Vol. 3526: S. B. Cooper, B. Löwe, L. Torenvliet (Eds.), New Computational Paradigms. XVII, 574 pages. 2005.

Vol. 3525: A.E. Abdallah, C.B. Jones, J.W. Sanders (Eds.), Communicating Sequential Processes. XIV, 321 pages. 2005.

Vol. 3524: R. Barták, M. Milano (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. XI, 320 pages. 2005.

Vol. 3523: J.S. Marques, N. Pérez de la Blanca, P. Pina (Eds.), Pattern Recognition and Image Analysis, Part II. XXVI, 733 pages. 2005.

Vol. 3522: J.S. Marques, N. Pérez de la Blanca, P. Pina (Eds.), Pattern Recognition and Image Analysis, Part I. XXVI, 703 pages. 2005.

Vol. 3521: N. Megiddo, Y. Xu, B. Zhu (Eds.), Algorithmic Applications in Management. XIII, 484 pages. 2005.

Vol. 3520: O. Pastor, J. Falcão e Cunha (Eds.), Advanced Information Systems Engineering. XVI, 584 pages. 2005.

Vol. 3519: H. Li, P. J. Olver, G. Sommer (Eds.), Computer Algebra and Geometric Algebra with Applications. IX, 449 pages. 2005.

Vol. 3518: T.B. Ho, D. Cheung, H. Liu (Eds.), Advances in Knowledge Discovery and Data Mining. XXI, 864 pages. 2005. (Subseries LNAI).

Vol. 3517: H.S. Baird, D.P. Lopresti (Eds.), Human Interactive Proofs. IX, 143 pages. 2005.

Vol. 3516: V.S. Sunderam, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science – ICCS 2005, Part III. LXIII, 1143 pages. 2005.

Vol. 3515: V.S. Sunderam, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science – ICCS 2005, Part II. LXIII, 1101 pages. 2005.

Vol. 3514: V.S. Sunderam, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science – ICCS 2005, Part I. LXIII, 1089 pages. 2005.

Table of Contents

Experiences in Using Cetus for Source-to-Source Transformations*

Troy A. Johnson, Sang-Ik Lee, Long Fei, Ayon Basumallik, Gautam Upadhyaya, Rudolf Eigenmann, and Samuel P. Midkiff

School of Electrical & Computer Engineering,
Purdue University, West Lafayette IN 47906, USA
{troyj, sangik, lfei, basumall, gupadhya, eigenman, smidkiff}@ecn.purdue.edu
http://www.ece.purdue.edu/ParaMount

Abstract. Cetus is a compiler infrastructure for the source-to-source transformation of programs. Since its creation nearly three years ago, it has grown to over 12,000 lines of Java code, been made available publically on the web, and become a basis for several research projects. We discuss our experience using Cetus for a selection of these research projects. The focus of this paper is not the projects themselves, but rather how Cetus made these projects possible, how the needs of these projects influenced the development of Cetus, and the solutions we applied to problems we encountered with the infrastructure. We believe the research community can benefit from such a discussion, as shown by the strong interest in the mini-workshop on compiler research infrastructures where some of this information was first presented.

1 Introduction

Parallelizing compiler technology is most mature for the Fortran 77 language [1,3,12,13,16]. The simplicity of the language without pointers or user-defined types makes it easy to analyze and to develop many advanced compiler passes. By contrast, parallelization technology for modern languages, such as Java, C++, or even C, is still in its infancy. When trying to engage in such research, we were faced with a serious challenge. We were unable to find a parallelizing compiler infrastructure that supported interprocedural analysis, exhibited state-of-the-art software engineering techniques to help shorten development time, and allowed us to compile large, realistic applications. We feel these properties are of paramount importance because they enable a compiler writer to develop "production strength" passes. Production strength passes, in turn, can work in the context of the most up-to-date compiler technology and lead to compiler research that can be evaluated with full suites of realistic applications. The lack of such thorough evaluations in many current research papers has been

* This material is based upon work supported in part by the National Science Foundation under Grant No. 9703180, 9975275, 9986020, and 9974976.

R. Eigenmann et al. (Eds.): LCPC 2004, LNCS 3602, pp. 1–14, 2005.

observed and criticized by many. The availability of an easy-to-use compiler infrastructure would help improve this situation significantly. Hence, continuous research and development in this area are among the most important tasks of the compiler community.

Cetus was created with those needs in mind. It supports analyses and transformations at the source level; other infrastructures are more appropriate for instruction-level compiler research. Cetus is composed of over 10,000 lines of Java code that implements the Cetus intermediate representation (IR), over 1,500 lines of code that implements source transformations, a C parser using Antlr, and standalone C and C++ Bison parsers that have yet to be integrated completely into Cetus. The Cetus IR is the product of three graduate students working part-time over two years. Several others have contributed analysis and transformation passes, as well as used Cetus for their own research projects. We discuss these projects in this paper from the perspective of how Cetus made these projects possible, how the needs of these projects influenced the development of Cetus, and the solutions we applied to problems we encountered with the infrastructure. We believe the research community can benefit from such a discussion, as shown by the strong interest in the mini-workshop on compiler research infrastructures where some of this information was first presented.

Section 2 briefly covers the Cetus IR. In Section 3, we cover basic analysis, transformation, and instrumentation passes. Section 4 contains five case studies of more complex passes. Section 5 discusses the effects of user-feedback on the project. Finally, Section 6 concludes.

2 Cetus Intermediate Representation

For the design of the IR we chose an abstract representation, implemented in the form of a class hierarchy and accessed through the class member functions. We consider a strong separation between the implementation and the interface to be very important. In this way, a change to the implementation may be done while maintaining the API for its users. It also permits passes to be written before the IR implementation is ready. These concepts had already proved their value in the implementation of the Polaris infrastructure [2], which served as an important example for the Cetus design. Polaris was rewritten three to four times over its lifetime while keeping the interface, and hence all compilation passes, nearly unmodified [5]. Cetus has a similar design, shown in Figure 1, where the high-level interface insulates the pass writer from changes in the base.

Our design goal was a simple IR class hierarchy easily understood by users. It should also be easy to maintain, while being rich enough to enable future extension without major modification. The basic building blocks of a program are the *translation units*, which represent the content of a source file, and *procedures*, which represent individual functions. Procedures include a list of simple or compound statements, representing the program control flow in a hierarchical way. That is, compound statements, such as *IF*-constructs and *FOR*-loops include inner (simple or compound) statements, representing *then* and *else* blocks

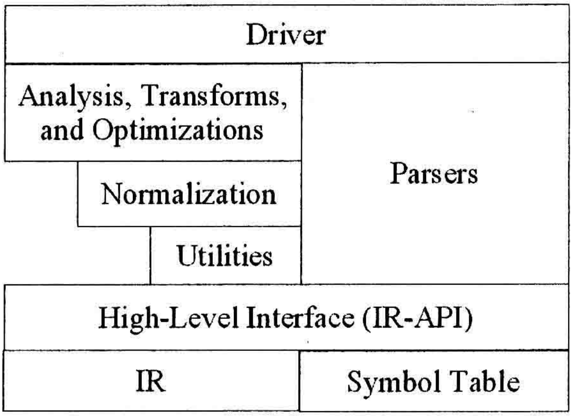

Fig. 1. Cetus components and interfaces: Components of Cetus only call methods of the components beneath them. The driver interprets command-line arguments and initiates the appropriate parser for the input language, which in turn uses the high-level interface to build the IR. The driver then initiates analysis and transformation passes. Normalization passes and utilities are provided to perform complex operations that are useful to multiple passes. The interface functions are kept lean and generally provide only a single way of performing IR manipulations

or loop bodies, respectively. *Expressions* represent the operations being done on variables, including the assignments to variables.

Cetus' IR contrasts with the Polaris Fortran translator's IR in that it uses a hierarchical statement structure. The Cetus IR directly reflects the block structure of a program. Polaris lists the statements of each procedure in a flat way, with a reference to the outer statement being the only way for determining the block structure. There are also important differences in the representation of expressions, which further reflects the differences between C and Fortran. The Polaris IR includes assignment statements, whereas Cetus represents assignments in the form of expressions. This corresponds to the C language's feature to include assignment side effects in any expression.

The IR is structured such that the original source program can be reproduced, but this is where source-to-source translators face an intrinsic dilemma. Keeping the IR and output similar to the input will make it easy for the user to recognize the transformations applied by the compiler. On the other hand, keeping the IR language-independent leads to a simpler compiler architecture, but may make it impossible to reproduce the original source code as output. In Cetus, the concept of statements and expressions are closely related to the syntax of the C language, facilitating easy source-to-source translation. The correspondence between syntax and IR is shown in Figure 2. However, the drawback is increased complexity for pass writers (since they must think in terms of C syntax) and limited extensibility of Cetus to additional languages. That problem is mitigated by the provision of several interfaces that represent generic control constructs. Generic

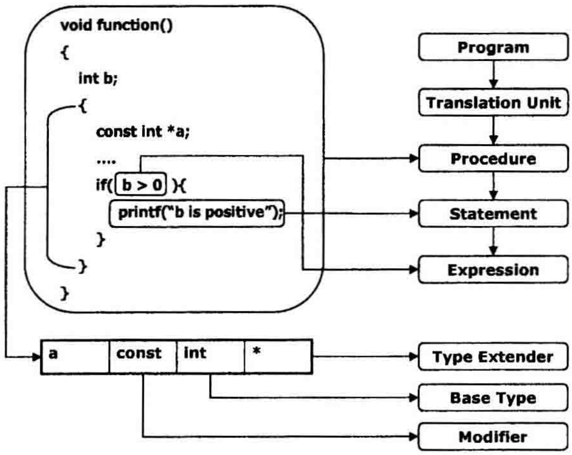

Fig. 2. A program fragment and its IR in Cetus. IR relationships are similar to the program structure and a symbol table is associated with each block scope

passes can be written using the abstract interface, while more language-specific passes can use the derived classes. For example, the classes that represent for-loops and while-loops both implement a loop interface. A pass that manipulates loops may be written using the generic loop interface if the exact type of loop is not important.

The high-level interface, or IR-API, is the interface presented to compiler writers. In general the IR-API is kept minimal and free of redundant functionality, so as to make it easy to learn about its basic operation and easy to debug. IR-API calls expect the IR to be in a consistent state upon entry and ensure the state is consistent upon their return. Cetus also provides a utility package, that offers convenience to pass writers. The utility package provides additional functions, where needed by more than a single compiler pass. Obviously, this criterion will depend on the passes that will be written in the future. Hence, the utilities will evolve, while we expect the base to remain stable. The utility functions operate using only the IR-API.

2.1 Navigating the IR

Traversing the IR is a fundamental operation that will be used by every compiler pass. Therefore, it is important that traversals be easy to perform and require little code. Cetus provides an abstract IRIterator class that implements the standard Java Iterator interface. The classes BreadthFirstIterator, DepthFirstIterator, and FlatIterator are all derived from IRIterator. The constructor for each of these classes accepts as its only parameter a Traversable object which defines the root of the traversal. Traversable is an interface that ensures any implementing class can act as a node of a tree by providing methods to access its parent and children. A design alternative here was to have every class provide a getIterator method instead of passing the root object to an iterator constructor, but that required adding an implementation of getIterator to every class, and was rejected.[1] The DepthFirstIterator visits statements and expressions sequentially in program order without regard to block scope. The BreadthFirstIterator visits all children of an object before visiting the children's children; i.e., block scope is respected with outer objects visited first. The FlatIterator does not visit the root of the traversal and instead visits the root's children sequentially without visiting the children's children.

In addition to providing a `next` method, as all Iterators must, an IRIterator provides `next(Class)`, `next(Set)`, and `nextExcept(Set)` to allow the caller to specify that only objects of a certain class, or that belong or do not belong to a particular set of classes, are of interest. When these methods were first introduced, we were able to rewrite older Cetus passes using considerably fewer lines of code. Figure 3 shows the usefulness of these methods.

```
/* 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) {
}
```

Fig. 3. Using iterators to find loops within a procedure. Outer loops are discovered first

2.2 Type System and Symbol Table

Modern programming languages provide rich type systems. In order to keep the Cetus type system flexible, we divided the elements of a type into three concepts: base types, extenders, and modifiers. A complete type is described by a combination of these three elements. Base types include built-in primitive types, which have a predefined meaning in programming languages, and user-defined types. User-defined types are new types introduced into the program by providing the layout of the structure and the semantics. These include typedef, struct, union, and enum types in C. Base types are often combined with type extenders. Examples of type extenders are arrays, pointers, and functions. The last concept is modifiers which express an attribute of a type, such as const and volatile in C. They can decorate any part of the type definition. Types

[1] The decision was primarily due to Java's lack of multiple inheritance, since in most cases inheritance had already been used.

are understood by decoding the description one element at a time, which is a sequential job in nature. We use a list structure to hold type information so that types can be understood easily by looking at the elements in the list one at a time.

Another important concept is a symbol, which represents the declaration of a variable in the program. Symbols are not part of the IR tree and reside in symbol tables. Our concept of a symbol table is a mapping from a variable name to its point of declaration, which is located in a certain scope and has all of the type information. As a result, scope always must be considered when dealing with symbols. In C, a block structure defines a scope. Therefore, structs in C are also scopes and their members are represented as local symbols within that scope. A compiler may use one large symbol table with hashing to locate symbols [4]. In Cetus, since source transformations can move, add, or remove scopes, we use distributed symbol tables where each scope has a separate physical symbol table. The logical symbol table for a scope includes its physical symbol table and the physical symbol tables of the enclosing scopes, with inner declarations hiding outer declarations. There are certain drawbacks to this approach, namely the need to search through the full hierarchy of symbol tables to reach a global symbol [6], but we find it to be convenient. For example, all the declarations in a scope can be manipulated as a group simply by manipulating that scope's symbol table. It is especially convenient in allowing Cetus to support object-oriented languages, where classes and namespaces may introduce numerous scopes whose relationships can be expressed through the symbol table hierarchy.

3 Capabilities for Writing Passes

Cetus has a number of features that are useful to pass writers. Classes that support program analysis, normalization, and modification are discussed below.

3.1 Analysis

Call Graph. Cetus provides a CallGraph class that creates a call graph from a Program object. The call graph maps callers to callees as well as callees to callers. A pass can query the call graph to determine if a procedure is a leaf of the call graph or if a procedure is recursive.

Control-Flow Graph. Cetus provides a ControlFlowGraph class, which creates a control-flow graph from a Program object. The graph represents the structure of each procedure in terms of its basic blocks connected by control-flow edges.

3.2 Normalization

Single Return. Compiler passes often become simpler if they can assume that each procedure has a single exit point. A procedure with multiple return statements complicates such actions as inserting code that should execute just prior