Samuel P. Midkiff · José E. Moreira
Manish Gupta · Siddhartha Chatterjee
Jeanne Ferrante · Jan Prins
William Pugh · Chau-Wen Tseng (Eds.)

LNCS 2017

Languages and Compilers for Parallel Computing

13th International Workshop, LCPC 2000
Yorktown Heights, NY, USA, August 2000
Revised Papers

Springer

Samuel P. Midkiff José E. Moreira Manish Gupta
Siddhartha Chatterjee Jeanne Ferrante Jan Prins
William Pugh Chau-Wen Tseng (Eds.)

Languages and Compilers for Parallel Computing

13th International Workshop, LCPC 2000
Yorktown Heights, NY, USA, August 10-12, 2000
Revised Papers

Springer

Volume Editors

Samuel P. Midkiff
José E. Moreira
Manish Gupta
Siddhartha Chatterjee
IBM T.J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598, USA
E-mail: {smidkiff,jmoreira,mgupta,sc}@us.ibm.com

Jeanne Ferrante
University of California at San Diego, Computer Science and Engineering
9500 Gilman Drive, La Jolla, CA 92093-0114, USA
E-mail: ferrante@cs.ucsd.edu

Jan Prins
University of North Carolina, Department of Computer Science
Chapel Hill, NC 27599-3175, USA
E-mail: prins@unc.edu

William Pugh
Chau-Wen Tseng
University of Maryland, Department of Computer Science
College Park, MD 20742, USA
E-mail: {pugh,tseng}@cs.umd.edu

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Languages and compilers for parallel computing : 13th international workshop ;
revised papers / LCPC 2000, Yorktown Heights, NY, USA, August 10 - 12, 2000.
Samuel P. Midkiff ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ;
Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2017)
ISBN 3-540-42862-3

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

ISSN 0302-9743
ISBN 3-540-42862-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 10782214 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 2017

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Foreword

This volume contains the papers presented at the 13th International Workshop on Languages and Compilers for Parallel Computing. It also contains extended abstracts of submissions that were accepted as posters. The workshop was held at the IBM T. J. Watson Research Center in Yorktown Heights, New York. As in previous years, the workshop focused on issues in optimizing compilers, languages, and software environments for high performance computing. This continues a trend in which languages, compilers, and software environments for high performance computing, and not strictly parallel computing, has been the organizing topic. As in past years, participants came from Asia, North America, and Europe.

This workshop reflected the work of many people. In particular, the members of the steering committee, David Padua, Alex Nicolau, Utpal Banerjee, and David Gelernter, have been instrumental in maintaining the focus and quality of the workshop since it was first held in 1988 in Urbana-Champaign. The assistance of the other members of the program committee – Larry Carter, Sid Chatterjee, Jeanne Ferrante, Jans Prins, Bill Pugh, and Chau-wen Tseng – was crucial. The infrastructure at the IBM T. J. Watson Research Center provided trouble-free logistical support. The IBM T. J. Watson Research Center also provided financial support by underwriting much of the expense of the workshop. Appreciation must also be extended to Marc Snir and Pratap Pattnaik of the IBM T. J. Watson Research Center for their support.

Finally, we would like to thank the referees who spent countless hours assisting the program committee members in evaluating the quality of the submissions: Scott B. Baden, Jean-Francois Collard, Val Donaldson, Rudolf Eigenmann, Stephen Fink, Kang Su Gatlin, Michael Hind, Francois Irigoin, Pramod G. Joisha, Gabriele Keller, Wolf Pfannenstiel, Lawrence Rauchweger, Martin Simons, D. B. Skillicorn, Hong Tang, and Hao Yu.

January 2001

Manish Gupta
Sam Midkiff
José Moreira

Organization

The 13th annual International Workshop on Languages and Compilers for High Performance Computing (LCPC 2000) was organized and sponsored by the IBM T. J Watson Research Center, Yorktown Heights, New York

Steering Committee

Utpal Banerjee	*Intel Corporation*
David Gelernter	*Yale University*
Alex Nicolau	*University of California at Irvine*
David A. Padua	*University of Illinois at Urbana-Champaign*

Program Committee

Siddhartha Chatterjee	*University of North Carolina at Chapel Hill*
Larry Carter	*University of California at San Diego*
Jeanne Ferrante	*University of California at San Diego*
Manish Gupta	*IBM T. J. Watson Research Center*
Sam Midkiff	*IBM T. J. Watson Research Center*
José Moreira	*IBM T. J. Watson Research Center*
Jans Prins	*University of North Carolina at Chapel Hill*
Bill Pugh	*University of Maryland*
Chau-Wen Tseng	*University of Maryland*

Sponsoring Institutions

The IBM T. J. Watson Research Center, Yorktown Heights, New York

Author Index

Lecture Notes in Computer Science

For information about Vols. 1–2161
please contact your bookseller or Springer-Verlag

Vol. 2017: S.P. Midkiff, J.E. Moreira, M. Gupta, S. Chatterjee, J. Ferrante, J. Prins, W. Pugh, C.-W. Tseng (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 2000. IX, 383 pages. 2001.

Vol. 2162: Ç. K. Koç, D. Naccache, C. Paar (Eds.), Cryptographic Hardware and Embedded Systems – CHES 2001. Proceedings, 2001. XIV, 411 pages. 2001.

Vol. 2163: P. Constantopoulos, I.T. Sølvberg (Eds.), Research and Advanced Technology for Digital Libraries. Proceedings, 2001. XII, 462 pages. 2001.

Vol. 2164: S. Pierre, R. Glitho (Eds.), Mobile Agents for Telecommunication Applications. Proceedings, 2001. XI, 292 pages. 2001.

Vol. 2165: L. de Alfaro, S. Gilmore (Eds.), Process Algebra and Probabilistic Methods. Proceedings, 2001. XII, 217 pages. 2001.

Vol. 2166: V. Matoušek, P. Mautner, R. Mouček, K. Taušer (Eds.), Text, Speech and Dialogue. Proceedings, 2001. XIII, 452 pages. 2001. (Subseries LNAI).

Vol. 2167: L. De Raedt, P. Flach (Eds.), Machine Learning: ECML 2001. Proceedings, 2001. XVII, 618 pages. 2001. (Subseries LNAI).

Vol. 2168: L. De Raedt, A. Siebes (Eds.), Principles of Data Mining and Knowledge Discovery. Proceedings, 2001. XVII, 510 pages. 2001. (Subseries LNAI).

Vol. 2169: M. Jaedicke, New Concepts for Parallel Object-Relational Query Processing. XI, 161 pages. 2001.

Vol. 2170: S. Palazzo (Ed.), Evolutionary Trends of the Internet. Proceedings, 2001. XIII, 722 pages. 2001.

Vol. 2171: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design. VII, 397 pages. 2001.

Vol. 2172: C. Batini, F. Giunchiglia, P. Giorgini, M. Mecella (Eds.), Cooperative Information Systems. Proceedings, 2001. XI, 450 pages. 2001.

Vol. 2173: T. Eiter, W. Faber, M. Truszczynski (Eds.), Logic Programming and Nonmonotonic Reasoning. Proceedings, 2001. XI, 444 pages. 2001. (Subseries LNAI).

Vol. 2174: F. Baader, G. Brewka, T. Eiter (Eds.), KI 2001: Advances in Artificial Intelligence. Proceedings, 2001. XIII, 471 pages. 2001. (Subseries LNAI).

Vol. 2175: F. Esposito (Ed.), AI*IA 2001: Advances in Artificial Intelligence. Proceedings, 2001. XII, 396 pages. 2001. (Subseries LNAI).

Vol. 2176: K.-D. Althoff, R.L. Feldmann, W. Müller (Eds.), Advances in Learning Software Organizations. Proceedings, 2001. XI, 241 pages. 2001.

Vol. 2177: G. Butler, S. Jarzabek (Eds.), Generative and Component-Based Software Engineering. Proceedings, 2001. X, 203 pages. 2001.

Vol. 2178: R. Moreno-Díaz, B. Buchberger, J.-L. Freire (Eds.), Computer Aided Systems Theory – EUROCAST 2001. Proceedings, 2001. XI, 670 pages. 2001.

Vol. 2180: J. Welch (Ed.), Distributed Computing. Proceedings, 2001. X, 343 pages. 2001.

Vol. 2181: C. Y. Westort (Ed.), Digital Earth Moving. Proceedings, 2001. XII, 117 pages. 2001.

Vol. 2182: M. Klusch, F. Zambonelli (Eds.), Cooperative Information Agents V. Proceedings, 2001. XII, 288 pages. 2001. (Subseries LNAI).

Vol. 2183: R. Kahle, P. Schroeder-Heister, R. Stärk (Eds.), Proof Theory in Computer Science. Proceedings, 2001. IX, 239 pages. 2001.

Vol. 2184: M. Tucci (Ed.), Multimedia Databases and Image Communication. Proceedings, 2001. X, 225 pages. 2001.

Vol. 2185: M. Gogolla, C. Kobryn (Eds.), «UML» 2001 – The Unified Modeling Language. Proceedings, 2001. XIV, 510 pages. 2001.

Vol. 2186: J. Bosch (Ed.), Generative and Component-Based Software Engineering. Proceedings, 2001. VIII, 177 pages. 2001.

Vol. 2187: U. Voges (Ed.), Computer Safety, Reliability and Security. Proceedings, 2001. XVI, 249 pages. 2001.

Vol. 2188: F. Bomarius, S. Komi-Sirviö (Eds.), Product Focused Software Process Improvement. Proceedings, 2001. XI, 382 pages. 2001.

Vol. 2189: F. Hoffmann, D.J. Hand, N. Adams, D. Fisher, G. Guimaraes (Eds.), Advances in Intelligent Data Analysis. Proceedings, 2001. XII, 384 pages. 2001.

Vol. 2190: A. de Antonio, R. Aylett, D. Ballin (Eds.), Intelligent Virtual Agents. Proceedings, 2001. VIII, 245 pages. 2001. (Subseries LNAI).

Vol. 2191: B. Radig, S. Florczyk (Eds.), Pattern Recognition. Proceedings, 2001. XVI, 452 pages. 2001.

Vol. 2192: A. Yonezawa, S. Matsuoka (Eds.), Metalevel Architectures and Separation of Crosscutting Concerns. Proceedings, 2001. XI, 283 pages. 2001.

Vol. 2193: F. Casati, D. Georgakopoulos, M.-C. Shan (Eds.), Technologies for E-Services. Proceedings, 2001. X, 213 pages. 2001.

Vol. 2194: A.K. Datta, T. Herman (Eds.), Self-Stabilizing Systems. Proceedings, 2001. VII, 229 pages. 2001.

Vol. 2195: H.-Y. Shum, M. Liao, S.-F. Chang (Eds.), Advances in Multimedia Information Processing – PCM 2001. Proceedings, 2001. XX, 1149 pages. 2001.

Vol. 2196: W. Taha (Ed.), Semantics, Applications, and Implementation of Program Generation. Proceedings, 2001. X, 219 pages. 2001.

Vol. 2197: O. Balet, G. Subsol, P. Torguet (Eds.), Virtual Storytelling. Proceedings, 2001. XI, 213 pages. 2001.

Vol. 2198: N. Zhong, Y. Yao, J. Liu, S. Ohsuga (Eds.), Web Intelligence: Research and Development. Proceedings, 2001. XVI, 615 pages. 2001. (Subseries LNAI).

Vol. 2199: J. Crespo, V. Maojo, F. Martin (Eds.), Medical Data Analysis. Proceedings, 2001. X, 311 pages. 2001.

Vol. 2200: G.I. Davida, Y. Frankel (Eds.), Information Security. Proceedings, 2001. XIII, 554 pages. 2001.

Vol. 2201: G.D. Abowd, B. Brumitt, S. Shafer (Eds.), Ubicomp 2001: Ubiquitous Computing. Proceedings, 2001. XIII, 372 pages. 2001.

Vol. 2202: A. Restivo, S. Ronchi Della Rocca, L. Roversi (Eds.), Theoretical Computer Science. Proceedings, 2001. XI, 440 pages. 2001.

Vol. 2204: A. Brandstädt, V.B. Le (Eds.), Graph-Theoretic Concepts in Computer Science. Proceedings, 2001. X, 329 pages. 2001.

Vol. 2205: D.R. Montello (Ed.), Spatial Information Theory. Proceedings, 2001. XIV, 503 pages. 2001.

Vol. 2206: B. Reusch (Ed.), Computational Intelligence. Proceedings, 2001. XVII, 1003 pages. 2001.

Vol. 2207: I.W. Marshall, S. Nettles, N. Wakamiya (Eds.), Active Networks. Proceedings, 2001. IX, 165 pages. 2001.

Vol. 2208: W.J. Niessen, M.A. Viergever (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001. Proceedings, 2001. XXXV, 1446 pages. 2001.

Vol. 2209: W. Jonker (Ed.), Databases in Telecommunications II. Proceedings, 2001. VII, 179 pages. 2001.

Vol. 2210: Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, M. Yasunaga (Eds.), Evolvable Systems: From Biology to Hardware. Proceedings, 2001. XI, 341 pages. 2001.

Vol. 2211: T.A. Henzinger, C.M. Kirsch (Eds.), Embedded Software. Proceedings, 2001. IX, 504 pages. 2001.

Vol. 2212: W. Lee, L. Mé, A. Wespi (Eds.), Recent Advances in Intrusion Detection. Proceedings, 2001. X, 205 pages. 2001.

Vol. 2213: M.J. van Sinderen, L.J.M. Nieuwenhuis (Eds.), Protocols for Multimedia Systems. Proceedings, 2001. XII, 239 pages. 2001.

Vol. 2214: O. Boldt, H. Jürgensen (Eds.), Automata Implementation. Proceedings, 1999. VIII, 183 pages. 2001.

Vol. 2215: N. Kobayashi, B.C. Pierce (Eds.), Theoretical Aspects of Computer Software. Proceedings, 2001. XV, 561 pages. 2001.

Vol. 2216: E.S. Al-Shaer, G. Pacifici (Eds.), Management of Multimedia on the Internet. Proceedings, 2001. XIV, 373 pages. 2001.

Vol. 2217: T. Gomi (Ed.), Evolutionary Robotics. Proceedings, 2001. XI, 139 pages. 2001.

Vol. 2218: R. Guerraoui (Ed.), Middleware 2001. Proceedings, 2001. XIII, 395 pages. 2001.

Vol. 2220: C. Johnson (Ed.), Interactive Systems. Proceedings, 2001. XII, 219 pages. 2001.

Vol. 2221: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing. Proceedings, 2001. VII, 207 pages. 2001.

Vol. 2223: P. Eades, T. Takaoka (Eds.), Algorithms and Computation. Proceedings, 2001. XIV, 780 pages. 2001.

Vol. 2224: H.S. Kunii, S. Jajodia, A. Sølvberg (Eds.), Conceptual Modeling – ER 2001. Proceedings, 2001. XIX, 614 pages. 2001.

Vol. 2225: N. Abe, R. Khardon, T. Zeugmann (Eds.), Algorithmic Learning Theory. Proceedings, 2001. XI, 379 pages. 2001. (Subseries LNAI).

Vol. 2226: K.P. Jantke, A. Shinohara (Eds.), Discovery Science. Proceedings, 2001. XII, 494 pages. 2001. (Subseries LNAI).

Vol. 2227: S. Boztaş, I.E. Shparlinski (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings, 2001. XII, 398 pages. 2001.

Vol. 2228: B. Monien, V.K. Prasanna, S. Vajapeyam (Eds.), High Performance Computing – HiPC 2001. Proceedings, 2001. XVIII, 438 pages. 2001.

Vol. 2229: S. Qing, T. Okamoto, J. Zhou (Eds.), Information and Communications Security. Proceedings, 2001. XIV, 504 pages. 2001.

Vol. 2230: T. Katila, I.E. Magnin, P. Clarysse, J. Montagnat, J. Nenonen (Eds.), Functional Imaging and Modeling of the Heart. Proceedings, 2001. XI, 158 pages. 2001.

Vol. 2232: L. Fiege, G. Mühl, U. Wilhelm (Eds.), Electronic Commerce. Proceedings, 2001. X, 233 pages. 2001.

Vol. 2233: J. Crowcroft, M. Hofmann (Eds.), Networked Group Communication. Proceedings, 2001. X, 205 pages. 2001.

Vol. 2234: L. Pacholski, P. Ružička (Eds.), SOFSEM 2001: Theory and Practice of Informatics. Proceedings, 2001. XI, 347 pages. 2001.

Vol. 2237: P. Codognet (Ed.), Logic Programming. Proceedings, 2001. XI, 365 pages. 2001.

Vol. 2239: T. Walsh (Ed.), Principles and Practice of Constraint Programming – CP 2001. Proceedings, 2001. XIV, 788 pages. 2001.

Vol. 2240: G.P. Picco (Ed.), Mobile Agents. Proceedings, 2001. XIII, 277 pages. 2001.

Vol. 2241: M. Jünger, D. Naddef (Eds.), Computational Combinatorial Optimization. IX, 305 pages. 2001.

Vol. 2242: C.A. Lee (Ed.), Grid Computing – GRID 2001. Proceedings, 2001. XII, 185 pages. 2001.

Vol. 2245: R. Hariharan, M. Mukund, V. Vinay (Eds.), FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science. Proceedings, 2001. XI, 347 pages. 2001.

Vol. 2247: C. P. Rangan, C. Ding (Eds.), Progress in Cryptology – INDOCRYPT 2001. Proceedings, 2001. XIII, 351 pages. 2001.

Vol. 2250: R. Nieuwenhuis, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings, 2001. XV, 738 pages. 2001. (Subseries LNAI).

Vol. 2248: C. Boyd (Ed.), Advances in Cryptology – ASIACRYPT 2001. Proceedings, 2001. XI, 603 pages. 2001.

Vol. 2256: M. Stumptner, D. Corbett, M. Brooks (Eds.), AI 2001: Advances in Artificial Intelligence. Proceedings, 2001. XII, 666 pages. 2001. (Subseries LNAI).

Vol. 2264: K. Steinhöfel (Ed.), Stochastic Algorithms: Foundations and Applications. Proceedings, 2001. VIII, 203 pages. 2001.

Table of Contents

Presented Papers

Posters

Accurate Shape Analysis for Recursive Data Structures*

Francisco Corbera, Rafael Asenjo, and Emilio Zapata

Dept. Computer Architecture, University of Málaga, Spain
{corbera,asenjo,ezapata}@ac.uma.es

Abstract. Automatic parallelization of codes which use dynamic data structures is still a challenge. One of the first steps in such parallelization is the automatic detection of the dynamic data structure used in the code. In this paper we describe the framework and the compiler we have implemented to capture complex data structures generated, traversed, and modified in C codes. Our method assigns a *Reduced Set of Reference Shape Graphs* (RSRSG) to each sentence to approximate the shape of the data structure after the execution of such a sentence. With the properties and operations that define the behavior of our RSRSG, the method can accurately detect complex recursive data structures such as a doubly linked list of pointers to trees where the leaves point to additional lists. Other experiments are carried out with real codes to validate the capabilities of our compiler.

1 Introduction

For complex and time-consuming applications, parallel programming is a must. Automatic parallelizing compilers are designed with the aim of dramatically reducing the time needed to develop a parallel program by generating a parallel version from a sequential code without special annotations. There are several well-known research groups involved in the development and improvement of parallel compilers, such as Polaris, PFA, Parafrase, SUIF, etc. We have noted that the detection step of current parallelizing compilers does a pretty good job when dealing with regular or numeric codes. However, they cannot manage irregular codes or symbolic ones, which are mainly based on complex data structures which use pointers in many cases. Actually, data dependence analysis is quite well known for array-based codes even when complex array access functions are present [5]. On the other hand, much less work has been done to successfully determine the data dependencies of code sections using dynamic data structures based on pointers. Nevertheless, this is a problem that cannot be avoided due to the increasing use of dynamic structures and memory pointer references.

* This work was supported by the Ministry of Education and Science (CICYT) of Spain (TIC96-1125-C03), by the European Union (BRITE-EURAM III BE95-1564), by APART: Automatic Performance Analysis: Resources and Tools, EU Esprit IV Working Group No. 29488

S.P. Midkiff et al. (Eds.): LCPC 2000, LNCS 2017, pp. 1–15, 2001.

With this motivation, our goal is to propose and implement new techniques that can be included in compilers to allow the automatic parallelization of real codes based on dynamic data structures. From this goal we have selected the shape analysis subproblem, which aims at estimating at compile time the shape the data will take at run time. Given this information, a subsequent analysis would detect whether or not certain sections of the code can be parallelized because they access independent data regions.

There are several ways this problem can be approached, but we focus in the graph-based methods in which the "storage chunks" are represented by nodes, and edges are used to represent references between them [2], [8], [9]. In a previous work [3], we combined and extended several ideas from these previous graph-based methods, for example, allowing more than a summary node per graph among other extensions. However, we keep the restriction of one graph per sentence in the code. This way, since each sentence of the code can be reached after following several paths in the control flow, the associated graph should approximate all the possible memory configurations arising after the execution of this sentence. This restriction leads to memory and time saving, but at the same time it significantly reduces the accuracy of the method. In this work, we have changed our previous direction by selecting a tradeoff solution: we consider several graphs with more than a summary node, while fulfilling some rules to avoid an explosion in the number of graphs and nodes in each graph.

Among the first relevant studies which allowed several graphs were those developed by Jones et al. [7] and Horwitz et al. [6]. These approaches are based on a "k-limited" approximation in which all nodes beyond a k selectors path are joined in a summary node. The main drawback to these methods is that the node analysis beyond the "k-limit" is very inexact and therefore they are unable to capture complex data structures. A more recent work that also allows several graphs and summary nodes is the one presented by Sagiv et al.[10]. They propose a parametric framework based on a 3-valued logic. To describe the memory configuration they use 3-valued structures defined by several predicates. These predicates determine the accuracy of the method. As far as we know the currently proposed predicates do not suffice to deal with the complex data structures that we handle in this paper.

With this in mind, our proposal is based on approximating all the possible memory configurations that can arise after the execution of a sentence by a set of graphs: the *Reduced Set of Reference Shape Graphs* (RSRSG). We see that each RSRSG is a collection of *Reference Shape Graphs* (RSG) each one containing several non-compatible nodes. Finally, each node represents one or several memory locations. Compatible nodes are "summarized" into a single one. Two nodes are compatible if they share the same reference properties. With this framework we can achieve accurate results without excessive compilation time. Besides this, we cover situations that were previously unsolved, such as detection of complex structures (lists of trees, lists of lists, etc.) and structure permutation, as we will see in this article.

The rest of the paper is organized as follows. Section 2 briefly describes the whole framework, introducing the key ideas of the method and presenting the data structure example that will help in understanding node properties and operations with graphs. These properties are described in Sect. 3 where we show how the RSG can accurately approximate a memory configuration. The analysis method have been implemented in a compiler which is experimentally validated, in Sect. 4, by analyzing several C codes based on complex data structures. Finally, we summarize the main contributions and future work in Sect. 5.

2 Method Overview

Basically, our method is based on approximating all possible memory configurations that can appear after the execution of a sentence in the code. Note that due to the control flow of the program, a sentence could be reached by following several paths in the control flow. Each "control path" has an associated memory configuration which is modified by each sentence in the path. Therefore, a single sentence in the code modifies all the memory configurations associated with all the control paths reaching this sentence. Each memory configuration is approximated by a graph we call *Reference Shape Graphs* (RSG). So, taking all this into account, we conclude that each sentence in the code will have a set of RSGs associated with it. This set of RSGs will describe the shape of the data structure after the execution of this sentence.

The calculation of this set of graphs is carried out by the **symbolic execution** of the program over the graphs. In this way, each program sentence transforms the graphs to reflect the changes in the memory configurations derived from the sentence execution. The RSGs are graphs in which nodes represent memory locations which have similar reference patterns. Therefore, a single node can safely and accurately represents several memory locations (if they are similarly referenced) without losing their essential characteristics.

To determine whether or not two memory locations should be represented by a single node, each one is annotated with a set of properties. Now, two different memory locations will be "summarized" in a single node if they fulfill the same properties. Note that the node inherits the properties of the memory locations represented by this node. Besides this, two nodes can be also summarized if they represent "summarizable" memory locations. This way, a possibly unlimited memory configuration can be represented by a limited size RSG, because the number of different nodes is limited by the number of properties of each node. These properties are related to the reference pattern used to access the memory locations represented by the node. Hence the name *Reference Shape Graph*.

As we have said, all possible memory configurations which may arise after the execution of a sentence are approximated by a set of RSGs. We call this set *Reduced Set of Reference Shape Graphs* (RSRSG), since not all the different RSGs arising in each sentence will be kept. On the contrary, several RSGs related to different memory configurations will be fused when they represent memory locations with similar reference patterns. There are also several properties related

to the RSGs, and two RSGs should share these properties to be joined. Therefore, besides the number of nodes in an RSG, the number of different RSGs associated with a sentence are limited too. This union of RSGs greatly reduces the number of RSGs and leads to a practicable analysis.

The symbolic execution of the code consists in the abstract interpretation of each sentence in the code. This abstract interpretation is carried out iteratively for each sentence until we reach a fixed point in which the resulting RSRSG associated with the sentence does not change any more [4]. This way, for each sentence that modifies dynamic structures, we have to define the abstract semantics which describes how these sentences modify the RSRSG. We consider six simple instructions that deal with pointers: $x = NULL$, $x = malloc$, $x = y$, $x \rightarrow sel = NULL$, $x \rightarrow sel = y$, and $x = y \rightarrow sel$. More complex pointer instructions can be built upon these simple ones and temporal variables.

The output RSRSG resulting from the abstract interpretation of a sentence over an input RSRSG$_i$ is generated by applying the abstract interpretation to each $rsg_i \in$ RSRSG$_i$. After the abstract interpretation of the sentence over the $rsg_i \in$ RSRSG$_i$ we obtain a set of output rsg_o. As we said, we cannot keep all the rsg_o arising from the abstract interpretation. On the contrary, each rsg_o will be compressed, which means the summarization of compatible nodes in the rsg_o. Furthermore, some of the rsg_os can be fused in a single RSG if they represent similar memory configurations. This operation greatly reduces the number of RSGs in the resulting RSRSG. In the worst case, the sequence of operations that the compiler carries out in order to symbolically execute a sentence are: graph division, graph prune, sentence symbolic execution (RSG modification), RSG compression and RSG union to build the final RSRSG. Due to space constraints we cannot formally describe this operations neither the abstract semantics carried out by the compiler. However, in order to provide an overview of our method we present a data structure example which will be refered to during the framework and operations description.

2.1 Working Example

The data structure, presented in Fig. 1 (a), is a doubly linked list of pointers to trees. Besides this, the leaves of the trees have pointers to doubly linked lists. The pointer variable S points to the first element of the doubly linked list (header list). Each item in this list has three pointers: nxt, prv, and $tree$. This $tree$ selector points to the root of a binary tree in which each element has the lft and rgh selectors. Finally, the leaves of the trees point to additional doubly linked lists. All the trees pointed to by the header list are independent and do not share any element. In the same way, the lists pointed to by the leaves of the same tree or different trees are also independent.

This data structure is built by a C code which traverses the elements of the header list with two pointers and eventually can permute two trees. Our compiler has analyzed this code obtaining an RSRSG for each sentence in the program. Figure 1 (b) shows a compact represetation of the RSRSG obtained for the last sentence of the code after the compiler analysis.