Algorithmic Debugging of SQL Views*

Rafael Caballero, Yolanda García-Ruiz and Fernando Sáenz-Pérez

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
{rafa,fernan}@sip.ucm.es and ygarciar@fdi.ucm.es

Abstract. We present a general framework for debugging systems of correlated SQL views. The debugger locates an erroneous view by navigating a suitable computation tree. This tree contains the computed answer associated with every intermediate relation, asking the user whether this answer is expected or not. The correctness and completeness of the technique is proven formally, using a general definition of SQL operational semantics. The theoretical ideas have been implemented in an available tool which includes the possibility of employing trusted specifications for reducing the number of questions asked to the user.

1 Introduction

SQL [12] is the *de facto* standard language for querying and updating relational databases. Its declarative nature and its high-abstraction level allows the user to easily define complex operations that could require hundreds of lines programmed in a general purpose language. In the case of relational queries, the language introduces the possibility of querying the database directly using a *select* statement. However, in realistic applications, queries can become too complex to be coded in a single statement and are generally defined using *views*. Views can be considered in essence as virtual tables. They are defined by a *select* statement that can rely on the database tables as well as in other previously defined views. Thus, views become the basic components of SQL queries. As in other programming paradigms, views can have bugs which produce unexpected results. However, we cannot infer that a view is buggy only because it returns an unexpected result. Maybe it is correct but receives erroneous input data from the other views or tables it depends on. There are very few tools for helping the user to detect the cause of these errors; so, debugging becomes a labor-intensive and time-consuming task in the case of queries defined by means of several intermediate views. The main reason for this lack of tools is that the usual *trace* debuggers used in other paradigms are not available here due to the high abstraction level of the language. A *select* statement is internally translated into a sequence of low level operations that constitute the *execution plan* of the query. Relating these operations to the original query is very hard, and debugging the execution plan step by step will be of little help. In this paper, we propose a theoretical framework for debugging SQL views based on *declarative debugging*, also known as *algorithmic debugging* [11]. This technique has been employed successfully in (constraint) logic programming [11], functional programming [9], functional-logic programming [2], and in deductive database languages [1]. The overall idea of declarative debugging [7] can be explained briefly as follows:

- The process starts with an initial error symptom, which in our case corresponds to the unexpected result of a user-defined view.
- The debugger automatically builds a tree representing the computation. Each node of the tree corresponds to an intermediate computation with its result. The children of a node are those nodes obtained from the subcomputations needed for obtaining the parent result. In our case, nodes will represent the computation of a relation R together with its answer. Children correspond to the computation of views and tables occurring in R if it is a view.
- The tree is *navigated*. An external oracle, usually the user, compares the computed result in each node with the *intended* interpretation of the associated relation. When a node contains the expected result, it is marked as *valid*, otherwise it is marked as *nonvalid*.
- The navigation phase ends when a nonvalid node with valid children is found. Such node is called a *buggy node*, and corresponds to an incorrect piece of code. In our case, the debugger will end pointing out either an erroneously defined view, or a table containing a nonvalid instance.

Our goal is to present a declarative debugging framework for SQL views, showing that it can be implemented in a realistic, scalable debugging tool.

* This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), Prometidos-CM (S2009TIC-1465) and GPD (UCM-BSCH-GR35/10-A-910502)

Owner

id	name
1	Mark Costas
2	Helen Kaye
3	Robin Scott
4	Tom Cohen

Pet

code	name	species
100	Wilma	dog
101	Kitty	cat
102	Wilma	cat
103	Lucky	dog
104	Rocky	dog
105	Oreo	cat
106	Cecile	turtle
107	Chelsea	dog

PetOwner

id	code
1	100
1	101
2	102
2	103
3	104
3	105
4	106
4	107

Fig. 1. All Pets Club database instance

We have implemented our debugging proposal in the Datalog Educational System (DES [10]), which makes it possible for Datalog and SQL to coexist as query languages for the same database. The current implementation of our proposal for debugging SQL views and instructions to use it can be downloaded from `https://gpd.sip.ucm.es/trac/gpd/wiki/GpdSystems/Des`.

2 SQL Semantics

The first formal semantics for relational databases based on the concept of set (e.g., relational algebra, tuple calculus [3]) were incomplete with respect to the treatment of non-relational features such as repeated rows and aggregates, which are part of practical languages such as SQL. Therefore, other semantics, most of them based on multisets [4], have been proposed. In our framework we will use the *Extended Relational Algebra* [6, 5]. We start by defining the concepts of database schemas and instances.

A *table schema* is of the form $T(A_1, \ldots, A_n)$, with T being the table name and A_i the attribute names for $i = 1 \ldots n$. We will refer to a particular attribute A by using the notation $T.A$. Each attribute A has an associated type (*integer*, *string*, …). An *instance* of a table schema $T(A_1, \ldots, A_n)$ is determined by its particular *rows*. Each row contains values of the correct type for each attribute in the table schema. *Views* can be thought of as new tables created dynamically from existing ones by using a SQL query. The general syntax of a SQL view is: create view $V(A_1, \ldots, A_n)$ as Q, with Q a SQL *select* statement, and $V.A_1, \ldots, V.A_n$ the names of the view attributes. In general, we will use the name *relation* to refer to either a table or a view (observe that the mathematical concept of relation is defined over sets, but in our setting we define relations among multisets). A *database schema* D is a tuple $(\mathcal{T}, \mathcal{V})$, where $\mathcal{T}$ is a finite set of table schemas and $\mathcal{V}$ a finite set of view definitions. Although database schemas also include constraints such as primary keys, they are not relevant to our setting.

A *database instance* d of a database schema is a set of table instances, one for each table in $\mathcal{T}$. To represent the instance of a table T in d we will use the notation $d(T)$.

The syntax of SQL queries can be found in [12]. The *dependency tree* of any view V in the schema is a tree with V labeling the root, and its children the dependency trees of the relations occurring in its query. The next example defines a particular database schema that will be used in the rest of the paper as a running example.

Example 1. The *Dog and Cat Club* annual dinner is going to take place in a few weeks, and the organizing committee is preparing the guest list. Each year they browse the database of the *All Pets Club* looking for people that own at least one cat and one dog. Owners come to the dinner with all their cats and dogs. However, two additional constraints have been introduced this year:

- People owning more than 5 animals are not allowed (the dinner would become too noisy).
- No animals sharing the same name are allowed at the party. This means that if two different people have a cat or dog sharing the same name neither of them will be invited. This severe restriction follows after last year's incident, when someone cried *Tiger* and dozens of pets started running without control.

Figure 1 shows the *All Pets Club* database instance. It consists of three tables: *Owner*, *Pet*, and *PetOwner* which relates each owner with its pets. Primary keys are shown underlined. Figure 2 contains the views for selecting the dinner guests. The first view is *AnimalOwner*, which obtains all the tuples (*id*, *aname*, *species*) such that *id* is the owner of an animal of name *aname* of species *species*. *LessThan6* returns the identifiers of the owners with less than six cats and dogs. *CatsAndDogsOwner* returns pairs (*id*, *aname*) where *id* is the identifier of the owner

```
create or replace view AnimalOwner(id,aname,species) as
 select O.id, P.name, P.species
 from Owner O, Pet P, PetOwner PO
 where O.id = PO.id and P.code = PO.code;

create or replace view LessThan6(id) as
 select id from AnimalOwner
 where species='cat' or species='dog'
 group by id having count(*)<6;

create or replace view CatsAndDogsOwner(id,aname) as
 select AO1.id,AO1.aname
 from AnimalOwner AO1, AnimalOwner AO2
 where AO1.id = AO2.id and AO1.species='dog'
      and AO2.species='cat';

create or replace view NoCommonName(id) as
 select id from CatsAndDogsOwner
 except
 select B.id from CatsAndDogsOwner A, CatsAndDogsOwner B
                  where A.id <> B.id
                  and A.aname = B.aname;

create or replace view Guest(id,name) as
 select id, name
 from Owner natural inner join NoCommonName
            natural inner join LessThan6;
```

Fig. 2. Views for selecting dinner guests

of either a cat or a dog with name *aname*, such that *id* owns both cats and dogs. *NoCommonName* is defined by removing owners sharing pet names from the total list of cats and dog owners. Finally, the main view is *Guest*, which selects those owners that share no pet name with another owner (view *NoCommonName*) and that have less than six cats and dogs (view *LessThan6*). However, these views contain a bug that will become apparent in the next sections.

The Extended Relational Algebra (ERA from now on) [6] is an operational SQL Semantics allowing aggregates, views and most of the common features of SQL queries. The main characteristics of ERA are:

1. The table instances and the result of evaluating queries/views are multisets, (it is also possible to consider *lists* instead of multisets if we consider relevant the order among rows in a query result).
2. ERA expressions define new relations by combining previously defined relations using multiset operators (see [5] for a formal definition of each operator).
3. We use Φ_R to represent a SQL query or view R as an ERA expression, as explained in [5]. Since a query/view depends on previously defined relations, sometimes it will be useful to write $\Phi_R(R_1, \ldots, R_n)$ indicating that R depends on $R_1, \ldots, R_n$. If $M_1, \ldots, M_n$ are multisets we use the notation $\Phi_R(M_1, \ldots, M_n)$ to indicate that the expression Φ_R is evaluated after substituting $R_1, \ldots, R_n$ by $M_1, \ldots, M_n$.
4. Tables are denoted by their names, that is, $\Phi_T = T$ if T is a table.
5. The computed answer of Φ_R with respect to some schema instance d will be denoted by $\| \Phi_R \|_d$, where
   - If R is a database table, $\| \Phi_R \|_d = d(R)$.
   - If R is a database view or a query and $R_1, \ldots, R_n$ the relations defined in R then $\| \Phi_R \|_d = \Phi_R(\| \Phi_{R_1} \|_d, \ldots, \| \Phi_{R_n} \|_d)$.

Observe that $\| \Phi_R \|_d$ is well defined since mutually recursive view definitions are not allowed[1]. We assume that $\| \Phi_R \|_d$ actually corresponds to the answer obtained by a correct SQL implementation, i.e., that the available SQL systems implement ERA. In fact our proposal is valid for any semantics that associate a formula Φ_R to any relation R and allow the recursive definition of computed answer of item 5 above.

[1] Recursive views are allowed in the SQL:1999 standard but they are not supported in all the systems, and they are not considered here.

AnimalOwner

id	aname	species
1	Wilma	dog
1	Kitty	cat
2	Wilma	cat
2	Lucky	dog
3	Rocky	dog
3	Oreo	cat
4	Cecile	turtle
4	Chelsea	dog

LessThan6

id
1
2
3
4

CatsAndDogsOwner

id	aname
1	Wilma
1	Kitty
2	Wilma
2	Lucky
3	Oreo
3	Rocky

NoCommonName

id
3

Guest

id	name
3	Robin Scott

Fig. 3. Intended answer for the views in Example 1

3 Declarative Debugging Framework

In this section, we assume a set of SQL views $\mathcal{V} = \{V_1, \ldots, V_n\}$ such that for some $1 \leq i \leq n$, and for some database instance d, V_i has produced an unexpected result in some SQL system. We also assume that this SQL system implements the ERA operational semantics of previous section. Our debugging technique will be based on the comparison between the answers by a SQL system implementing the ERA semantics, and the oracle intended answers. Next, we define the concept of intended answer for schema relations.

Definition 1. Intended Answers for Schema Relations
Let D be a database schema, d an instance of D, and R a relation defined in D. The intended answer for R w.r.t. d, is a multiset denoted as $\mathcal{I}(R,d)$ containing the answer that the user expects for the query select * from R; *in the instance d.*

The intended answer depends not only on the view semantics but also on the contents of the tables in the instance d. This concept corresponds to the idea of *intended interpretations* employed usually in algorithmic debugging. Figure 3 contains the intended answer for each view defined in Figure 2. For instance, it is expected that *AnimalOwner* will identify each owner *id* with the names and species of his pets. It is also expected than *LessThan6* will contain the *id* of all four owners, since all of them have less than six cats and dogs. The intended answer for view *CatsAndDogsOwner* contains the *id* and *name* attributes of those entries in *AnimalOwner* corresponding to owners with at least one dog and one cat, and removing pets different from cats and dogs. View *NoCommonName* is expected to contain only one row for owner with identifier 3. The reason is that both owners 1 and 2 share a pet name (*Wilma*). Finally, the only expected *Guest* will be the owner with identifier 3, *Robin Scott*. If now we try the query select * from Guest; in a SQL system, we obtain a computed answer representing the multiset $\{\!|(1, Mark\ Costas),\ (2, Helen\ Kaye),\ (3, Robin\ Scott)|\!\}$. This computed answer is different from the intended answer for *Guest*, and indicates that there is some error. However, we cannot ensure that the error is in the query for *Guest*, because the error can come from any of the relations in its *from* clause. And also from the relations used by these relations, and so on. In order to define the key concept of erroneous relation it will be useful to define the auxiliary concept of *inferred answer*.

Definition 2. Inferred Answers
Let D be a database schema, d an instance of D, and R a relation in D. The inferred answer for R, with respect to d, $\mathcal{E}(R,d)$, is defined as

1. *If R is a table, $\mathcal{E}(R,d) = d(R)$.*
2. *If R is a view, $\mathcal{E}(R,d) = \Phi_R(\mathcal{I}(R_1,d), \ldots, \mathcal{I}(R_n,d))$ with $R_1, \ldots, R_n$ the relations occurring in R.*

Thus, in the case of tables, the inferred answer is just its table instance. In the case of a view V, the inferred answer corresponds to the computed result that would be obtained assuming that all the relations R_i occurring in the definition of V contain the intended answers. For instance, consider Example 1 and the instance d of Figure 1. Assume that all the tables contain the intended answers, i.e., for every table T, $\mathcal{I}(T,d) = d(T)$. Then the inferred answer for view *CatsAndDogsOwner* is the same as its computed answer $\parallel CatsAndDogsOwner \parallel_d$:

$$\mathcal{E}(CatsAndDogsOwner, d) = \Phi_{CatsAndDogsOwner}(\mathcal{I}(AnimalOwner, d)) = \\ \{\!|\ (1, Wilma), (2, Lucky), (3, Rocky)|\!\}$$

However, this result is different from the intended answer for this view (Fig. 3). A discrepancy between $\mathcal{I}(R,d)$ and $\mathcal{E}(R,d)$ shows that R does not compute its intended answer, even assuming that all the relations it depends on contain their intended answers. Such relation is erroneous:

Definition 3. Erroneous Relation
Let D be a database schema, d an instance of D, an R a relation defined in D. We say that R is an erroneous relation when $\mathcal{I}(R,d) \neq \mathcal{E}(R,d)$.

Definition 3 clarifies the fundamental concept of erroneous relation. However, it cannot be used directly for defining a practical debugging tool, because in order to point out a view V as erroneous, it would require comparing $\mathcal{I}(V,d)$ and $\mathcal{E}(V,d)$. By Definition 2, to obtain $\mathcal{E}(V,d)$, the tool will need the intended answer $\mathcal{I}(R,d)$ for every R occurring in the query defining V. But $\mathcal{I}(R,d)$ is only known by the user, who should provide this information during the debugging process. Obviously, a technique requiring such amount of information would be rejected by most of the users. Instead, we will require from the oracle only to answer questions of the form *'Is the computed answer (...) the intended answer for view V?'* Thus, the declarative debugger will compare the computed answer –obtained from the SQL system– and the intended answer –known by the oracle– In a first phase, the debugger builds a *computation tree* for the main view. The definition of this structure is the following:

Definition 4. Computation Trees
Let D be a database schema with views $\mathcal{V}$, d an instance of D, and R a relation defined in D. The computation tree $CT(R,d)$ associated with R w.r.t. d is defined as follows:

- *The root of $CT(R,d)$ is $(R \mapsto \parallel \Phi_R \parallel_d)$.*
- *For any node $N = (R' \mapsto \parallel \Phi_{R'} \parallel_d)$ in $CT(R,d)$:*
  - *If R' is a table, then N has no children.*
  - *If R' is a view, the children of N will correspond to the CTs for the relations occurring in the query associated with R'.*

In practice, the nodes in the computation tree correspond to the syntactic dependency tree of the main SQL view, with the children at each node corresponding to the relations occurring in the definition of the corresponding view. After building the computation tree, the debugger will *navigate* the tree, asking the oracle about the validity of some nodes:

Definition 5. Valid, Nonvalid and Buggy Nodes
Let $T = CT(R,d)$ be a computation tree, and $N = (R' \mapsto \parallel \Phi_{R'} \parallel_d)$ a node in T. We say that N is valid *when $\parallel \Phi_{R'} \parallel_d = \mathcal{I}(R',d)$,* nonvalid *when $\parallel \Phi_{R'} \parallel_d \neq \mathcal{I}(R',d)$, and* buggy *when N is nonvalid and all its children in T are valid.*

The goal of the debugger will be to locate buggy nodes. The next theorem shows that a computation tree with a nonvalid root always contains a buggy node, and that every buggy node corresponds to an erroneous relation.

Theorem 1. *Let d be an instance of a database schema D, V a view defined in D, and T a computation tree for V w.r.t. d. If the root of T is nonvalid then:*

- *Completeness. T contains a buggy node.*
- *Soundness. Every buggy node in T corresponds to an erroneous relation.*

The debugging process starts when the user finds a view V returning an unexpected result. The debugger builds the computation tree for V, which has a nonvalid root as required by the theorem. Figure 4 shows the computation tree for our running example (after removing the repeated children). Nonvalid nodes are underlined, and the only buggy node (in bold face) corresponds to view *CatsAndDogsOwner*.

4 Conclusions

In this paper, we propose using algorithmic debugging for finding errors in systems involving several SQL views. To the best of our knowledge, it is the first time that a debugging tool of these characteristics has been proposed. The debugger is based on the navigation of a suitable computation tree corresponding to some view returning some unexpected result. The validity of the nodes in the tree is determined by an external oracle, which can be either the user, or a trusted specification containing a correct version of part of the views in the system. The debugger ends when a buggy node, i.e., a nonvalid node with valid children, is found. We prove formally that every buggy node corresponds to an erroneous relation, and that every computation tree with a nonvalid root contains some buggy node. Although the results are established in the context of the Extended Relational

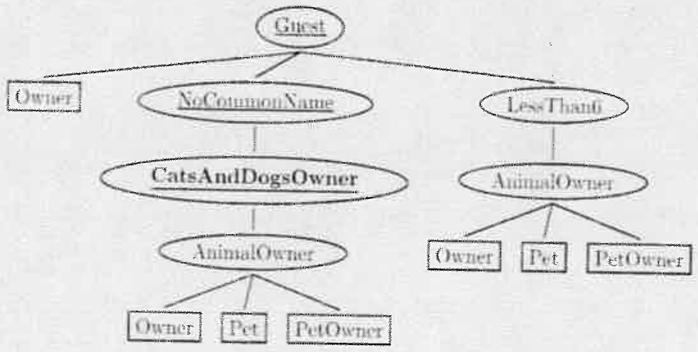

Fig. 4. Computation tree for view Guest

Algebra, they can be easily extended to other possible SQL semantics, such as the Extended Three Valued Predicate Calculus [8].
The technique is easy to implement, obtaining an efficient, platform-independent and scalable debugger without much effort. The tool is very intuitive, because it automates what usually is done when an unexpected answer is found in a system with several views: check the relations in the *from* clause, and if some of them return an unexpected answer, repeat the process. Automating this process is of great help, especially when the tool includes additional features as advanced navigation strategies, or the possibility of using trusted specifications. We have successfully implemented our proposal in the existing, widely-used DES system.
As future work, we plan the development of a graphical interface, which can be very helpful for inspecting the computation tree providing information about the node validity. It will also be useful to consider individual wrong tuples in the unexpected results and study its provenance[13], for a fine-grain error detection.

References

1. R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. A theoretical framework for the declarative debugging of datalog programs. In *SDKB 2008*, volume 4925 of *LNCS*, pages 143–159. Springer, 2008.
2. R. Caballero, F. López-Fraguas, and M. Rodríguez-Artalejo. Theoretical Foundations for the Declarative Debugging of Lazy Functional Logic Programs. In *Proc. FLOPS'01*, number 2024 in LNCS, pages 170–184. Springer, 2001.
3. E. Codd. Relational Completeness of Data Base Sublanguages. In Rustin, editor, *Data base Systems*, Courant Computer Science Symposia Series 6. Englewood Cliffs, N.J. Prentice-Hall, 1972.
4. U. Dayal, N. Goodman, and R. H. Katz. An extended relational algebra with control over duplicate elimination. In *PODS '82: Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 117–123, New York, NY, USA, 1982. ACM.
5. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2008.
6. P. W. Grefen and R. A. B. de. A multi-set extended relational algebra: a formal approach to a practical issue. In *10th International Conference on Data Engineering*, pages 80–88. IEEE, 1994.
7. L. Naish. A Declarative Debugging Scheme. *Journal of Functional and Logic Programming*, 3. 1997.
8. M. Negri, G. Pelagatti, and L. Sbattella. Formal semantics of SQL queries. *ACM Trans. Database Syst.*, 16(3):513–534, 1991.
9. H. Nilsson. How to look busy while being lazy as ever: The implementation of a lazy functional debugger. *Journal of Functional Programming*, 11(6):629–671, 2001.
10. F. Sáenz-Pérez. DES: A Deductive Database System. In *Spanish Conference on Programming and Computer Languages*, September 2010. In Press.
11. E. Shapiro. *Algorithmic Program Debugging.* ACM Distiguished Dissertation. MIT Press, 1982.
12. SQL, ISO/IEC 9075:1992, third edition, 1992.
13. S. Vansummeren and J. Cheney. Recording provenance for sql queries and updates. *IEEE Data Eng Bull*, 30(4):29–37, 2007.

Ershov Informatics Conference

PSI Series, 8th Edition

Preliminary Proceedings

June 27 – July 1, 2011
Akademgorodok
Novosibirsk, Russia

ERSHOV INFORMATICS CONFERENCE

PSI Series, 8th Edition

June 27 – July 1, 2011, Akademgorodok, Novosibirsk, Russia

Organized by:

◇ A.P. Ershov Institute of Informatics Systems,
Siberian Branch of Russian Academy of Sciences
◇ Novosibirsk State University

Sponsored by:

◇ Russian Foundation for Basic Research
◇ Google
◇ EMC^2
◇ Intel Corporation
◇ PlanetData
◇ SESAME-S
◇ Microsoft Research
◇ Formal Methods Europe
◇ DFG
◇ Centre National de la Recherche Scientifique,
Laboratoire J.-V. Poncelet franco-russe

Yolanda García Ruiz

Hek aquí incluid

UDK 519.6

Ershov Informatics Conference (Proceedings). – Novosibirsk: A.P. Ershov Institute of Informatics Systems, 2011. — 338 p.

The volume comprises the papers selected for presentation at the Ershov Informatics Conference (PSI Series, the 8th edition), Akademgorodok, Novosibirsk, Russia, June, 27 – July, 1, 2011. The main goal of the conference is to give an overview of research directions which are decisive for the growth of major areas of research activities in computer, software and information sciences.

FOREWORD

Welcome to the Ershov Informatics Conference (PSI Series, the 8th edition).

PSI is a forum for academic and industrial researchers, developers and users working on topics relating to computer, software and information sciences. The conference serves to bridge the gaps between different communities whose research areas are covered by but not limited to foundations of program and system development and analysis, programming methodology and software engineering, and information technologies. Another aim of the conference is the improvement of contacts and exchange of ideas between researchers from the East and West.

The previous seven PSI conferences were held in 1991, 1996, 1999, 2001, 2003, 2006 and 2009, respectively, and proved to be significant international events. Traditionally, PSI offers a program of tutorials, invited lectures, presentations of contributed papers and workshops complemented by a social program reflecting the amazing diversity of Novosibirsk culture and history.

The PSI 2011 conference is dedicated to the 80th anniversary of a pioneer in theoretical and system programming research, academician Andrei Petrovich Ershov (1931–1988), and to the 100th anniversary of one of the founders of cybernetics, a member of the Soviet Academy of Sciences Aleksei Andreevich Lyapunov (1911–1973).

Aleksei Lyapunov was one of the early pioneers of computer science in this country. He worked at the Steklov Institute of Mathematics and Institute of Applied Mathematics in Moscow. In 1961 he moved to Novosibirsk where the Siberian Branch of the USSR Acadamy of Sciences had been founded. First A.A. Lyapunov worked at the Institute of Mathematics, then he led the Laboratory of Cybernetics at the Institute of Hydrodynamics. He played an important role in the organization of the Physics and Mathematics School for talented children from all over Siberia, and was a professor at the Novosibirsk State University. In 1996 he posthumously received the IEEE Computer Society Computer Pioneer Award for his contribution to Soviet cybernetics and programming. There are many distinguished mathemicians among the students of A.A. Lyapunov, including the academician A.P. Ershov.

Andrei Ershov graduated from the Moscow State University in 1954. He began his scientific career under the guidance of professor Lyapunov who was the supervisor of his PhD thesis. A.P. Ershov worked at the Institute of Precise Mechanics and Computing Machinery, and later headed the Theoretical Programming Department at the Computing Center of the USSR Academy of Sciences in Moscow. In 1958 the Department of Programming was organized at the Institute of Mathematics of Siberian Branch of the USSR Academy of Sciences, and by the initiative of the academician S.L. Sobolev Ershov was appointed the head of this Department, which later became part of the Computing Center in Novosibirsk Akademgorodok. The first significant project of the Department was aimed at the development of ALPHA system, an optimizing compiler for an extension of Algol 60 implemented on a Soviet computer, M-20. Later the researchers of the Department created the Algibr, Epsilon, Sigma, and Alpha-6 programming systems for the BESM-6 computers. The list of achievements also includes the first Soviet time-sharing system AIST-0, the multi-language system BETA, research projects in artificial intelligence and parallel programming, integrated tools for text processing and publishing, and many more. A.P. Ershov was a leader and a participant of these projects. In 1974 he was nominated as a Distinguished Fellow of the British Computer Society. In 1981 he received the Silver Core Award for services rendered to IFIP. Andrei Ershov's brilliant speeches were always in the focus of public attention. Especially notable was his lecture on "Aesthetic and human factor in programming" presented at the AFIPS Spring Joint Computer Conference (Atlantic City, USA) in 1972.

60 papers have been submitted to the conference by researchers from 26 countries. Each paper was reviewed by four experts, at least three of them from the same or closely related discipline as the authors. The reviewers generally provided high quality assessment of the papers and often gave extensive comments to the authors for the possible improvement of the contributions. As the result, the Programme Committee has selected 18 high quality papers as regular talks and 10 papers as short talks to be presented at the conference. A range of hot topics in computer science and informatics will be covered by a tutorial and five invited talks given by prominent computer scientists from different countries.

We are glad to express our gratitude to all the persons and organizations who contributed to the conference — to the authors of the papers for their effort in producing the materials included here, to the sponsors for their moral, financial and organizational support, to the steering committee members for their coordination of the conference, to the programme committee members and the reviewers who did their best to review and select the papers, and to the members of the organizing committee for their contribution to the success of this event and its cultural program.

The programme committee work was done using the EasyChair conference management system.

June, 2011

Edmund Clarke,
Irina Virbitskaite,
Andrei Voronkov

CONFERENCE CHAIR

Alexander Marchuk (Novosibirsk, Russia)

STEERING COMMITEE

Members:
Dines Bjørner (Lyngby, Denmark)
Manfred Broy (München, Germany)
Victor Ivannikov (Moscow, Russia)
Bertrand Meyer (Zürich, Switzerland)
Leonid Libkin (Edinburgh, UK)

Honorary Member:
Tony Hoare (Cambridge, UK)

CONFERENCE SECRETARY

Natalya Cheremnykh (Novosibirsk, Russia)

PROGRAMME COMMITTEE CHAIRS

Edmund M. Clarke (Pittsburgh, USA)
Irina Virbitskaite (Novosibirsk, Russia)
Andrei Voronkov (Manchester, UK)

INVITED SPEAKERS

Eike Best (Oldenburg, Germany)
Peter Buneman (Edinburgh, UK)
Rupak Majumdar (Kaiserslautern, Germany)
Ugo Montanari (Pisa, Italy)
Andreas Zeller (Saarbrüken, Germany)

PROGRAMME COMMITTEE

Samson Abramsky (Oxford, UK)
Frédéric Benhamou (Nantes, France)
Leopoldo Bertossi (Carleton, Canada)
Eike Best (Oldenburg, Germany)
Kim Bruce (Pomona, USA)
Mikhail Bulyonkov (Novosibirsk, Russia)
Gabriel Ciobanu (Iaşi, Romania)
Dieter Fensel (Innsbruck, Austria)
Jean Claude Fernandez (Grenoble, France)
Jan Friso Groote (Eindhoven, The Netherlands)
Heinrich Herre (Leipzig, Germany)
Victor Kasyanov (Novosibirsk, Russia)
Joost-Pieter Katoen (Aachen, Germany)
Laura Kovács (Vienna, Austria)
Gregory Kucherov (Lille, France)
Kim Guldstrand Larsen (Aalborg, Denmark)
Johan Lilius (Turku, Finland)
Pericles Loucopoulos (Loughborough, UK)
Andrea Maggiolo-Schettini (Pisa, Italy)
Klaus Meer (Cottbus, Germany)
Dominique Méry (Nancy, France)
Torben Mogensen (Copenhagen, Denmark)
Hanspeter Mössenböck (Linz, Austria)
Peter Mosses (Wales, UK)
Peter Müller (Zürich, Switzerland)
Valery Nepomniaschy (Novosibirsk, Russia)
Nikolaj Nikitchenko (Kiev, Ukraine)
José R. Paramá (A Coruña, Spain)
Francesco Parisi-Presicce (Rome, Italy)
Wojciech Penczek (Warsaw, Poland)
Peter Pepper (Berlin, Germany)
Alexander Petrenko (Moscow, Russia)
Jaroslav Pokorný (Prague, Czech Republic)
Vladimir Polutin (Moscow, Russia)
Wolfgang Reisig (Berlin, Germany)
Donald Sannella (Edinburgh, UK)
Klaus-Dieter Schewe (Hagenberg, Austria)
David Schmidt (Manhattan, USA)
Alexander Semenov (Novosibirsk, Russia)
Nikolay Shilov (Novosibirsk, Russia)
Val Tannen (Philadelphia, USA)
Lothar Thiele (Zürich, Switzerland)
Alexander Tomilin (Moscow, Russia)
Mark Trakhtenbrot (Rehovot, Israel)
Alexander L. Wolf (London, UK)
Tatyana Yakhno (Izmir, Turkey)
Wang Yi (Uppsala, Sweden)

REFEREES

Agrigoroaiei, Oana
Aman, Bogdan
Andriamiarina, Manamiary
Anureev, Igor
Attiogb, Christian
Bauereiß, Thomas
Bochman, Alex
Brihaye, Thomas
Ceberio, Martine
Cranen, Sjoerd
Danelutto, Marco
Demin, Alexander
Dikovsky, Alexander
Ferrara, Pietro
Fertin, Guillaume
Fleischhack, Hans
Garanina, Natalia
Gierds, Christian
Gluck, Robert
Gordeev, Dmitry
Gray, Robert M.
Gretz, Friedrich
Guan, Nan
Hoger, Christoph
Idrisov, Renat
Jacob, Marie
Juhasz, Uri
Kassios, Ioannis
Keiren, Jeroen
Knapik, Michal
Kreinovich, Vladik
Lamarre, Philippe
Lampka, Kai
Lorenzen, Florian
Lv, Mingsong
Ma, Hui
Melatti, Igor
Meski, Artur
Mikucionis, Marius
Monahan, Rosemary
Muller, Richard
Nguyen, Viet Yen
Niehren, Joachim
Niewiadomski, Artur
Olesen, Mads Chr.
Olsen, Petur
Perathoner, Simon
Promsky, Alexey
Prufer, Robert
Pyjov, Konstantin
Raffelsieper, Matthias
Rama, Aureliano
Rohloff, Judith
Ruskiewicz, Joseph
Schulte, Christian
Shkurko, Dmitry
Singh, Neeraj
Stasenko, Alexander
Stigge, Martin
Stoimenov, Nikolay
Stoyanovich, Julia
Szreter, Maciej
Tarasyuk, Igor
Thalhammer, Andreas
Tronci, Enrico
Von Styp, Sabrina
Wagner, Christoph
Wang, Qing
Willemse, Tim
Winkowski, Jozef
Yang, Hoeseok
Zwirchmayr, Jakob

TABLE OF CONTENTS

Tutorial

Zeller A. Mining Programs 1

Invited Talks

Best E., Darondeau Ph. Petri Net Distributability 2

Bruni R., Melgratti H., Montanari U. Connector Algebras and Petri Nets 3

Buneman P. Models of Provenance 8

Majumdar R. End-to-End Guarantees in Embedded Control Systems 9

Zeller A. Mining Precise Specifications 10

Regular Papers

Baar Th., Kumar Ph. Detecting Entry Points in Java Libraries 11

Beyer M., Glesner S. Static Analysis of Run-Time Modes in Synchronous Process Network 20

Demin A., Ponomaryov D., Vityaev E. Probabilistic Concepts in Formal Contexts 29

Fried M.A.H., Fensel A., Facca F.M., Fensel D. An Extensible System for Enhancing Social Conference Experience 39

Garanina N.O. Exponential Acceleration of Model Checking for Perfect Recall Systems 50

Glück R. Bootstrapping Compiler Generators from Partial Evaluators 59

Hamilton G.W., Jones N.D. Proving the Correctness of Unfold/Fold Program Transformations Using Bisimulation 69

Khoroshilov A., Mutilin V., Novikov E., Shved P., Strakh A. Towards an Open Framework for C Verification Tools Benchmarking 82

Klimov A.V. Coverability Problem for Monotonic Counter Systems by Supercompilation 92

Klyuchnikov I., Romanenko S. Multi-Result Supercompilation as Branching Growth of the Penultimate Level in Metasystem Transitions 104

Knoop J., Kovács L., Zwirchmayr J. Symbolic Loop Bound Computation for WCET Analysis 116

Korovin K., Tsiskaridze N., Voronkov A. Implementing Conflict Resolution 127

Korovina M., Vorobjov N. Reachability in One-Dimensional Controlled Polynomial Dynamical Systems 137

Mardare R. Decidability of Modular Logics for Concurrency 144

Mogensen T.Æ. Partial Evaluation of Janus Part 2: Assertions and Procedures 155

Sharma A., Katoen J.-P. Weighted Lumpability on Markov Chains 164

Studer Th. Justified Terminological Reasoning 177

Veanes M., Bjørner N. Symbolic Tree Transducers 186

Short Papers

Bozhenkova E.N. Compositional Methods in Characterization of Timed Event Structures 198

Caballero R., García-Ruiz Y., Sáenz-Pérez F. Algorithmic Debugging of SQL Views 204

Dubtsov R. Timed Transition Systems with Independence and Marked Scott Domains: an Adjunction 210

Gribovskaya N.S. A Logic Characteristic For Timed Extensions of Partial Order Based Equivalences 216
Jaskelioff M., Russo A. Secure Multi-Execution in Haskell 223
Korovin K., Voronkov A. GoRRiLA and Hard Reality 229
Letichevsky A.A.,Letychevskyi O.A., Peschanenko V.S. Insertion Modeling System 234
Petrov E. Scalable Parallel Interval Propagation for Sparse Constraint Satisfaction Problems 242
Ponomarenko A., Rubanov V. A Combined Technique for Automatic Detection of Backward Binary Compatibility Problems 249
Shilov N.V., Akinin A.A., Zubkov A.V., Idrisov R.I. Development of the Computer Language Classification Knowledge Portal 255

Posters

Anureev I., Maryasov I., Nepomniaschy V. The Mixed Axiomatic Semantics Method for C-program Verification 261
Baranov S., Kotlyarov V., Weigert Th. Test Automation with Verifiable Coverage Criteria 267
Klimov A. Transforming Affine Nested Loop Programs to Dataflow Computation Model 274
Polo Usaola M., Reales Mateo P., Pérez Lamancha B. Reduction of Test Suites Using Mutation 286
Promsky A. V. Verification Condition Understanding 295
Shelekhov V. Rules of Correctness Proof for Programs with Simple Logic 301
Tarasyuk I.V. Performance Analysis of the Dining Philosophers System in dtsPBC 309
Zagorulko Yu., Zagorulko G. Open Extensible Tools for Development of Intelligent DSS: Towards an Architecture 322

Author Index

Akinin, Alexander 255
Anureev, Igor 261
Baar, Thomas 11
Baranov, Sergey 267
Best, Eike 2
Beyer, Michael 20
Bjørner, Nikolaj 186
Bozhenkova, Elena 198
Bruni, Roberto 3
Buneman, Peter 8
Caballero, Rafael 204
Darondeau, Philippe 2
Demin, Alexander 29
Dubtsov, Roman 210
Facca, Federico Michele 39
Fensel, Anna 39
Fensel, Dieter 39
Fried, Michael Alois Helmut 39
Garanina, Natalia 50
García-Ruiz, Yolanda 204
Glesner, Sabine 20
Glück, Robert 59
Gribovskaya, Natalia 216
Hamilton, Geoff 69
Idrisov, Renat 255
Jaskelioff, Mauro 223
Jones, Neil 69
Katoen, Joost-Pieter 164
Khoroshilov, Alexey 82
Klimov, Andrei 92
Klimov, Arkady 274
Klyuchnikov, Ilya 104
Knoop, Jens 116
Korovin, Konstantin 127, 229
Korovina, Margarita 137
Kotlyarov, Vsevolod 267
Kovács, Laura 116
Kumar, Philipp 11
Letichevsky, Alexander 234
Letichevskyi, Olexander 234
Majumdar, Rupak 9
Mardare, Radu 144
Maryasov, Ilya 261
Melgratti, Hernán 3
Mogensen, Torben Ægidius 155
Montanari, Ugo 3
Mutilin, Vadim 82
Nepomniaschy, Valery 261
Novikov, Eugene 82
Peschanenko, Vladimir 234
Petrov, Evgueni 242
Polo Usaola, Macario 286
Ponomarenko, Andrey 249
Ponomaryov, Denis 29
Promsky, Alexei 295
Pérez Lamancha, Beatriz 286
Reales Mateo, Pedro 286
Romanenko, Sergei 104
Rubanov, Vladimir 249
Russo, Alejandro 223
Sáenz-Pérez, Fernando 204
Sharma, Arpit 164
Shelekhov, Vladimir 301
Shilov, Nikolay 255
Shved, Pavel 82
Strakh, Alexander 82
Studer, Thomas 177
Tarasyuk, Igor 309
Tsiskaridze, Nestan 127
Veanes, Margus 186
Vityaev, Evgeny 29
Vorobjov, Nicolai 137
Voronkov, Andrei 127, 229
Weigert, Thomas 267
Zagorulko, Galina 322
Zagorulko, Yury 322
Zeller, Andreas 1, 10
Zubkov, Alexey 255
Zwirchmayr, Jakob 116