Position Statements on Strategic Directions for Research on Programming Languages

Chris Hankin
Imperial College, London
and
Hanne Riis Nielson
University of Aarhus
and
Jens Palsberg
Purdue University

One of the working groups at the workshop on *Strategic Directions in Computing Research* was concerned with *Programming Languages*. We summarise the strategic directions identified by the group and we introduce 16 of the position statements that formed the basis for the report. These statements can then be found at the following pages.

As part of the celebration of it's 50th anniversary, the ACM organised a workshop on *Strategic Directions in Computing Research* at MIT, Boston, in June 1996. The goal was to have a number of working groups examine the current status and future directions of computing research as well as strategic issues. More than 20 working groups were created including one on *Programming Languages*.

The working group has prepared a *report* and each of the participants were invited to write a *position statement*. The report [1] appears in Computing Surveys, Vol. 28(4), December 1996. The position statements appear in this volume of SIGPLAN Notices; also they appear as an electronic supplement to the above mentioned Computing Surveys issue.

We shall begin with a summary of some of the main points of the report: we refer to [1] for more details. Then we give a brief introduction to the position statements to be found on the following pages.

1. PROGRAMMING LANGUAGE RESEARCH

Programming language research covers a broad spectrum from systems work through to theory. The area studied by the working group is mainly concerned with the discovery of principles and generic techniques; it is not, with a few notable exceptions, about the invention of new languages. Many of the results of the area are generally applicable to a wide variety of language paradigms including procedural languages, functional and logic languages, object-oriented languages etc. Many common programming language features have emerged from this area.

In addition to language features, programming language research has also produced important techniques. Let us mention the five areas that received special attention by the working group:

—*Semantics*: Formal semantics is concerned with the description of program meanings by operational, denotational or axiomatic specifications. It improves our understanding of new as well as well-known programming constructs and it provides a yardstick for implementation and a foundation for analysis and verification techniques and program transformation. Over the years different techniques have been developed to handle the different programming paradigms and the different applications.

—*Type systems*: A type is a collection of values which share a common structure, operations and other properties. A type system is a specification of how types are assigned to values. Type safety – the prevention of certain classes of programming errors – is a desirable property of any programming language; many of the recently published standards for safety critical software insist upon the use of strongly typed programming languages. Over the years different type systems have been developed and found their ways into commercially successful languages.

—*Program analysis*: Program analysis is concerned with the problem of statically predicting properties of the dynamic executions of programs. Traditionally, program analysis has been used extensively to enable various optimizations and transformations in compilers; among the newer applications is the validation of software to reduce the likelihood of malicious behaviour. Over the years a wide variety of techniques have been developed to handle the different analysis problems and different programming paradigms.

—*Program transformation*: The goal of program transformation is to modify some representation of the program to change some of its properties whilst preserving others. For example, most useful program transformations preserve the input/output semantics of the program but might radically change the program's complexity. The transformation of programs is an important technique in the development of reliable and efficient software. Techniques have been developed for different stages of the software development process: when developing programs from specifications, when specialising existing programs to specific contexts and, in particular, in optimizing compilers.

—*Implementation*: This area concerns compilation and run-time support. Current concerns in compiler technology include compilation for distributed systems, optimisations across module boundaries and correctness issues. Memory management, particularly on complex cache architectures, is a critical concern in run-time support. This area is increasingly concerned with the development of generic tools rather than tailored solutions.

Programming language research in the 1990s is slowly changing from what it was in the 1970s and 1980s: the traditional uni-processor machines are being replaced by heterogeneous and physically distributed computer networks, the traditional line based user interfaces are being replaced by graphical user interfaces, multi-media applications are the rule rather than the exception, etc. There is a growing need for programming languages to adapt to this fundamental change and thereby for programming language research to address these problems more thoroughly: We need to understand the semantics of the language features developed for programming these systems; we need to understand what type security means in this setting;

we need to develop program analyses that statically predict the behaviour of these systems; and we need to develop implementation techniques that utilise the vast amount of parallelism present in these networks.

2. STRATEGIC DIRECTIONS

Based on the position statements of the group members, the Programming Languages working group identified five strategic directions. They represent a mix of continuing, long-term research going on within the field, and more recent directions sparked by changes in the larger world of computing. In the past, the programming of single computers and computers in local-area networks has been the dominant programming task. In the coming years, programs will be needed for global computing, domain-specific computing, embedded systems, large-scale systems, and more. These application areas almost certainly require new programming concepts. For example the heterogeneous, distributed and dynamic nature of global computing networks raise issues to do with configuration, coordination, security, and exploitation of interprocessor parallelism. In the context of embedded systems, performance predictability and fault tolerance pose new challenges.

The identified strategic directions are:

(1) *Distributed Computing*: One of the recurrent themes throughout the position statements is distributed computing. This poses new challenges in all areas of programming language research: how do we design languages for such systems that have firm semantic foundations, support the safe programming of distributed systems and utilise the vast amount of parallelism offered by the networks. Persistence and support for transactions are examples of areas of increasing importance. A common feature of distributed systems, which is also of independent interest, is mixed language working – "integration of programming paradigms" is an important theme.

(2) *Incrementality, Modularity and Abstraction*: Software systems are long-lived and must survive many modifications in order to prove useful over their intended life span. The primary linguistic mechanisms for managing complexity are modularity (separating a system into parts) and abstraction (hiding details that are only relevant to the internal structure of each part). A challenge for future language design is to support modularity and abstraction in a manner that allows incremental changes to be made as easily as possible. Object-oriented concepts have much to offer and are the topic of much on-going investigation.

(3) *Generic Formalisms, Integration of Tools and Techniques*: Many of the formalisms that we work with have been developed in the context of specific problem domains; there is the need to develop these further to provide the basis for generic formalisms. At the more systems-oriented end of the spectrum, there is the need to integrate the tools and techniques that are being produced by individual research activities into "real" systems. Part of the work in this direction also involves developing a better understanding of the relationship between different approaches.

(4) *Correctness, Efficiency, Engineering, and Pragmatics*: The goal of programming language research must be to define languages and techniques which improve the quality of software. The notion of *quality* is multi-faceted but must

include programmer productivity, verifiability, reliability, maintainability, and efficiency. This theme has assumed greater importance following a number of well-publicised disasters, such as the Pentium chip and Ariane-5.

(5) *Education and Technology Transfer*: Although only few of the position statements mention this aspect, the validity of all programming language research hinges on our ability to educate others and transfer our technology into practical systems. For example, we can consider TCL to be a failure of education and technology transfer because the language does not even have a syntax suitable for presentation by a grammar, and Java a success, since the industrial group responsible for its design used research developments of the past decade to great advantage.

3. RESEARCH THEMES

In addition to identifying the strategic directions mentioned above the working group has also identified *research themes* within the areas of Semantics, Type Systems, Program Analysis, Program Transformation and Implementation. We summarise the themes below and give pointers to the position statements discussing them in more detail. The position statements included in this collection are:

—Luca Cardelli: Global Computation

—Charles Consel: Program Adaption based on Program Transformation

—Patrick Cousot: Program Analysis: The Abstract Interpretation Perspective

—Michael Hanus: Integration of Declarative Paradigms: Benefits and Challenges

—Robert Harper and John Mitchell: ML and Beyond

—Daniel Le Métayer: Program Analysis for Software Engineering: New Applications, New Requirements, New Tools

—Flemming Nielson: Perspectives on Program Analysis

—Martin Odersky: Challenges in Type Systems Research

—Robert Paige: Future Directions in Program Transformations

—Alberto Pettorossi and Maurizio Proietti: Future Directions in Program Transformation

—John Reynolds: Beyond ML

—Jon G. Riecke: Semantics: The Description of Computational Structures

—Barbara Ryder: A Position Paper on Compile-time Program Analysis

—David A. Schmidt: On the Need for a Popular Formal Semantics

—Dennis Volpano: Provably-Secure Programming Languages for Remote Evaluation

—Reinhard Wilhelm: Program Analysis – A Toolmaker's Perspective

3.1 Semantics

The working group has identified four research themes in Semantics:

(1) Language design.

(2) Foundations.

(3) Paradigm integration.

(4) New applications.

These themes are primarily addressed in the position statements by Cardelli, Cousot, Hanus, Riecke, and Schmidt. The position statements of Harper and Mitchell, Nielson, Reynolds and Ryder also address issues related to semantics.

Theme (1) concerns the influence that semantics has had on programming language design. Riecke enumerates a number of successes but also highlights some outstanding problems for current semantic formalisms. Nielson also addresses the role that semantics should play in advising language designers about a judicious choice of language constructs. Theme (2) is mainly addressed by Riecke and Schmidt. Riecke identifies a number of new directions for research, including reactive systems, that will require further foundational work. Schmidt appeals for a "popular" style of semantics that extends BNF and carries little formal overhead – a formalism that can be used by novice programmers to reason about their programs. Both Schmidt and Cousot advocate that abstract interpretation provides the right link between different styles of semantics. Theme (3) is explicitly addressed by Hanus, who discusses the integration of functional and logic programming languages; it also implicitly addressed by Cardelli, who discusses global computation. Finally, Theme (4) is addressed by a number of the papers but notably by Cardelli and Riecke; the issues raised include models for global computation, reactive systems, programming in the large and distributed programs.

Type Systems

The working group has identified four research themes in Type Systems:

(1) Types for objects.

(2) Type based compilation.

(3) Type aware programming environments.

(4) New applications.

These themes are primarily addressed in the position statements by Harper and Mitchell, Odersky, and Volpano. The position statements of Cardelli, Reynolds, Riecke, and Nielson also address issues related to type systems.

Theme (1) is discussed by Harper and Mitchell in the context of extending Standard ML with objects. Theme (2) concerns the exploitation of type information in the process of compilation and it is also addressed by Harper and Mitchell. A related issues is to extend type systems to carry information about the use of resources; this is discussed by Odersky and Reynolds; more generally the relationship to program analysis is touched upon by Odersky and Nielson. One issue related to theme (4) is addressed by Volpano who discusses the use of type systems for ensuring the security of programs in languages with remote procedure calls; Cardelli advocates more generally for type systems for distributed systems.

3.2 Program Analysis

The working group has identified four research themes in Program Analysis:

(1) Unification of different techniques.

(2) Realisation of program analyses.

(3) Choice of program analyses.

(4) New applications.

These themes are primarily addressed in the position statements by Cousot, Le Métayer, Nielson, Ryder, and Wilhelm. The position statements of Odersky and Volpano also addresses issues related to program analysis.

Theme (1) is concerned with estabilishing a better understanding of the strengths and weaknesses of the different techniques to program analysis. This is discussed in detail by Nielson and Ryder and touched upon by Le Métayer and Wilhelm. Theme (2) is concerned with the development of generic tools for program analysis. Wilhelm argues that the time is ripe for addressing this challenge. Le Métayer and Ryder point out that user-interaction is needed in these tools if they are not just to be hidden components in compilers. Theme (3) is addressed by Ryder who raises issues as how do one select the appropriate analysis for a given application, how do we trade cost for precision etc. The issues is also discussed by Cousot in the context of designing abstract domains (or algebras) for abstract interpretations. Related to theme (4), Le Métayer argues that program analysis has a role to play outside optimising compilers, namely in software engineering, and discusses the new demands such applications will put on the program analysis techniques in general. It is argued by Nielson and Ryder that program analysis should have an impact on programming language design to ensure acceptable behaviour of software on, say, heterogeneous and physically distributed computer networks.

3.3 Program Transformation

The working group has identified four research themes in Program Transformation:

(1) Automatic tools.

(2) Algorithm development and design.

(3) Foundations.

(4) New applications.

These themes are primarily addressed in the position statements by Consel, Paige, and Pettorossi and Proietti.

The first theme is addressed by all of the position statements. Consel and Paige focus mainly on partial evaluation, although Paige discusses finite differencing and data structure selection by real-time simulation as well. Petterossi and Proietti emphasise the fold/unfold style of program transformation. Theme (2) concerns the development of new algorithms using transformational programming techniques; Paige identifies the better integration of algorithm design and program development as one of the research directions for transformational programming. Paige and Petterossi and Proietti discuss the semantic foundations of program transformation. Finally, Theme (4) is addressed by Consel; *program adaptation* is a collection of techniques which allow generic programs to be adapted for a particular context of use – the techniques include partial evaluation, run-time code generation and data specialisation. Petterossi and Proietti identify the "lifting" of program transformation techniques to collections of program modules (rather than single stand-alone programs) as a pressing problem.

3.4 Implementation

The working group has identified five research themes in Implementation:

(1) Compilation for distributed systems.
(2) Optimization.
(3) Compiler correctness.
(4) Memory management.
(5) Generic tools.

These themes are not discussed in detail by any of the position statements below. Issues related to theme (1) are briefly discussed by Harper and Mitchell, issues related to theme (4) are touched upon by Reynolds, and issues related to theme (2) and (5) are also discussed by the position statements on program analysis and program transformations.

4. CONCLUSION

The position statements included in this collection provide a good overview of the range of the discussions of the working group. The interested reader is urged to consult [1] for a more detailed presentation of the findings of the group. A number of other groups also discussed programming language issues: most notably, there were groups on Object-Oriented Programming; Software Engineering and Programming Languages; and Human Computer Interaction. The reports of these other groups may also be found in the December 1996 volume of Computing Surveys.

REFERENCES

[1] Chris Hankin, Hanne Riis Nielson, Jens Palsberg 1996. Strategic Directions for Research on Programming Languages. In *Computing Surveys* Vol. 28(4), December 1996.