Paul Klint, Thomas Reps,
Gregor Snelting (editors):

Programming Environments

Dagstuhl-Seminar-Report; 34
9.-13.3.92 (9211)

ISSN 0940-1121

Tel.: +49-6871 - 2458
Fax: +49-6871 - 5942

Das Internationale Begegnungs- und Forschungszentrum für Informatik (IBFI) ist eine gemeinnützige GmbH. Sie veranstaltet regelmäßig wissenschaftliche Seminare, welche nach Antrag der Tagungsleiter und Begutachtung durch das wissenschaftliche Direktorium mit persönlich eingeladenen Gästen durchgeführt werden.

Bezugsadresse: Geschäftsstelle Schloß Dagstuhl
Informatik, Bau 36
Universität des Saarlandes
W - 6600 Saarbrücken
Germany
Tel.: +49 -681 - 302 - 4396
Fax: +49 -681 - 302 - 4397
e-mail: office@dag.uni-sb.de

Report on the Dagstuhl Seminar on

Programming Environments

organized by

Paul Klint (Centrum voor Wiskunde en Informatica Amsterdam)
Thomas Reps (University of Wisconsin-Madison)
Gregor Snelting (Technische Universität Braunschweig)

March 9–13, 1992

Introduction

The goal of research on programming environments is to increase the productivity of the software development process by providing better tools for creating, manipulating, and understanding programs. Originating in the late 70's (and then mainly concerned with language-specific editors), the field has grown considerably, covering a wide variety of software development activities today. Recently, some systems and tools originating in academia have become available commercially.

During the week of March 9–13, 1992, a workshop on programming environments was held at the International Conference and Research Center for Computer Science at Dagstuhl Castle. The workshop brought together thirty active researchers in the area. It focused on new and recent developments, with emphasis on the following topics:

- Single-programmer language-based environments
- Generation of programming environments
- Tool-integration mechanisms
- Debugging
- Merging and restructuring of systems
- Projection and understanding of systems
- Incremental computation.

The intention of the workshop was to stimulate intellectual ferment among the participants. The workshop's format made it possible for participants to give more in-depth presentations than is customary at conferences. The list of researchers invited contained people from both the United States and Europe, covering academia as well as industry. In addition to a number of prominent researchers, it included a number of students and recent Ph.D.s.

This report contains the abstracts of the talks presented at the workshop. They are direct transcripts of the hand-written abstracts that the participants entered into the Dagstuhl seminar book – only only light copy-editing has been performed.

In addition to the talks, several systems have been demonstrated, namely the Wisconsin Program Integration Ssystem, the FIELD system, the Synthesizer Generator, the SAMPλE language and environment, the ASF+SDF meta-environment, the partial evaluator Similix, and the PSG system.

Dagstuhls excellent facilities, as well as the nice setting, made this a particular interesting and stimulating week. Special thanks go to the Dagstuhl personnel for taking good care of us.

Paul Klint
Thomas Reps
Gregor Snelting

Abstracts

Program Visualization: Where We Go From Here

Steven P. Reiss
Brown University

Modern technology allows us to view our programs through the visual representation that we normally use to describe and understand them. Our experience with program visualization started with multiple views in the program development system Pecan and with algorithm animation in Balsa and later Tango. It continued with the programming system Garden and has culminated with the Field programming environment. Building on these efforts, we are currently developing a new visualization system that will take advantage of today's technology to provide a flexible interface to a variety of program visualizations. The system will allow the programmer to define visualizations as abstractions using queries over an object-oriented database of information about the program. These abstractions can then be visualized and browsed using easily defined type-based mappings and a generic filtering mechanism.

Tool Integration Technologies Through the 90's

William H. Harrison
IBM Thomas J. Watson Research Laboratory

The problem of integrating separately written tools so that they work together cooperatively is recognized as a key issue in CASE frameworks. The existing model for building tools emphasizes separate large components, sharing rigid models of the data they manipulate. In order to construct tools which can be more readily mixed and which can meet the demand for more well-integrated visual environments, we see trends toward a finer grain-size for both data elements and control elements and from a procedural control-flow approach toward a compositional, object-oriented tool construction.

The Computational Complexity of Incremental Algorithms

Thomas Reps*
University of Wisconsin Madison

A common way to evaluate the time complexity of an algorithm is to use asymptotic worst-case analysis and to express the cost of the computation as a function of the size of the input. However, for an incremental algorithm this kind of analysis is often not very informative. (By an "incremental algorithm", we mean an algorithm that makes use of the solution to one problem instance to find the solution to a "nearby" problem instance.) When the cost of the computation is expressed as a function of the size of the (current) input, many incremental algorithms that have been proposed run in time asymptotically no better in the worst case than the time required to perform the computation from scratch. Unfortunately, this kind of information is not very helpful if one wishes to compare different incremental algorithms for a given problem.

We have explored a different way to analyze incremental algorithms. Rather than express the cost of an incremental computation as a function of the size of the current input, we measure the cost in terms of the sum the sizes of the *changes* in the input and the output. This change in approach allows us to develop a more informative theory of computational complexity for incremental problems.

In our work, we have developed new upper-bound results as well as new lower-bound results. First, three problems – the single-sink shortest-path problem with positive edge weights, the all-pairs shortest-path problem with positive edge weights and the circuit-value problem – are shown to have bounded incremental complexity (*i.e.*, incremental complexity bounded by a function of the sum of the sizes of the changes in the input and the output). The single-sink shortest-path problem with positive edge weights and the all-pairs shortest-path problem with positive edge weights are shown to be P-time incremental; the circuit-value problem is shown to be Exp-time incremental. We have also established a number of lower bounds with respect to a class of algorithms called the locally persistent algorithms. We demonstrated the existence of a non-incremental problem (*i.e.*, a problem for which no bounded locally persistent incremental algorithm exists). We also demonstrated that a number of other problems, including the closed-semiring path problems in directed graphs and the meet-semilattice data-flow analysis problems, are non-incremental with respect to the class of locally persistent algorithms.

---

joint work with G. Ramalingam

ObjectMath - A Very High Level Programming Environment and Language for Equation-based Modeling in Scientific Computing.

Peter Fritzson*
Linköping University

We present the first programming environment and language which integrates inheritance within a computer algebra language. This environment and language, called ObjectMath (Object Oriented Mathematical Language for Scientific Computing), is currently being used for applications in advanced mechanical analysis, but is generally applicable to other areas. Using ObjectMath, it is possible to model classes of equation objects, to support inheritance of equations, and to solve systems of equations. The ObjectMath environment is designed to handle realistic problems. This is achieved by allowing the user to specify transformations and simplifications of formulae in the model, in order to arrive at a representation which is efficiently solvable. Such algebraic transformations can conveniently be supported since ObjectMath models are translated into the Mathematica computer algebra language. When necessary, equations can be transformed to C++ code for efficient numerical solution. The motivation for this work is the current low-level state of the art in programming for scientific computing. Much numerical software is still being developed the traditional way in FORTRAN. This is especially true in application areas such as machine elements analysis, where complex non-linear problems are standard.

We also report some experience from successful use of a prototype version of ObjectMath in representing and solving a 3-dimensional example made of 200 equations describing a rolling bearing. The re-use of equations through inheritance reduced the model by a factor of two, compared to a direct representation of the model in the Mathematica computer algebra language.

Graphs as Central Data Structures in a Database Design Environment

Gregor Engels†
Leiden University

The talk illustrates that graphs are a well-suited data structure to be used for the modelling, *i.e.*, external representation, as well as for the internal representation of database schemas.

We present the database design environment CADDY (Computer-Aided conceptual Design of non-traditional Databases) which offers an integrated tool set for editing, and prototyping a conceptual database schema. The theoretical background is given by a semantically well-defined data model which enables the specification of the static structure of an application area by an Extended Entity-Relationship diagram and the specification of the dynamic behaviour by temporal integrity constraints and extended data flow graphs. The tool set of CADDY comprises language-sensitive editors for all these specification paradigms and prototyping

* joint work with Dag Fritzson, Lars Villand, and Johan Herber

† joint work with U. Hülsmann, P. Löhr-Richter

tools like a query interpreter or a graphical database browser to join experiences with partly developed conceptual database schemata.

All developed documents within CADDY are internally represented by attributed abstract syntax graphs. These are extensions of abstract syntax trees, where in addition to context-free interrelations also context-sensitive interrelations are expressed by nodes and edges. The classes of these syntax graphs are specified by an operational specification approach which is based on programmed graph representation systems. These graph grammar specifications serve as guideline for an efficient implementation of all CADDY-tools.

Object–Oriented Techniques for Replacing Components on the Fly in a Running Software System

Manfred Stadel
Siemens Nixdorf

In an object oriented programming language virtual classes can be used to define interfaces. Implementations are then subclasses of the corresponding virtual classes. This also supports the coexistence of several implementation variants for the same interface. Clients should now call methods only via the virtual superclass.

To replace an implementation by a new version in a running program we need a dynamic link loader. The dynamic link loader loads the new code and class descriptor and links it as a subclass of the already existing virtual superclass. Since only the link loader knows the address of the new class, creation of objects can be done by the link loader, *i.e.*, the link loader provides a create method to create objects of a new class.

If for a certain class objects are frequently created and destroyed, after a new implementation has been loaded further objects are created from the new implementation and we can wait until all objects of the old implementation have died out. Then the code of the old implementation can be unloaded.

This will not work for long-living objects. Such objects must be converted to objects of the new implementation. To do this each class should provide a method

```
clone(new_version:  virtual_interface_class) is
  require -- new version has just been created
  do ...
  ensure -- new version is in the same state as current
end;
```

"clone" copies its internal state to new_version by using the (constructor) methods provided by the class. After cloning, the new_version can be used instead of the current one and the current one can be discarded.

A programming environment should support this style of programming, *e.g.*, it should automatically generate dynamic link loaders, garbage collectors to unload unused code, and should clone long-living objects.

Current Developments in the Synthesizer Generator

Tim Teitelbaum
Cornell University & GrammaTech, Inc.

The Synthesizer Generator is a system for generating language-sensitive editing environments from high-level descriptions. It has been licensed as of now to over 330 sites worldwide, 50%/50% US/non-US, 50%/50% university/industry. Current work enhancing the Synthesizer Generator at GrammaTech is directed toward providing:

- Better Support for Tool Integration and Multilingual Environments
  - *an editor command language:* a uniform input abstraction allowing full control and command extensibility of an editor by external tools, with support for the editor-server model and batch use.
  - *Events:* a uniform output abstraction allowing full control of data exchanges with external tools through broadcast messages.
  - *Message server:* Adoption of emerging industrial selective-broadcast messaging platforms.
  - *Data Exchange Representations:* Improved compact linear representations of structural data and generation of code stubs for use by external tools exchanging such data.
  - *Imported and Exported Attributes:* Exchange of attribute data among disjoint objects in the same process, different processes, or data repositories.
  - *Annotations:* Opening of the object base for associated data of external tools.
  - *Grammar modules:* Support for multilingual languages within one editing process.
- Better Engineered Editing Interfaces
  - *Strengthening the Textual Viewpoint:* adoption of an editing paradigm more compatible with standard text-editing paradigms. A seamless integration of text and structure editing.
- A better Specification System
  - *Higher-Order Attribute Grammars:* Support for macros and more powerful prettyprinting.
  - *More Powerful Transformation capabilities*
  - *Compact Notations*
  - *Separate Compilation:* maturing environments faster to generate.

A demonstration of the new system under development was presented.

How We Can Work Together

Mark Wegman
IBM Thomas J. Watson Research Center

I started out by describing environments work at Watson. This consists of work on tools and an infrastructure that makes the whole greater than the sum of the parts. The infrastructure consists of components like an object-oriented data base to store all the data passed around, control systems, and user interfaces. I mentioned two tools – an information-retrieval system, and a program-transformation tool.

I then discussed that tools or interfaces separately were of little value and that taking a big step forward requires re-doing many components. No one group can afford to take as radical a step as several groups together could.

Based on this we discussed the creation of the Environment Consortium. If groups create components with well-defined interfaces they can put these components on the net. Different groups could provide competing components.

Some problems that need to be addressed are licensing, dependencies on other groups, and the fact that de facto standards may inhibit progress.

Fine-Grain Implementation of Algebraic Specifications

Emma van der Meulen
CWI Amsterdam

An incremental implementation can be derived from algebraic specifications belonging to the subclass of well-presented primitive recursive schemes with parameters. The implementation is based on the result by Courcelle & Franchi-Zanettaci that this class of specifications is equivalent to strongly noncircular attribute grammars. Attributes are associated with "incremental" functions and attribute update strategies are transferred to be used in incremental term rewriting. *E.g.*, typecheck functions can typically be implemented incrementally.

Considering symbol tables as aggregate (attribute) values, however, hinders the incremental evaluation. In this presentation we addressed the problem of expensive operations on tables. The solution we presented is to apply the incremental technique for operations like typechecking to operations on auxiliary data as well. Thus we obtain fine-grain incremental implementations. This fine-grain incrementality can be derived from a subclass of algebraic specifications that we will call layered primitive recursive schemes.

Automated Assistance for Program Restructuring

William Griswold
University of California, San Diego

Maintenance tends to degrade the structure of software, ultimately making maintenance more costly. At times, then, it is worthwhile to manipulate the structure of a system to make changes easier. However, manual restructuring is an error-prone and expensive activity. By separating structural changes from other maintenance activities, the semantics of a system can be held constant by a tool, assuring that no errors are introduced by restructuring. To allow the maintenance team to focus on the aspects of restructuring and maintenance requiring human judgment, a transformation-based tool can be provided – based on a model that exploits preserving data flow-dependence and control flow-dependence – to automate the repetitive, error-prone, and computationally-demanding aspects of restructuring. A set of automatable transformations is introduced; their impact on structure is described, and their usefulness is demonstrated in examples. A model to aid building meaning-preserving restructuring transformations is described, and its realization in a functioning prototype tool for restructuring Scheme programs is discussed.

Program Dependance Graphs for the Rest of Us

Robert A. Ballance*
The University of New Mexico

Fully general algorithms for computing control dependence information construct both the reverse control flow graph and its domination tree. The domination tree encodes the global program flow information needed to compute a control dependence graph (CDG).

However, for most programs, the postdominator information can be inferred directly from the abstract syntax tree. In this talk, we present a succession of algorithms for computing CDG's from the AST. The first algorithm handles structured programs consisting of single-entry, single-exit control constructs. The language is then extended to include C-style exit constructs (break, continue, return) and the algorithm is extended to handle such constructs. Finally, in the presence of arbitrary control flow, we show how to derive a reverse control flow graph in which each node will be a region in the final CDG. From the reverse control flow graph, conventional algorithms can be used to compute the CDG. Simple algorithms for data flow analysis using the CDG will also be discussed.

* Work performed in collaboration with Arthur B. Maccabe

Language-based Document Processing

Isabelle Attali*
INRIA Sophia Antipolis

This work proposes an application of programming environment generation to structured document manipulation. We use Centaur as a formal tool to model and implement logical and physical structure, logical editing and layout processing, document analysis, re-use applications and document conversion for a sample class of documents: scientific articles including equations and figures. To make connections with real document systems, we choose to give two particular external forms to the logical structure: Tioga source and LaTeX source.

From the specifications of the logical and physical structure of the article document class on one hand, and, on the other hand, the specification of the layout processing (including its semantics according to the Tioga or the LaTeX layout) and other semantic tools, the Centaur system automatically generates structured environments for Tioga and LaTeX documents and conversions between them.

SAMPλE – An Environment for Lazy Functional Programming

Stefan Kaes
Technische Hochschule Darmstadt

The programming language SAMPλE and its programming environment are the outcome of the DFG-project "Environment for constructive specifications" at the Technical University of Darmstadt. The primary goals of the project were the design of a high-level functional programming language and the implementation of a programming environment to support the development of large systems written in that language.

SAMPλE is a modern language with a simple, Modula-2–like module concept. The language features non-strict, higher-order functions and data constructors, pattern matching and fully static typing, which prevents type errors at runtime. The type system extends parametric polymorphism by overloading, implicit coercions, and recursive types, based on the theory of *constrained types*.

SAMPλE is embedded into a language dependent, fully integrated interactive programming environment, consisting of a syntax-oriented editor, a type-inference component, an interpreter, a compiler, a debugger, and a library system. All components are controlled by a window-oriented, menu-driven user interface. Management of module dependencies, type inference and recompilation is fully automated. Special care has been taken to provide facilities for interactive type error correction and debugging of lazy functional programs.

* Joint work with Dennis Arnon and Paul Franci-Zanettaci

Pregmatic - A Generator for Incremental Programming Environments

Mark van den Brand
Katholieke Universiteit Nijmegen

The main goal of this research was the development of a complete system for the generation of incremental interactive programming environments from extended Affix Grammars. Furthermore we did not want to extend or restrict the given formalism. The research objectives were incremental evaluation, affix-directed parsing, ambigous context-free grammars, implicit derivation of unparsing rules, and derivation of an execution tool. We concentrate on the integration of the first three points, and now only the affix-directed parsing and ambiguous grammars will be discussed.

The principle of affix-directed parsing for semantic-directed parsing was not yet used in a programming environment. The benefits of affix-directed parsing are:

- detecting typechecking errors as soon as possible
- parsing context-depending language constructs, such as the offside rule in Miranda
- disambiguate (local) ambiguous context-free grammar rules.

The drawback is the reduction in the incrementality in case the input sentence contains an error. Affix-directed parsing is obtained by alternating parsing and propagation.

Ambiguous grammars increase the flexibility of specification and allow us to introduce a new language construct: *untyped placeholder*. The multiple subtrees found in case of ambiguity are combined into a 3-dimensional tree.

Term Rewriting as Unifying Principle in the ASF+SDF Meta-Environment

Paul Klint
CWI & UvA Amsterdam

Have Programming Environment Generators a future? The success of current systems like the Synthesizer Generator shows that the generation of programming environments from formal language definitions is feasible. However, now writing large language definitions becomes the bottleneck. We have been working on a "meta-environment" that solves (parts of) the software engineering problem for language definition development.

In the talk I have presented ASF+SDF: an integrated formalism for defining both syntax and semantics. Interesting aspects are:

- the fixed mapping between concrete and abstract syntax.
- the close integration of syntax and semantics.

After presenting some examples and showing how language definitions can be developed interactively in the ASF+SDF meta-environment, I have concentrated on the main topic of the talk, the unifying role of term rewriting in our system:

(i) Simple operational semantics where the editing paradigm is equal to the execution paradigm (*i.e.*, term replacement);

(ii) Syntactic functions (*e.g.* the constructor for an assignment statement) and auxiliary functions (*e.g.* lookup) are treated uniformly. This permits the use of incremental evaluation techniques at all these levels.

(iii) Generic debugging can be obtained by interrupting the rewrite process when the current redex matches a given pattern. In this way, language-specific breakpoints can be obtained.

A notion of "origins" can be defined which effectively summarizes the history of the rewriting process. It constructs breakpoints from subterms that occur during rewriting to the original term. This can be used for animation and precise error indications.

(iv) Finally, I showed how the notion of terms helps to solve the problem of specifying how functions in the language definition can be applied to the current "program" in a (generated) syntax-directed editor. We propose a generic approach where
- "buttons" in the interface can be enabled/disabled depending on specified conditions;
- on activation of a button, specified subterms from various windows are assembled into a new term which is then reduced. The result can be put in a specified place in the user interface (*e.g.* a new window, the current focus, etc.).

To conclude, *yes* Programming Environment Generators have a future, but then we need to solve the "language-definition engineering problem". Our system is one step in that direction.

Empirical Experience with a New Interprocedural Pointer Aliasing Approximation Algorithm

Barbara Ryder
Rutgers University

Compile-time analysis of C programs is useful for optimization and parallelization of code, semantic change analysis of evolving systems, and tools for debugging and testing of programs. Aliasing occurs when more than one name exists simultaneously for the same memory location. Existing aliasing algorithms for pointers are quite imprecise. The extensive use of pointers in languages like C has rendered such analyses impractical. Aliasing induced through pointers also has been studied in dealing with dynamically growing structures in parallelization of codes.

Our new aliasing analyses (i) have determined the theoretical complexity of intraprocedural and interprocedural aliasing problems induced through pointers and (ii) have resulted in a new approximation technique for interprocedural, program-point-specific aliases. In empirical experiments our method seems much more precise than previous techniques. We will explain our approach, present our empirical experiments and discuss the utilization of these methods in our semantics change analysis project.

This work was done jointly with Dr. William Landi and appears in his dissertation (November 1991).

A Rewriting Semantics for Imperative Programs and its Use in Programming Environments

John H. Field
IBM Thomas J. Watson Research Center

We present Pim, a formal system for reasoning about programs written in realistic imperative programming languages. Pim consists of a *term language* and a nested set of *equational logics*. The core of Pim is a confluent *rewriting system* formed by orienting the equations of a subsystem designated by Pim$^{\rightarrow}$. The full equational logic (*i.e.* Pim) adds rules for reasoning about observational congruence of Pim terms with respect to the semantics defined by Pim$^{\rightarrow}$, and includes rules for performing inductive reasoning.

Pim is intended to be used to define the operational semantics of imperative programming languages incorporating "problematic" constructs such as pointers, arrays, procedures with various calling conventions, indefinite loops, and arbitrary control flow. Pim is distinguished in particular by its ability to reason about the address-generating constructs that result in aliasing - a phenomenon that often bedevils many analysis algorithms.

Our principal application of Pim is as an intermediate representation for *semantics-based* programming tools. By "semantics-based", we mean capable of taking advantage of the program's *behavior*, rather than just its syntax. We illustrate the utility of Pim in this context by showing that program *slicing* in the presence of pointer constructs can be performed easily by partially evaluating the program's corresponding Pim representation, then performing a simple graph traversal.

A Visual Environment for Distributed Object-Oriented Multi-Applications

Daniel Yellin*
IBM Thomas J. Watson Research Center

We describe a new graphical, object-oriented environment for distributed interacting multi-applications. The end-user's view is a collection of active objects represented as windows or icons decorated with input and output slots, and plug socket connectors. These active objects represent a particular user's view of those applications on the network currently accessible to him. Objects can be created, destroyed, deposited in other objects, and emitted from other objects. Objects can be packaged together to form composite objects. Composite objects can be packaged into single objects and later unpackaged to reveal their internal detail. A prototype of this environment has been implemented in Hermes.

* This work has been done jointly with Robert Strom (IBM).

Origin Tracking

Frank Tip
CWI Amsterdam

Origin tracking is a method for relating (partially) rewritten terms to an initial term, given a conditional rewriting system. When a rewrite rule is applied, relations between "equal" subterms in a redex and contractum are automatically derived from the syntactic shape of that rule. The origin of a term, which is a set of related subterms in the initial term, is defined as the transitive and reflexive closure of this relation.

Whereas rewriting according to an uncontidional TRS must be considered as a linear sequence of rewrite steps, this is no longer the case when conditions are allowed. Rewriting according to a CTRS is modeled as a tree of reduction sequences, containing one node per normalization of a condition side. Consequently, in addition to the previously described relations for unconditional TRS's, relations between terms in different reduction sequences need to be defined.

Some properties of the origin function are described, and the effects of allowing lists, and of representing terms as DAG's are considered.

An efficient implementation of origin tracking is presently based on several optimizations of the basic algorithm. Finally, several applications of origin tracking are considered: visualization of execution, generic debugging, and localization of errors.

Demand-Driven Data Flow Analysis for Program Debugging

Jong-Deok Choi
IBM Thomas J. Watson Research Center

[no written abstract available]

An Environment for Developing by Transformation

Burkhart Wolff
Universität Bremen

The main goal for this research is the construction and foundation of a transformation- and software-development system based on the experiences made with the PROSPECTRA system.

Three major points were outlined:

- The theoretical background (the object language SPECTRAL and its relation to the transformation system)
- The methodological background (outlining the methodological framework of KORSO, the BMFT-founded research programme for correct software
- The technical background (the transformation engine)

Because of the wish to use incremental evaluation, the transformation engine is based on attributes.

For this purpose, the concept of attribute specifications has to be extended by well-known decomposition theorems of algebraic specifications, namely

- Signature morphisms on the abstract syntax ("Views")
- Parametrization (of "Layers" with sorts, *functions, and even attributes*)
- Extension (hierarchical organization of layers)

Inference-Based Support for Programming in the Large

Gregor Snelting
Technische Universität Braunschweig

The experimental NORA Inference Based Software ENvironment* can handle incomplete, missing or inconsistent information about a software project. NORA comprises an incremental interface checker for software component libraries, support for polymorphic component reuse, an inference-based interactive configuration system, and retrieval algorithms based on usage patterns. We make heavy use of automated deduction techniques such as order-sorted unification or AC1 unification. All tools are generic and parameterized with language-specific information. NORA is an interactive environment currently under development; it can complete partial information by itself and detect errors earlier than conventional tools.

* NORA is no real acronym. It is a drama by the Norwegian writer H. Ibsen.

The Self-Applicable Partial Evaluator Similix

Anders Bondorf
DIKU, University of Copenhagen

Similix is an autoprojector (self-applicable partial evaluator) for a higher-order subset of the strict functional language Scheme. We present and demonstrate some applications of Similix: generating deterministic finite automata from regular expressions by partial evaluating a regular-expression matcher written in Scheme; compiling a lazy functional language into Scheme by partially evaluating an interpreter (written in Scheme) for the lazy language.

Partial evaluation does not always give good results. One often has to rewrite the source program into a form that makes more expressions static, that is, reducible at partial evaluation time. We illustrate this by the regular-expression matcher example.

A Logical Framework as the Basis for Language Definition

Frank Pfenning
Carnegie Mellon University

We begin with a brief review of the LF logical framework and show how it can be used to specify the (abstract) syntax and (operational) semantics of programming languages. We then introduce Elf, a logic programming language providing an operational semantics to LF, *e.g.*, to implement an interpreter or type checker. Finally we sketch how one can also employ the same framework to implement the meta-theory of deductive systems and thus prove properties of the programming languages which have been axiomatized. The language has been implemented in Standard ML and is available via anonymous ftp on the Internet.

A Tool for Software Migration

Mooly Sagiv
IBM Scientific Center Haifa

[No written abstract available. This is due to the fact that the work described is still confidential, and Mooly received clearance for his oral presentation only immediately prior to his talk. His system transforms machine-dependent PL/I programs (such as operating system code) into equivalent PL/I programs which are to run on a new architecture. The problem is not the overall program structure, but low-level details like word sizes, overlays, and strings used as pointers (yes, in PL/I this is possible). Types and intended uses of objects are inferred from the abstract syntax tree, and rules specify how to transform certain constructs. The method is not completely automatic and very slow, but it works. GS]

Logical Views, Feature Contexts, and Animation of Object-Oriented Design

John Shilling
Georgia Institute of Technology

Logical Views is a technology for interactive tools in a software-development environment. The paradigm extends the object-oriented paradigm in three steps:

1) Allow multiple interfaces to a class
2) Allow visibility control over instance variables by interface
3) Allow multiple activations of the same interface

This allows tools to share data without interfering.

Feature contexts are a way to tie concise design decisions to their distributed implementation. Features can be extracted and replaced. Feature interaction is detected. This system was implemented on top of Gandalf.

Groove allows animation and captures object-oriented design protocols.

Dagstuhl-Seminar 9211

Participants

Isabelle **Attali**
INRIA
Sophia Antipolis
2004 Route de Lucioles
F-06565 Valbonne Cedex
France
ia@trinidad.inria.fr
tel.: +33-93657804

Rolf **Bahlke**
Software AG
Uhlandstr. 12
W-6100 Darmstadt
Germany
tel.: +49-6151-921706

Robert A. **Ballance**
University of New Mexico
Department of Computer Science
Albuquerque NM 87131
USA
ballance@unmvax.cs.unmi.edu
tel.: +1-505-277-6509

Anders **Bondorf**
University of Copenhagen
Dept. of Computer Science / DIKU
Universitetsparken 1
2100 Copenhagen 0
Denmark
anders@diku.dk
tel.: +45-31396466

Mark v.d. **Brand**
Katholieke Universiteit Nijmegen
Department of Informatica
Toernooiveld
NL-6525 ED Nijmegen
The Netherlands
mark@cs.kun.nl
tel.: +31-80-65 32 96

Jong-Deok **Choi**
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598
USA
jdchoi@watson.ibm.com
tel.: +1-914-784-7961

Gregor **Engels**
University of Leiden
Department of Computer Science
P.O. Box 9512
NL-2300 RA Leiden
The Netherlands
engels@rulwi.LeidenUniv.nl
tel.: +31-71-277069

John H. **Field**
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598
USA
jfield@watson.ibm.com
tel.: +1-914-784-6650

Peter **Fritzson**
Linköping University
Dept. of Computer and
Information Science
S-58183 Linköping
Sweden
paf@ida.liu.se
tel.: +46-13-281484

William **Griswold**
University of California at San Diego
Dept. of Computer Science and
Engineering
La Jolla CA 92093
USA
wgg@cs.ucsd.edu
tel.: +1-619-534-6898

William **Harrison**
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598
USA
harrisn@ibm.com
tel.: +1-914-784-7631

Stefan **Kaes**
Technische Hochschule Darmstadt
Praktische Informatik
Magdalenenstr. 11c
D-6100 Darmstadt
Germany
kaes@pi.informatik.th.darmstadt.de
tel.: +49-6151-16-3414

Paul **Klint**
CWI - Mathematisch Centrum
Kruislaan 413
NL-1098 SJ Amsterdam
The Netherlands
paulk@cwi.nl
tel.: +3/-20-5924126

Wojtek **Kozaczynski**
Andersen Consulting
Center for Strategic Technology
Research
100 S. Wacker
Chicago IL 60606
USA
wojtek@andersen.com
tel.: +1-312-507-6682

Arun **Lakhotia**
University of Southwestern Louisiana
Center for Advanced Computer Studies
P.O. Box 44330
Lafayette LA 70504
USA
arun@cacs.usl.edu
tel.: +1-318-231-6766

Emma **van der Meulen**
CWI - Mathematisch Centrum
Kruislaan 413
NL-1098 SJ Amsterdam
The Netherlands
emma@cwi.nl
tel.: +31-20-592 4007

Frank **Pfenning**
Carnegie Mellon University
School of Computer Science
Pittsburgh PA 15213
USA
fp@cs.cmu.edu
tel.: +1-412-268-6343

Steven **Reiss**
Brown University
Dept. of Computer Science
Box 1910
Providence RI 02912
USA
spr@cs.brown.edu
tel.: +1-401-863-7641

Tom **Reps**
University of Wisconsin-Madison
Computer Sciences Department
1210 W. Dayton St.
Madison WI 53706
USA
reps@cs.wisc.edu
tel.: +1-608-262-1204

Barbara **Ryder**
Rutgers University
Dept. of Computer Science
Hill Center / Busch Campus
New Brunswick NJ 08903
USA
ryder@cs.rutgers.edu
tel.: +1-908-932-3699

Mooly **Sagiv**
IBM Scientific Center
Technion City
Haifa 32000
Israel
sagiv@haifasc3.vnet.ibm.com
tel.: +972-4-296-283

John **Shilling**
Georgia Institute of Technology
College of Computing
Atlanta GA 30332
USA
shilling@cc.gatech.edu
tel.: +1-404-894-7512

Gregor **Snelting**
TU Braunschweig
Institut für Programmiersprachen und
Informationssysteme
Gaußstraße 17
W-3300 Braunschweig
Germany
snelting@infbs.uucp
tel.: +49-531-391-7577

Manfred **Stadel**
SNI AG - STM SD 21
Otto-Hahn-Ring 6
W-8000 München 83
Germany
tel.: +49-89-636-45505

Tim **Teitelbaum**
Cornell University
Department of Computer Science
Upson Hall
Ithaca NY 14853
USA
tt@cs.cornel.edu
tel.: (607)255-7573

Frank **Tip**
CWI - Mathematisch Centrum
Kruislaan 413
NL-1098 SJ Amsterdam
The Netherlands
tip@cwi.nl
tel.: +31-20-5924007

Mark **Wegman**
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598
USA
wegman@watson.ibm.com
tel.: +1-914-984-7809

Burkhart **Wolff**
Universität Bremen
Fachbereich Mathematik/Informatik
Postfach 33 04 40
W-2800 Bremen 33
Germany
bu@informatik.uni-bremen.de
tel.: +49-421-218-4228

Daniel **Yellin**
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598
USA
dmy@watson.ibm.com
tel.: +1-914-784-7699

Zuletzt erschienene und geplante Titel:

J. Berstel , J.E. Pin, W. Thomas (editors):
Automata Theory and Applications in Logic and Complexity, Dagstuhl-Seminar-Report; 5, 14.-18.1.1991 (9103)

B. Becker, Ch. Meinel (editors):
Entwerfen, Prüfen, Testen, Dagstuhl-Seminar-Report; 6, 18.-22.2.1991 (9108)

J. P. Finance, S. Jähnichen, J. Loeckx, M. Wirsing (editors):
Logical Theory for Program Construction, Dagstuhl-Seminar-Report; 7, 25.2.-1.3.1991 (9109)

E. W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms, Dagstuhl-Seminar-Report; 8, 4.-8.3.1991 (9110)

M. Broy, P. Deussen, E.-R. Olderog, W.P. de Roever (editors):
Concurrent Systems: Semantics, Specification, and Synthesis, Dagstuhl-Seminar-Report; 9, 11.-15.3.1991 (9111)

K. Apt, K. Indermark, M. Rodriguez-Artalejo (editors):
Integration of Functional and Logic Programming, Dagstuhl-Seminar-Report; 10, 18.-22.3.1991 (9112)

E. Novak, J. Traub, H. Wozniakowski (editors):
Algorithms and Complexity for Continuous Problems, Dagstuhl-Seminar-Report; 11, 15-19.4.1991 (9116)

B. Nebel, C. Peltason, K. v. Luck (editors):
Terminological Logics, Dagstuhl-Seminar-Report; 12, 6.5.-18.5.1991 (9119)

R. Giegerich, S. Graham (editors):
Code Generation - Concepts, Tools, Techniques, Dagstuhl-Seminar-Report; 13, 20.-24.5.1991 (9121)

M. Karpinski, M. Luby, U. Vazirani (editors):
Randomized Algorithms, Dagstuhl-Seminar-Report; 14, 10.-14.6.1991 (9124)

J. Ch. Freytag, D. Maier, G. Vossen (editors):
Query Processing in Object-Oriented, Complex-Object and Nested Relation Databases, Dagstuhl-Seminar-Report; 15, 17.-21.6.1991 (9125)

M. Droste, Y. Gurevich (editors):
Semantics of Programming Languages and Model Theory, Dagstuhl-Seminar-Report; 16, 24.-28.6.1991 (9126)

G. Farin, H. Hagen, H. Noltemeier (editors):
Geometric Modelling, Dagstuhl-Seminar-Report; 17, 1.-5.7.1991 (9127)

A. Karshmer, J. Nehmer (editors):
Operating Systems of the 90s and Beyond, Dagstuhl-Seminar-Report; 18, 8.-12.7.1991 (9128)

H. Hagen, H. Müller, G.M. Nielson (editors):
Scientific Visualization, Dagstuhl-Seminar-Report; 19, 26.8.-30.8.91 (9135)

T. Lengauer, R. Möhring, B. Preas (editors):
Theory and Practice of Physical Design of VLSI Systems, Dagstuhl-Seminar-Report; 20, 2.9.-6.9.91 (9136)

F. Bancilhon, P. Lockemann, D. Tsichritzis (editors):
Directions of Future Database Research, Dagstuhl-Seminar-Report; 21, 9.9.-12.9.91 (9137)

H. Alt , B. Chazelle, E. Welzl (editors):
Computational Geometry, Dagstuhl-Seminar-Report; 22, 07.10.-11.10.91 (9141)

F.J. Brandenburg , J. Berstel, D. Wotschke (editors):
Trends and Applications in Formal Language Theory, Dagstuhl-Seminar-Report; 23, 14.10.-18.10.91 (9142)

H. Comon , H. Ganzinger, C. Kirchner, H. Kirchner, J.-L. Lassez , G. Smolka (editors):
Theorem Proving and Logic Programming with Constraints, Dagstuhl-Seminar-Report; 24, 21.10.-25.10.91 (9143)

H. Noltemeier, T. Ottmann, D. Wood (editors):
Data Structures, Dagstuhl-Seminar-Report; 25, 4.11.-8.11.91 (9145)

A. Dress, M. Karpinski, M. Singer(editors):
Efficient Interpolation Algorithms, Dagstuhl-Seminar-Report; 26, 2.-6.12.91 (9149)

B. Buchberger, J. Davenport, F. Schwarz (editors):
Algorithms of Computeralgebra, Dagstuhl-Seminar-Report; 27, 16.-20.12.91 (9151)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13..-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.02.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Modelling with Programmed Machines, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

K. Compton, J.E. Pin , W. Thomas (editors):
Automata Theory: Infinite Computations, Dagstuhl-Seminar-Report; 28, 6.-10.1.92 (9202)

H. Langmaack, E. Neuhold, M. Paul (editors):
Software Construction - Foundation and Application, Dagstuhl-Seminar-Report; 29, 13.-17.1.92 (9203)

K. Ambos-Spies, S. Homer, U. Schöning (editors):
Structure and Complexity Theory, Dagstuhl-Seminar-Report; 30, 3.-7.2.92 (9206)

B. Booß, W. Coy, J.-M. Pflüger (editors):
Limits of Information-technological Models, Dagstuhl-Seminar-Report; 31, 10.-14.2.92 (9207)

N. Habermann, W.F. Tichy (editors):
Future Directions in Software Engineering, Dagstuhl-Seminar-Report; 32; 17.2.-21.2.92 (9208)

R. Cole, E.W. Mayr, F. Meyer auf der Heide (editors):
Parallel and Distributed Algorithms; Dagstuhl-Seminar-Report; 33; 2.3.-6.3.92 (9210)

P. Klint, T. Reps, G. Snelting (editors):
Programming Environments; Dagstuhl-Seminar-Report; 34; 9.3.-13.3.92 (9211)

H.-D. Ehrich, J.A. Goguen, A. Sernadas (editors):
Foundations of Information Systems Specification and Design; Dagstuhl-Seminar-Report; 35; 16.3.-19.3.9 (9212)

W. Damm, Ch. Hankin, J. Hughes (editors):
Functional Languages:
Compiler Technology and Parallelism; Dagstuhl-Seminar-Report; 36; 23.3.-27.3.92 (9213)

Th. Beth, W. Diffie, G.J. Simmons (editors):
System Security; Dagstuhl-Seminar-Report; 37; 30.3.-3.4.92 (9214)

C.A. Ellis, M. Jarke (editors):
Distributed Cooperation in Integrated Information Systems; Dagstuhl-Seminar-Report; 38; 5.4.-9.4.92 (9215)