SAS-seminar Partial Evaluation
Anders Haraldsson
2012-03-02

A number of tracks:

My own research in partial evaluation, REDFUN
Research Datalogilaboratoriet -> MAI/IDA -> Pelab
The development of a research area.
Our influence of that area

The techniques of Partial Evaluation
Interesting and fascinating results
Interpretation – compilation
Self application

Anders Haraldsson

- 1966 Start studying Computer Science, Uppsala
- 1968 Studied last year for Erik Sandewall (Lisp, AI, logik)
- 1969 Bachelor exam in Uppsala
- 1969 Started to teach at Umeå, only one semester
- 1969 We started Datalogilaboratoriet
- 1970 IBM scholarship
- 1970 Research PCDB, Partial evaluation
- 1975 Erik Sandewall appointed professor in CS
- Most of Datalogilaboratoriet moved to Linköping
- 1977 Ph D (number 14 LiTH, today 1400)
- To 1982 Research group with partial evaluation
- Studierektor, head of department, head of undergraduate

Datalogilaboratoriet, Uppsala Univ. 1969-1975

Artificial intelligence (AI), Knowledge representation. Natural language. Programming languages. Data bases.

Implementing Lisp systems (IBM/360, Siemens, Fortran)

11 PhD thesis's in Computer Science at MAI/IDA.

My own work 1970-

PCDB Predicate Calculus Data Base. Organization of storing facts. Translating axioms to Lisp-code. “Compiling logic programming (PROLOG) to Lisp” . Continuations.

REDFUN. REDuce FUNarg. Partial evaluation.

Partial evaluation

Simple idea. Kleenes S-m-n theorem 1952

Given f(x, y) = x*y+x
Make a specialized version g giving that x=2
g(y) = f(2, y) = 2y+2

A partial evaluator generates that specialized function.
PE(f, “x=2) = g

More interesting:

- pow(x,y) = if y=0 then 1 else x*pow(x,y-1)
- PE(pow,"y=3") = pow-to-3(x) = x*x*x

A Partial Evaluator

A Partial evaluator evaluates what can be done and leaves the rest of the code. Some call this for mixed computation (Ershov).

- could handle if a variable had a known value or not
- constant propagation – calculate constant expressions
- unfolding (beta-expansion, inline) Replace a call with the body, after substitution of formal parameters against actual ones.
- pruning conditionals
- dead code elimination
- syntax transformations and simplifications
- ……

Partial evaluation REDFUN

We did an online Partial Evaluator.

Written in Lisp and could partial evaluate Lisp programs.

Advantage: No parsing. Lisp code represented as lists, no special AST (Abstract Syntax Tree). Code could be constructed and evaluator with the eval-function.

In principle followed the same strategy as an interpreter.

Se following outline as a separate slide.

Partial evaluation REDFUN

The first application.

Used it for PCDB.

Generated specialized functions for manipulation of facts.

Idea: Write few general functions, with a lot of parameters.

For every predicate a number of this parameters can be given (arity, one-many-ness, key). Partial evaluate these general functions to a specialized one for each predicate.

Axioms were translated to non-deterministic code and there was different skeleton depending on search strategy (backward, forward, deep first breadth first) and then specialized.

Partial evaluation REDFUN-2

My thesis work

Implement a next generation of REDFUN.

Handle a larger language

- side effects
- assigments
- goto-programs

Derive more information from the program

- a variable can have one of a set of values or is of a data type
- a variable can not have a set values
- handling of side effects
- ….

….

Partial evaluation REDFUN-2

The environment consisted of value-descriptors

((x . nobin) (y . (:values . (1 2 3))) (z . (:novalues . (4 5 6)))
(z . (:datatype . list-notnil)))

if x=2 then …. *x has the value 2* …… else …. *x can't be 2*

if member(x, (1 3 5)) then … *x can be 1, 3 or 5* else …. *not 1, 3, 5*

if number?(x) then …….. else

We have ((y . (:values . (1 2 3))) (x . nobin))

if x=y and member(x, (2 3 4))
then …*((x . (:values . (2 3)) (y . (:values . (2 3)))) …*
else … *((x . nobin) (y . (:values . (1 2 3)))*

Partial evaluation REDFUN-2

One observation: A lot of examples seem obscure. No one will write such code. Such code may appear when transforming and generating code.

And there is of course a lot of information which can not be derived, relations between variables …
For that we may need a theorem prover.

```
if x=y then …..
if prime?(x) then if x=7 then …..
```

Partial evaluation REDFUN-2

En online Partial Evaluator traverse the code as an interpreter and keeps information about variables in the environment.

In REDFUN-2:

Every expression was partial evaluated in a context, in order to know what kind of information is interested to extract.

(if expr …. …)	boolean context
(and expr …. …. ….)	boolean + “true” context
(f expr …. ….)	value context
expr	novalue context (as a statement)

Partial evaluation REDFUN-2

For every expression, which should remain in the code, was annotated with information.

This information was given by semantic functions associated with a function and construction.

expression

- its value (a value-descriptor)
- if it may perform or depend of a side effect or no
- variables which may get new values from assignments
- extracted information if expressions evaluates to true or false

Partial evaluation REDFUN-2

Example in Scheme (an obscure one)

(if (= x (+ (print 5) (set! y 10))) then … else

translates to

(if (= x (progn (print 5) (set! y 10) 15)) then … else …

print and set! return values. The value of the + expression evaluates. Side effects must remain.

The red is evaluated in an Boolean context:

it performs or depends on side effects

y will be set to 10

if it evaluates to true x will have the value 15

if it evaluates to false x will not have the value 15

Partial evaluation Futamuras first projection

Relation interpreter – compiler
Compiling embedded languages

We have an interpreter for a language R. It is written in language L.

Interpreter-for-R-written-in-L (program-in-R, environment)
Given a fixed program and an PE-for-L

PE-for-L (Interpreter-for-R-written in-L, program-in-R) = ?

A version of the interpreter specialized for the program =
compiled code for program-in-R to L

Partial evaluation Futamuras first projection

We used this for a number of embedded languages in the INTERLISP system.

- Pattern matching (regular expressions)
- Iterative expressions

We follow an example for a small pattern match language.

Int_{Lisp}^{PM} ($program_{PM}$, environment)

PE_{Lisp}^{Lisp} (Int_{Lisp}^{PM} , $program_{PM}$)= $program_{Lisp}$

A compilation of a program in pattern-match language to Lisp.

We follow an example.

Partial evaluation Futamuras second projection

But wait a minute. We had:

Int_{Lisp}^{PM} (program$_{PM}$, environment)

$\mathrm{PE}_{Lisp}^{Lisp}$ (Int_{Lisp}^{PM} , program$_{PM)}$ = program$_{Lisp}$

If ithe first argument (the interpreter) is known, we can partial evaluate the partial evaluator itself?

$\mathrm{PE}_{Lisp}^{Lisp}$ ($\mathrm{PE}_{Lisp}^{Lisp}$, Int_{Lisp}^{PM}) = ?

What is the specialized partial evaluator ? A compiler.

Partial evaluation Futamuras second projection

We got a compiler to compile programs in the pattern match language to Lisp!

PE_{Lisp}^{Lisp} (PE_{Lisp}^{Lisp} , Int_{Lisp}^{PM}) = $Comp_{Lisp}^{PM\text{->}Lisp}$

$Comp_{Lisp}^{PM>Lisp}$ ($program_{PM}$) = $program_{Lisp}$

We call such partial evaluator self-applicable.

Partial evaluation Futamuras third projection

But wait a minute. We had:

PE_{Lisp}^{Lisp} (PE_{Lisp}^{Lisp} , Int_{Lisp}^{PM}) = $Comp_{Lisp}^{PM>Lisp}$

If the first arguememt to PE_{Lisp}^{Lisp} is known it must be possible to partial evaluate PE_{Lisp}^{Lisp}

PE_{Lisp}^{Lisp} (PE_{Lisp}^{Lisp} , PE_{Lisp}^{Lisp}) = ?

Is it possible, and what is it ?

? is called cogen (compiler generator).
It is an interpreter to compiler generator!

Partial evaluation Futamuras third projection

We wrote such program for hand, called Redcompile.

A bachelor thesis by Lennart Beckman in Uppsala. Used for the PCDB application.

Instead of

PE(general-PCDB-functions, known-parametervalues) = specialized-PCDB-function

we did

PCOMP(A-general-PCDB-function, “parameters with known parameters”) = A-special-function-for-one-PCDB-function

A-special-function-for-one-PCDB-function(known-parametervalues)
=specialized-PCDB-function

Back to partial evaluation history

Futamura described the two first projections in a paper 1970

The third projection was first (in parallel with Turchin) mentioned by us in a paper Beckman & al, A partial evaluator, and its use as a programming tool, in the Artificial Intelligence journal 1976.

REDFUN was the first real partial evaluator for a quit large language.

The end of 70’s the most prominent researcher in computer science in Soviet Union, Andrei Ershov, worked on mixed-computation and met Sandewall at a conference 1977 in Soviet, where I came in contact with him.

I think I made Futamura well-known by sending his paper to Ershov 1978.

Back to partial evaluation history - IDA

At IDA we went on with research and Pär Emanuelson wrote a thesis on using my REDFUN-2 to general pattern matching. He tested our ideas with a much richer pattern match language with conditionals (compare today regular expressions)

Jan Komorowski applied the technique to logic programming = partial deduction, and is regarded the pioneer, the first application of partial evaluation to logic programming.

Ulf Nilsson had one part of his thesis 1992, where he with partial evaluation mapped Prolog-programs to WAM (Warrens Abstract Machine).

Back to partial evaluation history

In about 1982 we invited the Danish professor Neil Jones, having a seminar of compiler generation from denotational semantics. We pointed out the other direction with partial evaluation and self-application.

He took the ideas and after about 2 years he made Futamuras third projection. System called MIX/Similix. Start again from the beginning implementing very simple languages (pure functional Scheme).

Then applied to more rich languages (imperative …) and developing more theory.

Back to partial evaluation history

They introduced a two step PE – offline partial evaluator and introduced BTA – Binding Time Analysis.
`

Two steps:

First an annotation step, analyzing parts if it is static (possible to evaluate) or dynamic. Introduced two version of operations.

```
(define (f (x static) (y dynamic))
(if-s (= x 1) (if-d (= y 2) … …) ….)
```

Next step the optimization step.

Back to partial evaluation history

In Sweden:

Logic programming:

Uppsala (Ken Kahn – Pär Emanuelson opponent)
SICS (Dan Sahlin)

Functional programming, type inference:

Chalmers (John HugesJ

Back to partial evaluation history

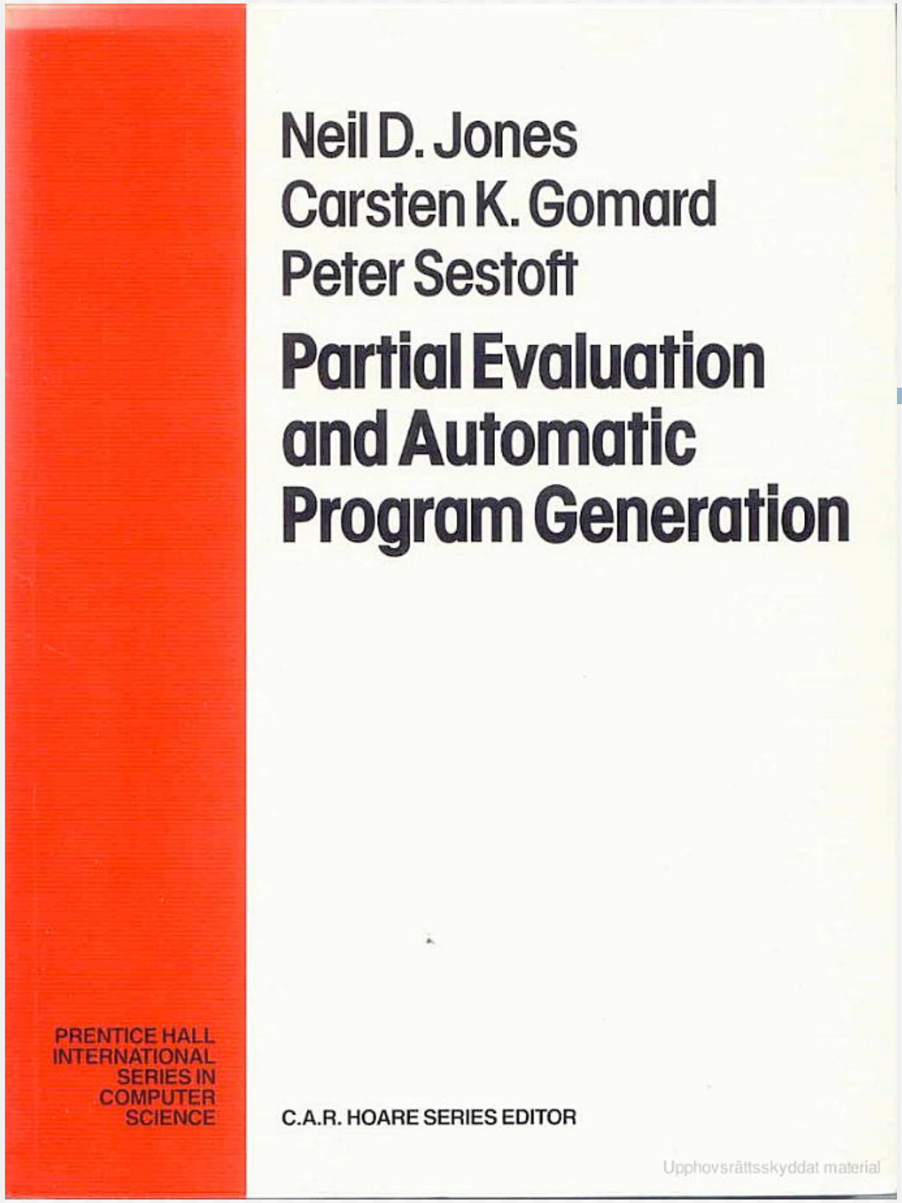
Now spread out in the world. Main research at DIKU Copenhagen of Neil Jones and his colleagues. They wrote a book, the PE-bible.

I was involved in organizing the first and second Partial evaluation conference and and one editor to a special edition …..

1987 in Gamle Avernes
1992 in New Haven.

The conference today is called PEPM (Partial Evaluation and (Semantics-based) Program Manipulation)

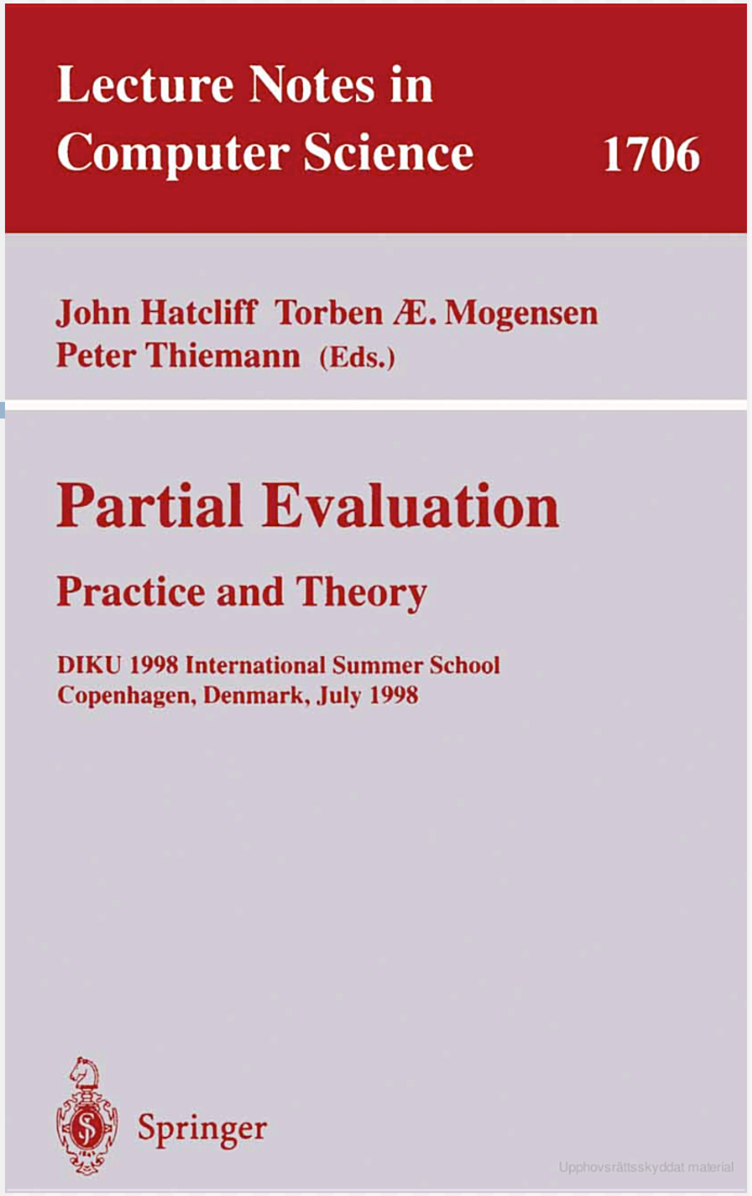
Back to partial evaluation history


Neil D. Jones
Carsten K. Gomard
Peter Sestoft

Partial Evaluation and Automatic Program Generation

PRENTICE HALL INTERNATIONAL SERIES IN COMPUTER SCIENCE

C.A.R. HOARE SERIES EDITOR


Lecture Notes in Computer Science 1706

John Hatcliff Torben Æ. Mogensen
Peter Thiemann (Eds.)

Partial Evaluation

Practice and Theory

DIKU 1998 International Summer School
Copenhagen, Denmark, July 1998

Springer

Today

PEPM 2012 Philadelphia

The PEPM Symposium/Workshop series aims to bring together researchers and practitioners working in the broad area of program transformation, which spans from refactoring, partial evaluation, supercompilation, fusion and other metaprogramming to model-driven development, program analyses including termination, inductive programming, program generation and applications of machine learning and probabilistic search. PEPM focuses on techniques, supporting theory, tools, and applications of the analysis and manipulation of programs.

Today

POPL 2012 Philadelphia

The annual Symposium on Principles of Programming Languages is a forum for the discussion of all aspects of programming languages and systems, with emphasis on how principles underpin practice. Both theoretical and experimental papers are welcome, on topics ranging from formal frameworks to experience reports. Papers discussing new ideas and areas are most welcome, as are high-quality expositions or elucidations of existing concepts that are likely to yield new insights ("pearls").

Looking back on the research

More “proof of concept” than theory.

Building systems and used it in applications.

Very important with our 1976 paper in the AI journal. The main reference paper in the area, during a long time.

I (also Komorowski) got a paper at the prestige conference 1978 POPL (Principal Of Programming Languages) and at the first Lisp conference 1980 at Stanford.

We could have written more after that. More how to organize a partial evaluator. Methods and algorithms. Why not more journal papers?

Compare today with other research areas

Can an optimizing compiler do PE today, i.e. the pattern match interpreter. You define that some functions (patmatch and segmatch) should be compiled inline? Is it smart to do all optimization and simplifications as the PE?

My REDFUN-2 is mode of less the same as attribute grammars. Inherited and derived attributes. What information can I get from an attribute grammar?

What are Fritzson and Broman doing together Magnusson and Herdin in Lund with metacompilation?

Mariam Kamkar made a PhD Thesis on slicing. A formal paper 2006 descries the relationships between PE and slicing. Our work was referenced!

Today

Questions:

- Is partial evaluation used today? Program transformation on the source level?

- Is it today an interesting optimization, computers are so fast, why bother of this kind of optimizations?, use general functions.

- Is it interesting to automate the process to generate compilers? Are there other techniques to use instead?

- Is it still a hard problem to make really good partial evaluation? Have we obtained optimal code?