Conference Publications

A Practical Guide for the Practicing PhD Student

Kevin Hammond
University of St Andrews, Scotland
Target Audience

New PhD Students
- How to get your work published
- How does the system work

More Experienced PhD Students
- What is a good conference paper
- Some Tips and Tricks
My Background

Around 80-90 refereed papers published so far
  – About 10-12 last year (6 so far this year)
  – Mainly in programming language research
  – POPL, PLDI, ICFP, TOPLAS, …
  – My $h$-index is around 24

Lots of programme committees
  – 6-7 so far this year

I do have a PhD
  – but I had to “learn the hard way”
  – first conference presentation was at a major conference
    » giving the invited talk, on behalf of my PhD supervisor

I enjoy writing papers
Please ask me Questions!

Lets me know that you’re awake!

- keeps me awake

Tells me what’s important!

- the only stupid question is the one you’re afraid to ask

If you ask too many questions, I can always answer them later!

If you don’t ask me any questions, I will ask you questions!
Audience Expertise

Publications
- Never
- Workshops Only
- 1-2 Conference Papers
- > 2 Conference Papers

Programme Committees
- Never
- Workshops Only
- 1-2 Conferences
- > 2 Conferences
Overview

• **Session 1: 10:30-11:30**
  – structure of a paper
  – writing tips
  – reviews and programme committees
  – a worked example

• **Coffee/Tea Break/Poster Session**
  (i.e. Networking Opportunity)

• **Session 2: 12:00-13:00**
  – structure of a paper
  – writing tips
  – reviews and programme committees
  – a worked example
Audience Question: Why do I want to write a Paper?

Please add answers directly to

http://sicsa-phd.wikispaces.com

or hand notes to one of the helpers
What is the Point of a Conference?

- Share ideas and results
- Learn from others
  - and what they are working on
- *Networking*
- Promotion/advertising
- Discussion
- (go to talks)
- (present a paper)
A Verified Staged Interpreter is a Verified Compiler

Multi-stage Programming with Dependent Types

Edwin Brady, Kevin Hammond
School of Computer Science, University of St Andrews, St Andrews, Scotland.
Email: eb, kh@dcs.st-and.ac.uk

Abstract

Dependent types and multi-stage programming have both been used, separately, in programming language design and implementation. Each technique has its own advantages — with dependent types, we can verify aspects of interpreters and compilers such as type safety and stack invariants. Multi-stage programming, on the other hand, can give the implementor access to underlying compiler technology, a staged interpreter is a translator. In this paper, we investigate the combination of these techniques. We implement an interpreter for a simply typed lambda calculus, using dependent types to guarantee correctness properties by construction. We give explicit proofs of these correctness properties, then add staging annotations to generate a translator from the interpreter. In this way, we have constructed a verified compiler from a verified staged interpreter. We illustrate the application of the technique by considering a simple staged interpreter that provides guarantees for some simple resource bound properties, as might be found in a domain specific language for real-time embedded systems.

Categories and Subject Descriptors
D.3.4 [Programming Languages]: Processors — Interpreters, Compilers, Translator writing systems and compiler generators, D.2.4 [Software Engineering]: Software/Program Verification — Correctness proofs, Formal methods.

General Terms
Languages, Theory, Verification.

Keywords
Dependent types, Multi-stage programming, Partial evaluation, DSL Implementation, Resource aware programming.

1. Introduction

Multi-stage programming supports separation of concerns in compiler writing, by allowing automatic program generation to proceed in a series of stages. Each stage captures some new aspect of the problem space that is then reflected in subsequent stages through the program that is generated. A primary advantage of the approach is that it supports the construction of domain specific notations in a nested fashion. Here, each stage allows the encapsulation of domain knowledge in a precise way, and programmers may work at different levels (corresponding to stages) according to their degree of specialisation. For example, in the domain of real-time embedded systems which we are investigating, the first stage might be a restricted notation that guaranteed bounded time and space usage, and be used by the applications programmer; the subsequent stage might be used to define these restricted notations in terms of the underlying meta-programming system, and be used by the domain expert, and the final stage would correspond to the generation of executable code, and be used by the compiler writer.

A major problem with this approach arises in ensuring that generated programs conform to the properties required by the interpreter/programmer. This problem has been explored in outline by Tahay, Sheard and Patillic amongst others [27, 31, 29], who have produced systems that use capable of correctly preserving type information across stages. While this is a valuable contribution in reducing runtime type errors for generated programs, these approaches restrict the expressivity of their type systems. This approach is valuable in allowing the automatic verification of types, but verification of more complex properties will generally require more complex proof structures than can be supported by such frameworks.

For example, the calculation of bounds on the resources used by a generated program may be essential in a real-time embedded systems setting. Previous work e.g. [17, 27] uses multi-stage programming to generate resource correct programs, but is limited to specific resource correctness properties. We are thus motivated to consider how arbitrary proofs may be embedded within multi-stage programs in a homogenous framework, in order to allow automatic verification of required program properties for domain specific languages implemented using a multi-stage approach.

1.1 Overview of our Approach

Types give a program meaning; dependent type systems, in which types may be predicated on values, allows us to give a more precise type to a program and therefore to be more confident that it has the intended meaning. In this paper, we consider how the separate techniques of multi-stage programming and dependently typed programming can be combined in order to implement an efficient and correct implementation of a functional programming language.

We use dependent types to implement a well-typed interpreter, following and extending the ideas of Aspinwood and Curien [4]. We are able to show by construction that the interpreter returns a value of the correct type and correctly evaluates well-typed terms — we take types as the prior notion representing a specification, and use the typechecker to guarantee that our program respects this specification. Dependent types enable, by static checking, that the interpreter cannot be executed on badly formed or ill-typed code.

Thus dependent types provide static guarantees of certain desired correctness properties.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists requires prior specific permission and/or a fee.

Copyright © 2006 ACM 1-59593-775-2/06/0010...

Kevin Hammond, University of St Andrews

SICSA PhD Conference 2011
General Structure

1. Abstract
2. Introduction
3. (Related Work/Context/Background)
4. New Content
5. Evaluation
6. Related Work
7. Conclusions and Further Work
8. Acknowledgements
9. References
Abstract

What is the topic of the paper?

What are the key results?

Why should I spend time reading this paper?

1-2 paragraphs
Introduction

Motivates the research
why is this research interesting

Introduces general ideas
key background

States the novelty of the research

Gives an overview of the paper

Try to fit the introduction onto the first page if possible

Typically 1-2 pages
Background/Context

- Describe the key ideas in the area
  - give a taxonomy if appropriate

- Define any important terms that you will use

- Provide all the necessary background to understand the new content

- Typically 0-4 pages
Content

• The important stuff!

• Explain what you have done

• Don’t give a history, give a coherent picture!

• Give enough detail so the reader knows what you’ve done
  – but don’t drown them in detail: use structure

• Typically 4-5 pages
Evaluation

• **Could be**
  – experimental results
  – examples
  – proofs

• **Explain why your approach works**
  – highlight any important findings
  – show any limitations

• *Typically 1-3 pages*
Related Work

• Discuss the *most relevant* pieces of research

• Explain why work that sounds similar is different

• Explain why other work is limited and doesn’t solve the problem you’re addressing

• Discuss different approaches to the problem
  – if not already done, outline the field

• *Typically about 0.5-1 page*
Conclusions

• What are the key issues that have been described in the paper

• What is the significance of what has been done
  – What is the contribution to knowledge

• What are the limitations on what has been done
  – How well does the paper address the problem

• How could the work be improved (Further Work)

• Typically 0.5 pages
Acknowledgements

• Acknowledge your sponsors

• Acknowledge people who’ve helped you
  – those who gave you ideas
  – work that you’ve used

• Acknowledge the referees (in the final version)

• **CUT this section in the draft if you’re short of space!!**
References

- Cite the most relevant research papers
- Cite the seminal research papers
- Cite a few of your own papers
- Cite the papers that sound like they’re related
- (Cite papers written by the programme committee members)

- Typically about 20-30 references for a conference paper
  - some short papers may have only 5-10!
Writing Tips
How to write a Paper

1. Write the abstract
2. Write the introduction
3. Do the work
4. Write the content
5. Write the conclusion
6. Write the background
7. Write the related work
8. *Rewrite* the introduction, conclusion and finally abstract
9. Get feedback from colleagues, friends, …
10. *Rewrite as necessary*
11. Submit it!
Key Points that need to be Addressed

1. (Self-contained)
   - I should not need to read another paper to understand this one

2. Novel
   - The paper contains new results/new information or describes a new approach

3. Coherent
   - The paper makes sense
   - It has a logical flow of argument
   - Don’t hold results back
   - But don’t stuff too many ideas into one paper

4. Well-written
   - Good, grammatical English
   - Well-structured – terms described before use
   - Well-illustrated – diagrams, examples
Reinventing the Wheel

• Don’t be afraid to do your own thing!
  – Maybe your idea is better!

• Don’t read so much literature that you don’t know where to start
  – Do the work first!
6 Commandments for Bad Writing

1. Thou shalt not define thy terms
2. Neither shalt thou explain anything
3. Thou shalt publish before implementing
4. Thou shalt replace “will do” with “have done”
5. Thou shalt not mention any drawbacks
6. Thou shalt not reference any papers

Prof. David Patterson, UCB
Reviewing and Programme Committees
How Programme Committees Work

- PC members bid for papers to review
- The PC chair allocates papers
- PC members review their papers
  - or allocate sub-reviewers
- Reviewers submit their reviews
- (authors respond to reviews)
- Papers are *ranked*, and categorised
  - accept, accept?, discuss, reject?, reject
- PC members may suggest changes to the categories
Double-blind Submissions

• Advantage
  – nobody knows who you are
  – no bias (e.g. against newbies, institutions, people)

• Disadvantage
  – nobody knows who you are
  – no credit if you get something wrong
  – hard to really be anonymous
  – difficult to reference your own work properly
  – hard to evaluate scientific quality = more random!
  – malicious reviewers can break anonymity anyway
  – difficult to give extra material
How Programme Committees Work (2)

- Clear accepts/rejects are moved to one side
- Unclear papers are discussed
  - in person
  - electronically (e.g. email or easychair)
- Additional reviews may be sought (e.g. from an expert)
- Reviews may be updated
- More discussion
- Final decisions are taken
- Accept/reject notifications are sent
- Programme chair designs conference program

*often restricted by talk slots*
Review Forms

• **Key elements**
  – overall rating
  – reviewer confidence
  – novelty
  – significance
  – accuracy
  – relevance to the conference
  – writing quality

• **Plus**
  – freely structured comments for the author
  – private comments for the programme committee
Overall Rating

- Usually
  -3 strong reject (including out of scope)
  -2 reject
  -1 weak reject
  0 borderline
  1 weak accept
  2 accept
  3 strong accept

- undecided...
- rare!
Confidence

- **Usually**
  - 0: none
  - 1: low
  - 2: medium
  - 3: high
  - 4: expert

- I have read papers in the area
- I have done research in the area
- I am the world authority
Ranking

• Often overall rating combined with confidence
  – more confident reviews get more weight

• Numerical value used to give initial ranking
  – then similar papers discussed to get strong order
  – often only for borderline papers

• Accepted papers almost always need some clear accepts
The Champion System

- Rather than ranking papers, give a qualitative rating
  - A: I will champion the paper
  - B: OK, but I will not champion it
  - C: Weak, but I will not fight it
  - D: I will actively fight to exclude the paper

- At the Programme Committee meeting, champions speak for their papers
  - unchampioned papers are rejected

- Advantage: someone has to *like* the paper
- Disadvantage: strong personalities get their way!
Exercise: Programme Committee

- Divide into groups of three
  - each group will have the same 3 papers to referee

- Spend 15 minutes reading 2 papers each
  - each paper should have 2 referees

- Fill in the referee form for each paper individually

- Spend 5 minutes ranking the papers
  - Best, second best, worst

- Decide which paper(s), if any to accept
End of Part 1
Programme Committee Results

![Graphs showing the results of the programme committee evaluations for three papers.](image-url)
Everyone gets some Rejections

- Even Turing Award winners
- The question is how to respond
How to Respond to a Bad Review

• Aim to be a good academic

• Throw a massive tantrum

• Sulk for a week

• Blacklist the conference and never submit there again

• Tell everyone how badly you’ve been treated
  – and how stupid the referees are
Seriously...

- *Wait a day*

- *Read the reviews* *carefully*
  
  - pick out any points to learn from
  - think about *why* the reviewer made the comment
    
    » didn’t understand the paper
    » novelty wasn’t made obvious
    » missed some points that you thought you’d made
    » there is some missing related work
    » presentation could be better
  
  - think about how the paper could be *improved*

*I also do this for positive reviews*
Then...

- Think about what to do with the paper
  - revise it and send it to another conference
  - do some more work, then rewrite it
  - extend it and send it to a journal
  - use it as an invited talk/book chapter
  - turn it into a technical/online report
  - make it into a poster
Example Reports

• The paper appears to be written in a hurry
  – there are a few grammatical/presentation errors

• Get a native speaker to check the paper
  – there are a LOT of grammatical/English errors

• The paper didn’t mention X
  – X is a pet theory of the reviewer

• (null review)
  – the reviewer made comments to the PC chair
Some Howlers

• “This paper does not say its ideas are original so I will assume they are not”

• “The authors clearly don’t understand X”

• “In 1996 we submitted a paper to the top AI conference that year. We had spent ages and ages making the slightly complicated idea easy to understand. One of the reviews (which was on paper, believe it or not, and I don't have a copy) said words to the effect of "This idea is so simple that I will assume it is not new." Without - of course - giving any indication of a previous source.

• The next year we submitted it to the top AI conference and made sure to obfuscate it a bit this time. And it got accepted.”
Author Responses

• Be polite
  – always thank the referees for their comments
• Highlight the positives
• Point out any contradictions between the referees
• Defend any negatives/point out any mistakes
• Respond to any questions
• Always quote the text you’re responding to
  – much easier for the programme committee

A good response can make all the difference
If your paper is accepted

• Follow the referee instructions
  – as best you can
  – if sensible
  – *in case of doubt, the PC overrides the referees!*
• Comply with the page limits
  – you may be able to ask for more space
• Reformat the paper if necessary
• Meet the deadlines
  – you may be able to ask for more time
• Communicate with the PC chair
  – revisions, problems, questions, …

adding better results is rarely a problem!
More on Authorship
Who Should be an Author

- Everyone who has contributed written text
- Everyone who has committed major ideas
- (Your supervisor/boss)
- (Those who have provided major help)
Author Order

- Often determined by the research group/area
  - alphabetical
  - most involved first
  - (most senior first)

- First author is often assumed to be more prestigious

- Single author papers often assumed to be better
  - but not practicable in applied areas
Who Should be Acknowledged

• Those who helped
  – discuss the paper
  – run experiments
  – by supporting your research

• Everyone who has committed some ideas that you have used

• Sponsors, institutions you visited while writing the paper, institutions that loaned facilities, …
Choosing a Title

- Descriptive
- Witty
- Informative
- Use a Colon/subtitle to make it clear!

A Verified Staged Interpreter is a Verified Compiler
Multi-stage Programming with Dependent Types

Edwin Brady Kevin Hammond
School of Computer Science, University of St Andrews, St Andrews, Scotland.
Email: eb,kh@dcs.st-and.ac.uk
Some Good Titles in my Area

- Report on the Programming Language Fortran
- The next 700 Programming Languages
- Linear Types can Change the World!
- Why Functional Programming Matters
Selecting the Conference

• How well does your paper fit the call for papers
  – make sure your paper is in scope and firmly on topic

• Be realistic but ambitious
  – is the paper good enough?
  – but higher impact conferences are often better on the CV

• Remember good conferences are made by good papers
  – a bad paper is not improved by being in a good conference
  – even the best conferences publish some dud papers
Selecting the Conference (2)

• **Think about the audience**
  – focused events are often more effective than big, diffuse conferences
  – you want to target key researchers
  – networking is important too: discuss ideas
  – are you targeting a new audience?
  – who is on the programme committee?

• **Conference v. Symposium**
  – probably doesn’t matter!
  – POPL is a *symposium*
  – *workshops* are usually less selective though (not always!)
Selecting the Conference (3)

• Is the page limit right
  – needs to be long enough for the material

• Are the deadlines realistic
  – if you don’t get any sleep, the paper won’t be good!

• Don’t choose a conference just because of its location
  – a nice location is a bonus for a good conference
  – but some conferences count negatively on your CV!
Good (and bad) Papers
Audience Input

• Why do you want to write a paper?
Some Reasons for Writing a Paper

• To tell people about a new result
• To develop a new research idea
• To explain something better
• To introduce a new audience to the work
• To demonstrate collaboration
• To record your plans
• To survey an area
• To state your position on an important issue
• To increase your publication count
How many Papers should I publish?

Only those that are necessary!

“Every time you have a good idea”

*Active* computer scientists typically publish 3-4 per year

A *really good* PhD will yield about 3-5 papers

It usually takes about 2-3 months to write a good paper if you write more than this, either the quality is low you’re very experienced you’re working with a large group of co-authors
Bad Career Move #7: Writing Tactics for a Bad Career

- Papers: It’s Quantity, not Quality
  - Personal Success = Length of Publication List
  - “The LPU (Least Publishable Unit) is Good for You”

```
1 idea
4 journal papers
16 extended abstracts
64 technical reports
```

“Publication pyramid of success”

(c) David Patterson
The Debt to Science

Every time you submit a paper, you owe the world 3-4 reviews...
I have reviewed about 50-60 papers this year...
plus about 8 PhD theses...
plus about 10 grant proposals...

Have a good reason for wanting to publish
Paper Quality/Length

• Don’t hold back on material

• Do make sure you have a single coherent subject

• Use all the space you have wisely
  — include diagrams, think about what can be added

• A conference paper usually needs more detail than a workshop paper
How to write a Good 12-page Paper

• First write the 16-20 page paper you would like to write
• Trim any less important/superfluous material
  — e.g. full proofs, full experimental results, digressions, …
  — could be an appendix to the submission
  — could refer to a website with extra material
  — the full version could be a technical report/online draft

• Put your system online so the referees can try it out!
  — and include a URL in the paper
  — this can save putting a detailed explanation in the paper
How to write a Good 12-page Paper (2)

- Squeeze the paper into the available space
  - shorten paragraphs by rewriting them
  - eliminate any “widows” and bad page breaks
  - remove any redundancy
  - move the diagrams around (don’t put figures on a separate page)
  - shorten the references
  - combine related paragraphs
  - squeeze vertical whitespace
  - combine multiple figures into one
  - use a double-column format (diagrams/code in one column)
  - omit the acknowledgements in the draft
  - use numbered references (BUT ORDER them alphabetically!)
How to write a Good 12-page Paper (3)

• If all else fails…
  – reduce the related work
  – cut out a few references
  – shorten the conclusions and abstract
  – omit less important results
  – consider writing two papers! Or a journal paper!

• But never…
  – make the paper hard to read
  – fail to contextualise the work properly
  – fail to state the novelty
  – trim important results or fail to explain them
Using Papers to help Write a Thesis

+ Sorts out ideas

+ Forces you to read the literature, work out what’s important and develop your own taxonomy

+ Forces you to run all the experiments/finish the proof...

+ Gives you feedback
  + from the reviews
  +++ more importantly, from talking to people at the event

+ Gives writing practice
Using Papers to help Write a Thesis

- You still need to “join the dots”
- You need to restructure
- You need to do a deeper literature survey
To Conclude
Communication with the PC Chair

• Many conferences extend deadlines
  – if you need more time ask the PC chair
  – don’t send an incomplete/bad paper

• Ask if you’re not sure whether your paper is in scope

• When revising a paper for publication
  – if you need more space, ask
  – if you don’t understand a comment, ask
  – if you need a little more time, ask
  – give a commented list of changes, if need be
    » and state what you haven’t changed, and why!
How to Find out More

• **Volunteer to help**
  – refereeing
  – organisation

• **Online resources**
  – Simon Peyton Jones “How to write a research paper”
  – David Patterson “How to have a bad career”
Question and Answer Session
My Home Page

Kevin Hammond

Professor in Computer Science

I'm a (full) Professor in Computer Science, in the School of Computer Science, at the University of St. Andrews, St. Andrews, Scotland, where I lead the Functional Programming research group. My main interests are in cost modelling, parallelism and real-time and embedded systems. Please email me if you are interested in undertaking a PhD in my area.

www.cs.st-andrews.ac.uk/~kh