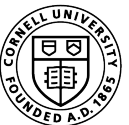
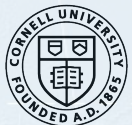


Finding a research topic or being found by a research topic?

Alexandra Silva



My research journey



Cornell Bowers C/IS
Computer Science

My research journey

Functional
Programming

Strong types for relational databases



Authors: [Alexandra Silva](#), [Joost Visser](#) [Authors Info & Claims](#)

Haskell '06: Proceedings of the 2006 ACM SIGPLAN workshop on Haskell • September 2006 • Pages 25–36 • <https://doi.org/10.1145/1159842.1159846>



Cornell Bowers C/IS
Computer Science

My research journey

Functional
Programming

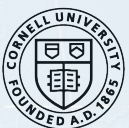
Coalgebra

An Algebra for Kripke Polynomial Coalgebras



Authors:  [Marcello Bonsangue](#),  [Jan Rutten](#),  [Alexandra Silva](#) [Authors Info & Claims](#)

LICS '09: Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science • August 2009 • Pages
49–58 • <https://doi.org/10.1109/LICS.2009.18>



Cornell Bowers C/IS
Computer Science

My research journey

Functional
Programming

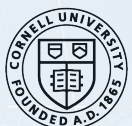
Coalgebra

An Algebra for Kripke Polynomial Coalgebras



Authors: [Marcello Bonsangue](#), [Jan Rutten](#), [Alexandra Silva](#) [Authors Info & Claims](#)

LICS '09: Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science • August 2009 • Pages 49–58 • <https://doi.org/10.1109/LICS.2009.18>



Cornell Bowers C/IS
Computer Science

My research journey

Functional
Programming

Coalgebra

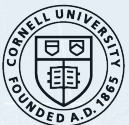
Functional
Programming

Language constructs for non-well-founded computation



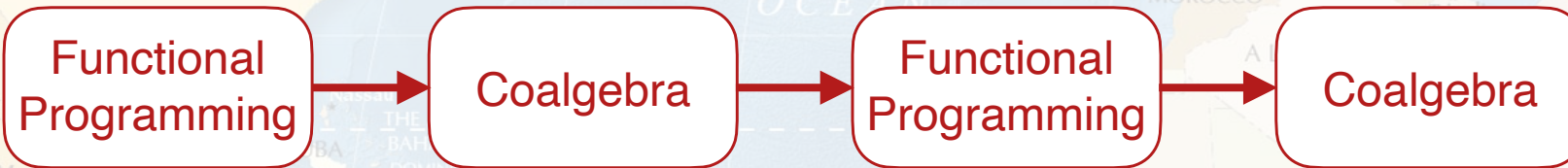
Authors: [Jean-Baptiste Jeannin](#), [Dexter Kozen](#), [Alexandra Silva](#) [Authors Info & Claims](#)

ESOP'13: Proceedings of the 22nd European conference on Programming Languages and Systems • March 2013 • Pages 61–80 • https://doi.org/10.1007/978-3-642-37036-6_4



Cornell Bowers C/IS
Computer Science

My research journey

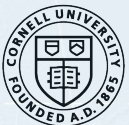


Algebra-coalgebra duality in brzozowski's minimization algorithm



Authors: [Filippo Bonchi](#), [Marcello M. Bonsangue](#), [Helle H. Hansen](#), [Prakash Panangaden](#),
 [Jan J. M. M. Rutten](#), [Alexandra Silva](#) [Authors Info & Claims](#)

ACM Transactions on Computational Logic, Volume 15, Issue 1 • February 2014 • Article No.: 3, pp 1–



Cornell Bowers C/IS
Computer Science

My research journey



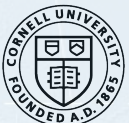
Cantor meets Scott: semantic foundations for probabilistic networks



Authors: [Steffen Smolka](#), [Praveen Kumar](#), [Nate Foster](#), [Dexter Kozen](#), [Alexandra Silva](#) [Authors Info & Claims](#)

POPL 2017: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages • January 2017

• Pages 557–571 • <https://doi.org/10.1145/3009837.3009843>



Cornell Bowers CIS
Computer Science

My research journey



RESEARCH-ARTICLE •   

Prognosis: closed-box analysis of network protocol implementations



Authors:  [Tiago Ferreira](#),  [Harrison Brewton](#),  [Loris D'Antoni](#),  [Alexandra Silva](#) [Authors Info & Claims](#)

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference • August 2021 • Pages 762–774 • <https://doi.org/10.1145/3452296.3472938>



Cornell University
Bowers CIS
Computer Science

My research journey



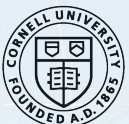
RESEARCH-ARTICLE •   

Prognosis: closed-box analysis of network protocol implementations



Authors:  [Tiago Ferreira](#),  [Harrison Brewton](#),  [Loris D'Antoni](#),  [Alexandra Silva](#) [Authors Info & Claims](#)

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference • August 2021 • Pages 762–774 • <https://doi.org/10.1145/3452296.3472938>



Cornell University
Bowers CIS
Computer Science

My research journey



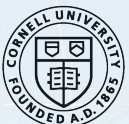
RESEARCH-ARTICLE •   

Prognosis: closed-box analysis of network protocol implementations



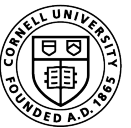
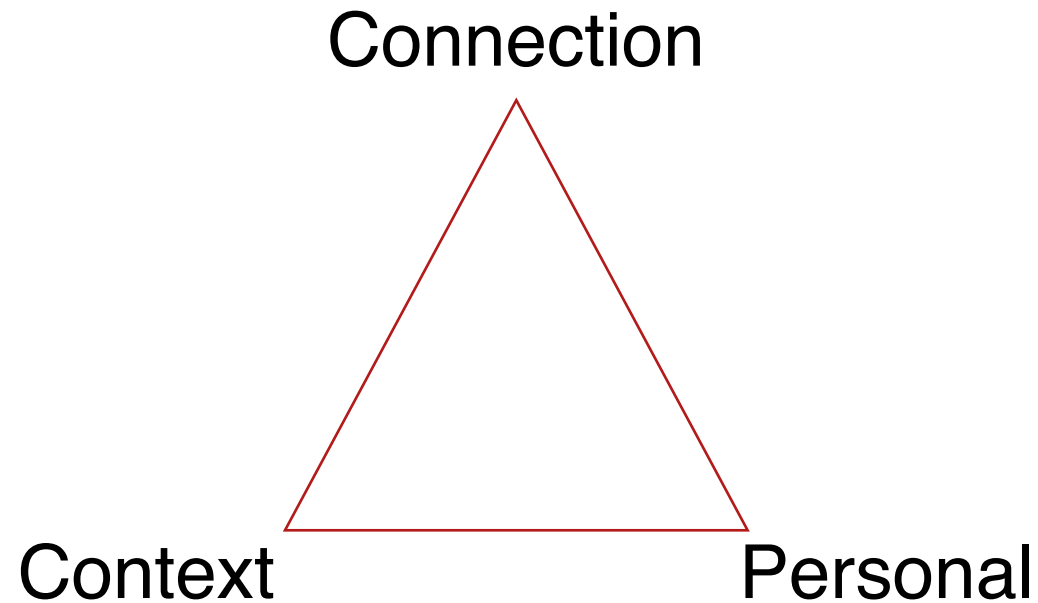
Authors:  [Tiago Ferreira](#),  [Harrison Brewton](#),  [Loris D'Antoni](#),  [Alexandra Silva](#) [Authors Info & Claims](#)

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference • August 2021 • Pages 762–774 • <https://doi.org/10.1145/3452296.3472938>



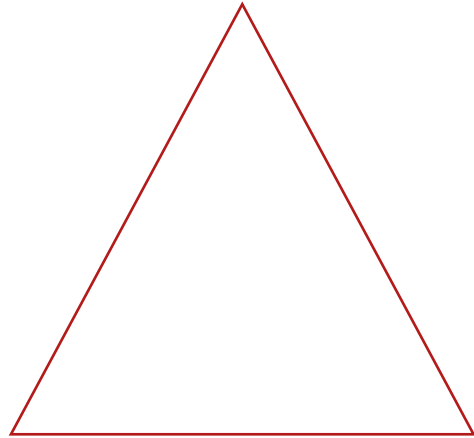
Cornell University
Bowers CIS
Computer Science

How did I choose a research topic?



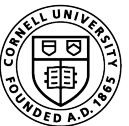
How did I choose a research topic?

Connection



Context

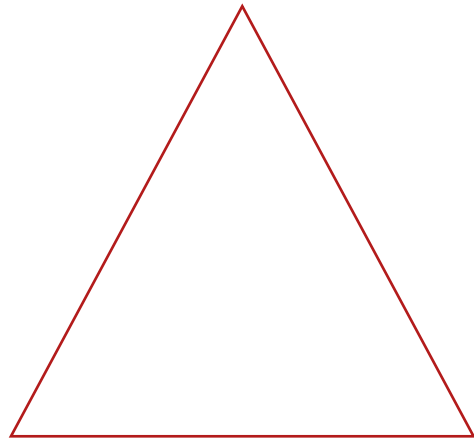
Personal



How did I choose a research topic?

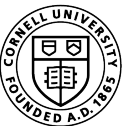
Some lessons I learned along the way:

Connection



Context

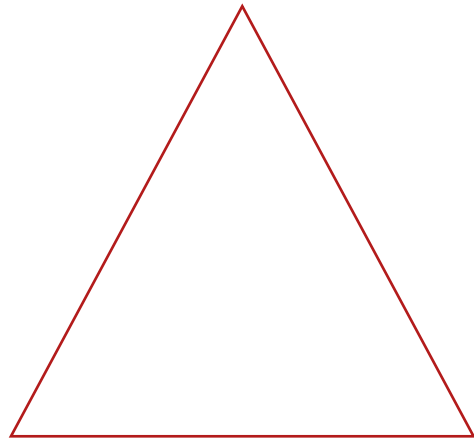
Personal



How did I choose a research topic?

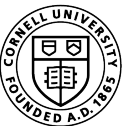
Some lessons I learned along the way:

Connection



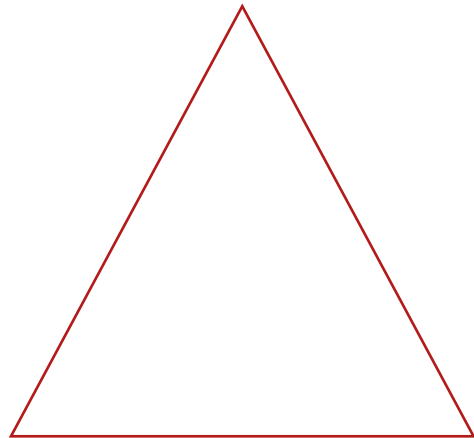
Context

Personal



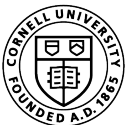
How did I choose a research topic?

Connection



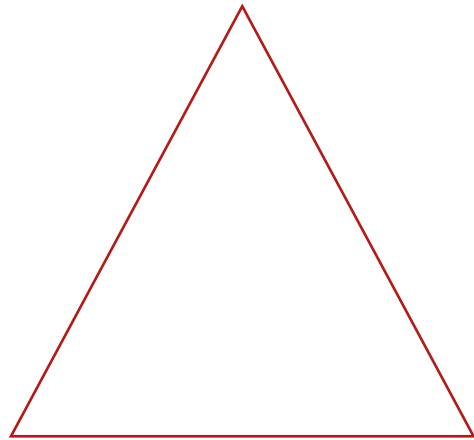
Some lessons I learned along the way:

- The topic is only one component



How did I choose a research topic?

Connection

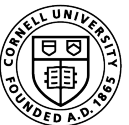


Context

Personal

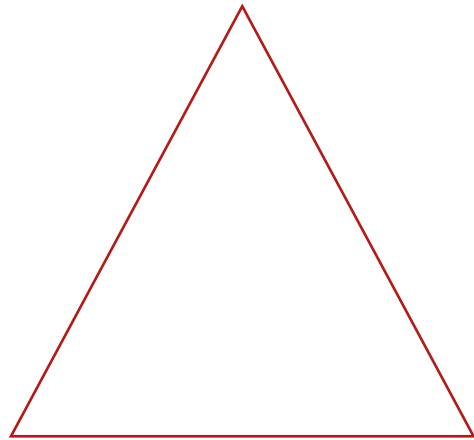
Some lessons I learned along the way:

- The topic is only one component
- Working with other people implies compromise



How did I choose a research topic?

Connection

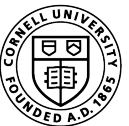


Context

Personal

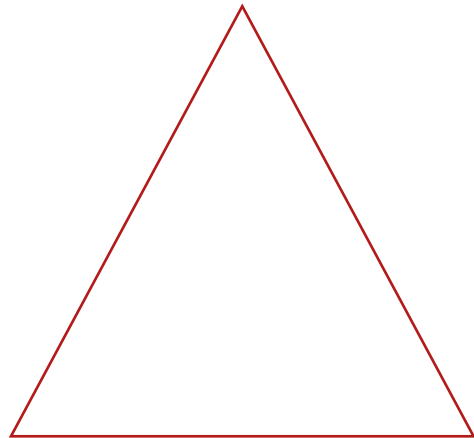
Some lessons I learned along the way:

- The topic is only one component
- Working with other people implies compromise
- There are things I will not work on



How did I choose a research topic?

Connection

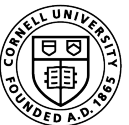


Context

Personal

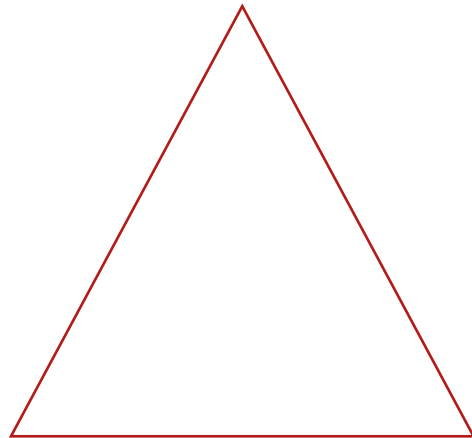
Some lessons I learned along the way:

- The topic is only one component
- Working with other people implies compromise
- There are things I will not work on
- There are people I will not work with



How did I choose a research topic?

Connection

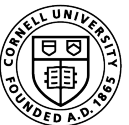


Context

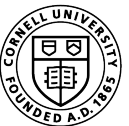
Personal

Some lessons I learned along the way:

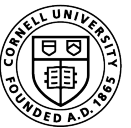
- The topic is only one component
- Working with other people implies compromise
- There are things I will not work on
- There are people I will not work with
- What you do not want is as important as what you want



Supervisors (-or- the early days)



Supervisors (-or- the early days)

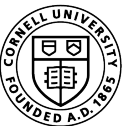


Supervisors (-or- the early days)



Which of these is most important in a supervisor:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

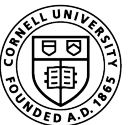
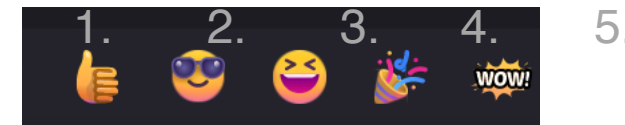


Supervisors (-or- the early days)



Which of these is most important in a supervisor:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

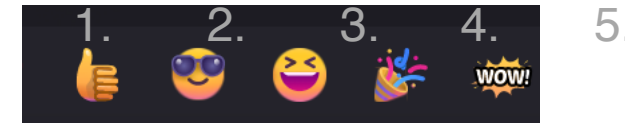


Supervisors (-or- the early days)

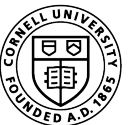


Which of these is most important in a supervisor:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun



Others?

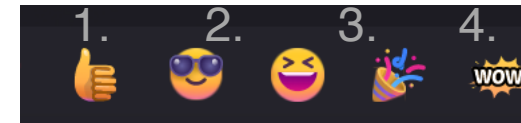


Supervisors (-or- the early days)

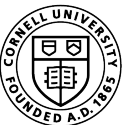


Which of these is most important in a supervisor:

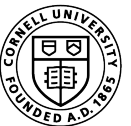
1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun



Others?



Collaborators

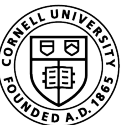


Collaborators



Which of these is most important in a collaborator?

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

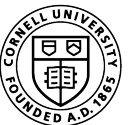
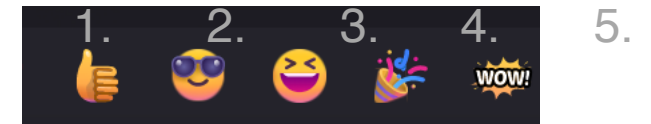


Collaborators



Which of these is most important in a collaborator?

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun



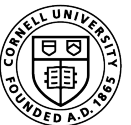
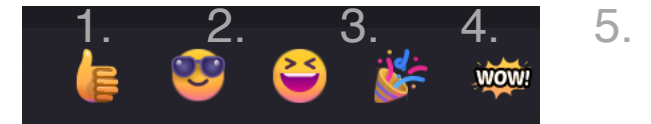
Collaborators



Which of these is most important in a collaborator?

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

Others?



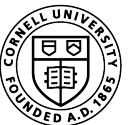
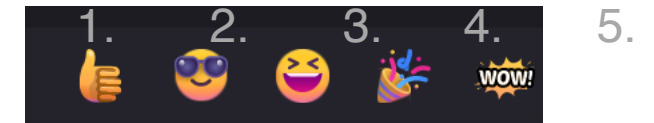
Collaborators



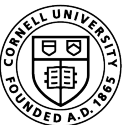
Which of these is most important in a collaborator?

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

Others?



Students

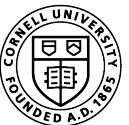


Students



Which of these is most important in a student:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

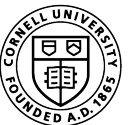
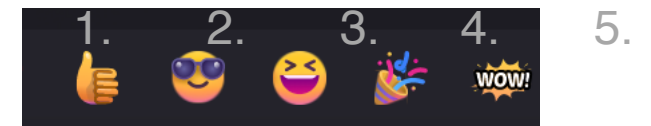


Students



Which of these is most important in a student:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun



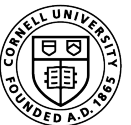
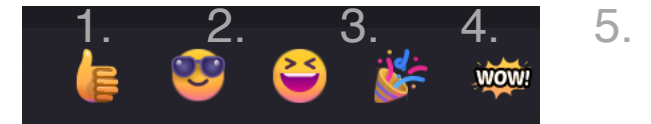
Students



Which of these is most important in a student:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

Others?



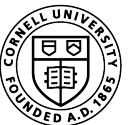
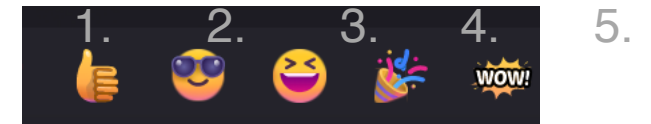
Students



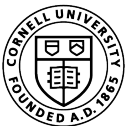
Which of these is most important in a student:

1. Experience
2. Knowledge (breadth, depth)
3. Network
4. Empathy
5. Fun

Others?



Community

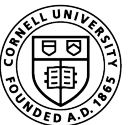
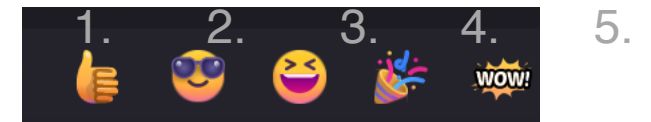


Community



Which of these is most important in a research community:

1. Hot topics to work on
2. Mentoring events
3. Collaborative opportunities
4. Good teaching materials
5. Conferences/Networking events



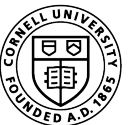
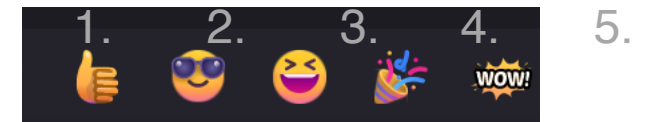
Community



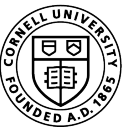
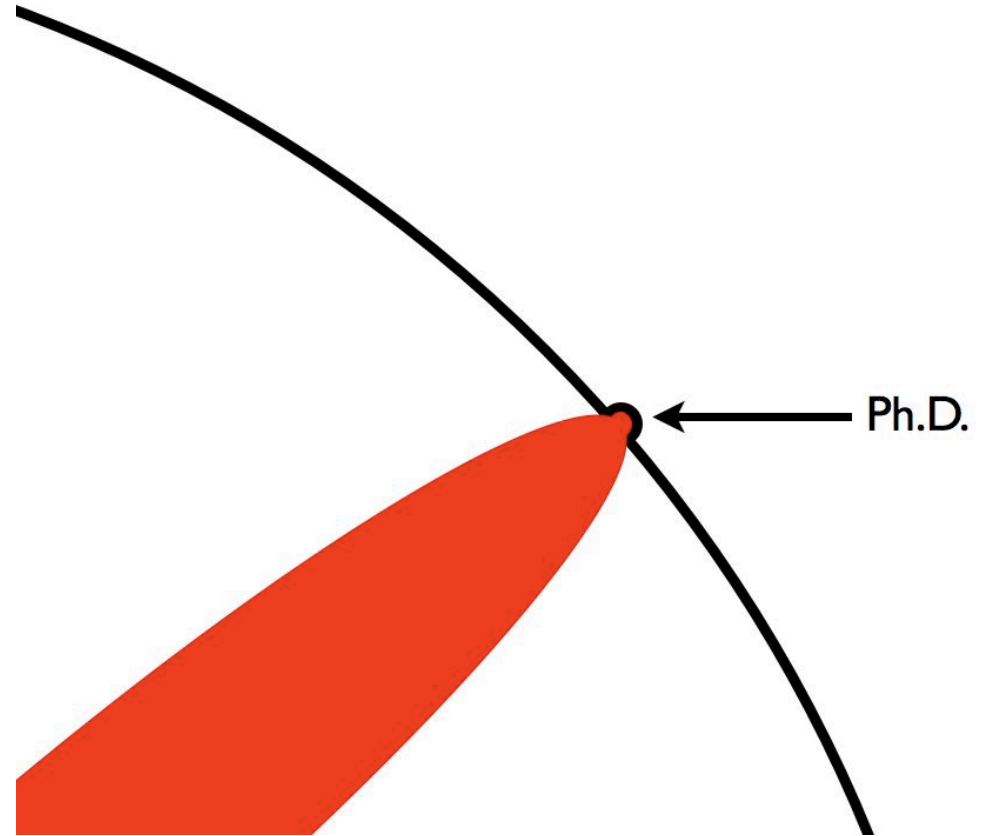
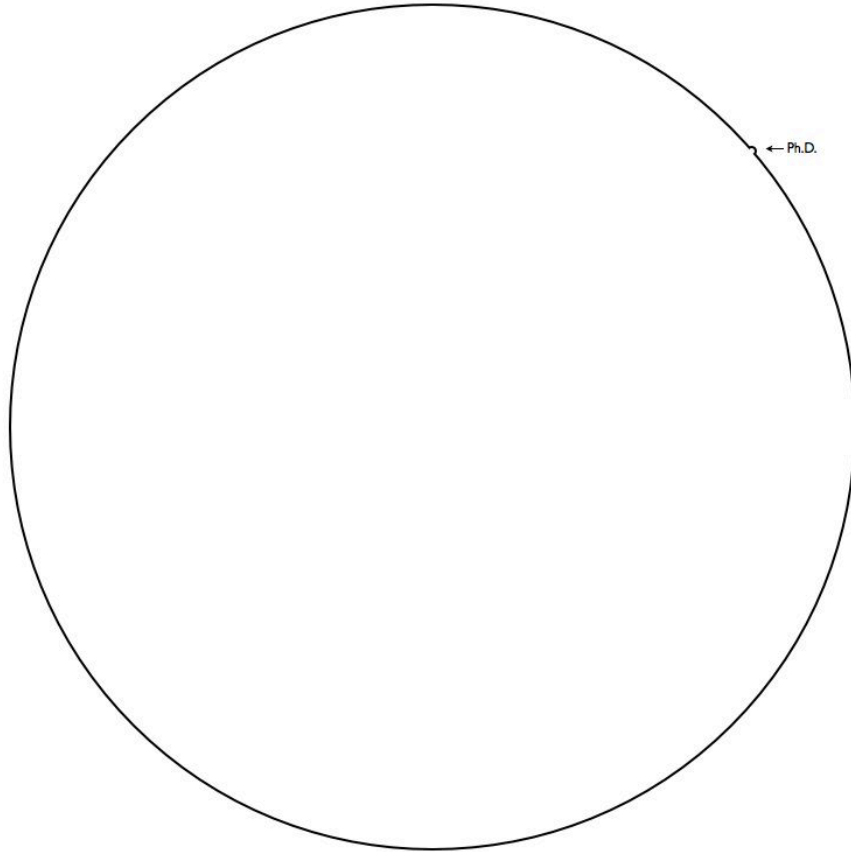
Which of these is most important in a research community:

1. Hot topics to work on
2. Mentoring events
3. Collaborative opportunities
4. Good teaching materials
5. Conferences/Networking events

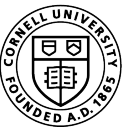
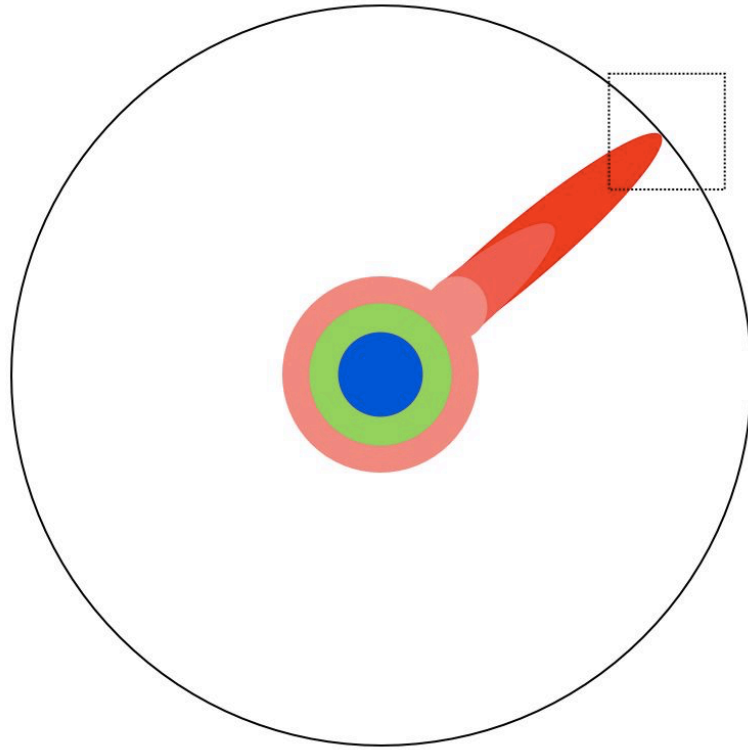
Others?



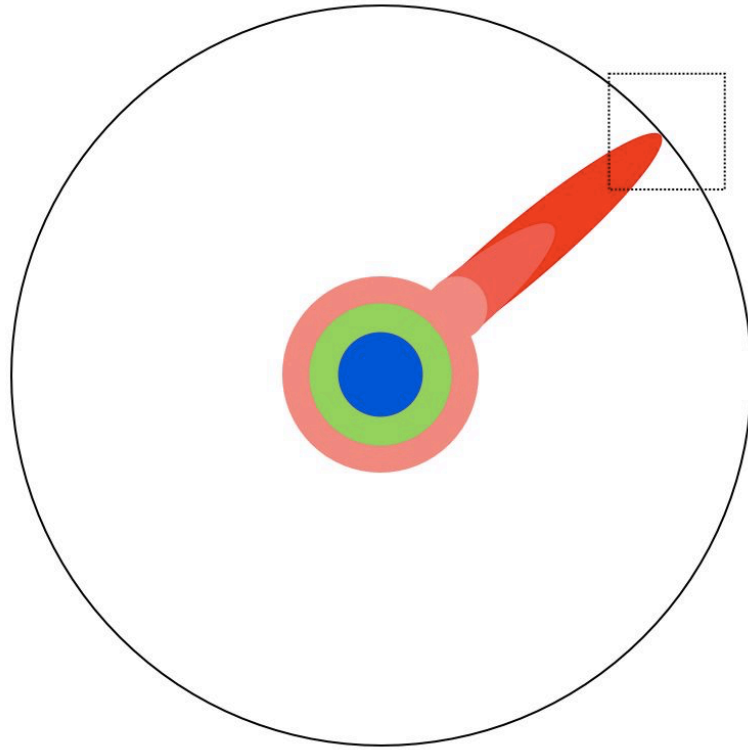
Enthusiasm matters...



... but ...



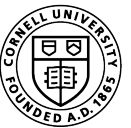
... but ...



Keep moving, even if slowly!



Find your style



Find your style



✨ Jean Yang ✨
@jeanqasaur

...

Taylor Swift as important papers in programming languages, a thread.

"An Axiomatic Basis for Computer Programming,"
C.A.R. Hoare, 1969. Introduced Hoare Logic for
proving program properties.

cs.cmu.edu/~crary/819-f09...

Computer Programming

C. A. R. HOARE
The Queen's University of Belfast, Northern Ireland

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming, proofs of program, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORIES: A.0, A.21, A.32, S.20, S.21, S.23, S.24

1. Introduction

Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs. The exact choice of axioms will to some extent depend on the choice of programming language. For illustrative purposes, this paper is confined to a very simple language, which is effectively a subset of all current procedure-oriented languages.

2. Computer Arithmetic

The first requirement in valid reasoning about a program is to know the properties of the elementary operations which it invokes, for example, addition and multiplication of integers. Unfortunately, in several respects computer arithmetic is not the same as the arithmetic familiar to mathematicians, and it is necessary to exercise some care in selecting an appropriate set of axioms. For example, the axioms displayed in Table I are rather a small selection of axioms relevant to integers. From this incomplete set

$$\begin{aligned}x &= x + y \times 0 \\ y < r &\supset r + y \times q = (r - y) + y \times (1 + q)\end{aligned}$$

The proof of the second of these is:

$$\begin{aligned}\text{A5 } (r - y) + y \times (1 + q) \\ &= (r - y) + (y \times 1 + y \times q) \\ \text{A9 } &= (r - y) + (y + y \times q) \\ \text{A3 } &= ((r - y) + y) + y \times q \\ \text{A6 } &= r + y \times q \quad \text{provided } y < r\end{aligned}$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to nonnegative numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

(2) Firm boundary: the result of an overflowing operation is taken as the maximum value represented.

(3) Modulo arithmetic: the result of an overflowing operation is computed modulo the size of the set of integers represented.

These three techniques are illustrated in Table II by addition and multiplication tables for a trivially small model in which 0, 1, 2, and 3 are the only integers represented.

It is interesting to note that the different systems satisfying axioms A1 to A9 may be rigorously distinguished from each other by choosing a particular one of a set of mutually exclusive supplementary axioms. For example, infinite arithmetic satisfies the axiom:

$$\text{A10, } \neg \exists x \forall y \quad (y < x),$$

where all finite arithmetics satisfy:

$$\text{A10, } \forall x \quad (x < \text{max})$$

where "max" denotes the largest integer represented.

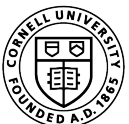
Similarly, the three treatments of overflow may be distinguished by a choice of one of the following axioms relating to the value of $\text{max} + 1$:

$$\text{A11, } \neg \exists x \quad (x = \text{max} + 1) \quad (\text{strict interpretation})$$

$$\text{A11, } \text{max} + 1 = \text{max} \quad (\text{firm boundary})$$

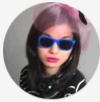
$$\text{A11, } \text{max} + 1 = 0 \quad (\text{modulo arithmetic})$$

Modern subsets of these axioms, and more, are given in Table I.



Cornell Bowers C'IS
Computer Science

Find your style

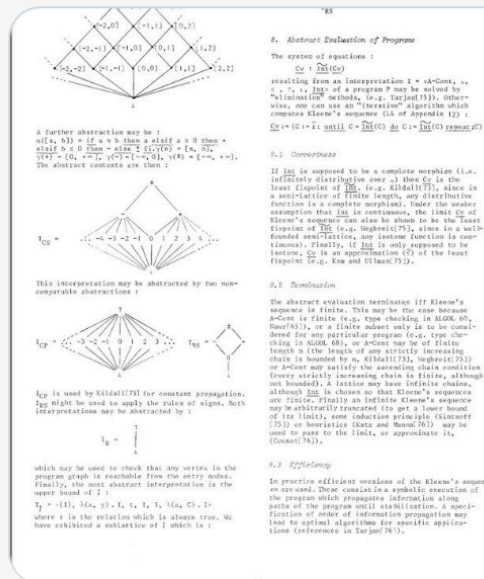


★ Jean Yang ★ @jeanqasaur · 11 Aug 2020

Replying to @jeanqasaur

"Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," Cousot & Cousot, 1977. Introduced abstract interpretation for statically analyzing program properties. Now used by Airbus.

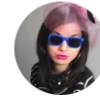
di.ens.fr/~cousot/COUSOT...



2

3

56



★ Jean Yang ★

@jeanqasaur

...

Taylor Swift as important papers in programming languages, a thread.

"An Axiomatic Basis for Computer Programming," C.A.R. Hoare, 1969. Introduced Hoare Logic for proving program properties.

cs.cmu.edu/~crary/819-f09...

Computer Programming

C. A. R. HOARE
The Queen's University of Belfast, Northern Ireland

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming, proofs of program, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORIES: A.0, A.21, A.32, S.20, S.21, S.23, S.24

1. Introduction

Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning. Deductive reasoning involves the application of valid rules of inference to sets of valid axioms. It is therefore desirable and interesting to elucidate the axioms and rules of inference which underlie our reasoning about computer programs. The exact choice of axioms will to some extent depend on the choice of programming language. For illustrative purposes, this paper is confined to a very simple language, which is effectively a subset of all current procedure-oriented languages.

2. Computer Arithmetic

The first requirement in valid reasoning about a program is to know the properties of the elementary operations which it involves, for example, addition and multiplication of integers. Unfortunately, in several respects computer arithmetic is not the same as the arithmetic familiar to mathematicians, and it is necessary to exercise some care in selecting an appropriate set of axioms. For example, the axioms displayed in Table I are rather a small selection of axioms relevant to integers. From this incomplete set

$$x = x + y \times 0$$

$$y < r \supset r + y \times q = (r - y) + y \times (1 + q)$$

The proof of the second of these is:

$$A5 \quad (r - y) + y \times (1 + q)$$

$$= (r - y) + (y \times 1 + y \times q)$$

$$A9 \quad = (r - y) + (y + y \times q)$$

$$A3 \quad = ((r - y) + y) + y \times q$$

$$A6 \quad = r + y \times q \quad \text{provided } y < r$$

The axioms A1 to A9 are, of course, true of the traditional infinite set of integers in mathematics. However, they are also true of the finite sets of "integers" which are manipulated by computers provided that they are confined to nonnegative numbers. Their truth is independent of the size of the set; furthermore, it is largely independent of the choice of technique applied in the event of "overflow"; for example:

(1) Strict interpretation: the result of an overflowing operation does not exist; when overflow occurs, the offending program never completes its operation. Note that in this case, the equalities of A1 to A9 are strict, in the sense that both sides exist or fail to exist together.

(2) Firm boundary: the result of an overflowing operation is taken as the maximum value represented.

(3) Modulo arithmetic: the result of an overflowing operation is computed modulo the size of the set of integers represented.

These three techniques are illustrated in Table II by addition and multiplication tables for a trivially small model in which 0, 1, 2, and 3 are the only integers represented.

It is interesting to note that the different systems satisfying axioms A1 to A9 may be rigorously distinguished from each other by choosing a particular one of a set of mutually exclusive supplementary axioms. For example, infinite arithmetic satisfies the axiom:

$$A10 \quad \neg \exists y \quad (y < \max)$$

where all finite arithmetics satisfy:

$$A10' \quad \forall x \quad (x < \max)$$

where "max" denotes the largest integer represented.

Similarly, the three treatments of overflow may be distinguished by a choice of one of the following axioms relating to the value of max + 1:

$$A11a \quad \neg \exists x \quad (x = \max + 1) \quad (\text{strict interpretation})$$

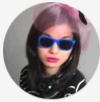
$$A11b \quad \max + 1 = \max \quad (\text{firm boundary})$$

$$A11c \quad \max + 1 = 0 \quad (\text{modulo arithmetic})$$

Modern authors use of these axioms is as follows:



Find your style

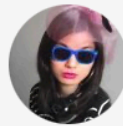
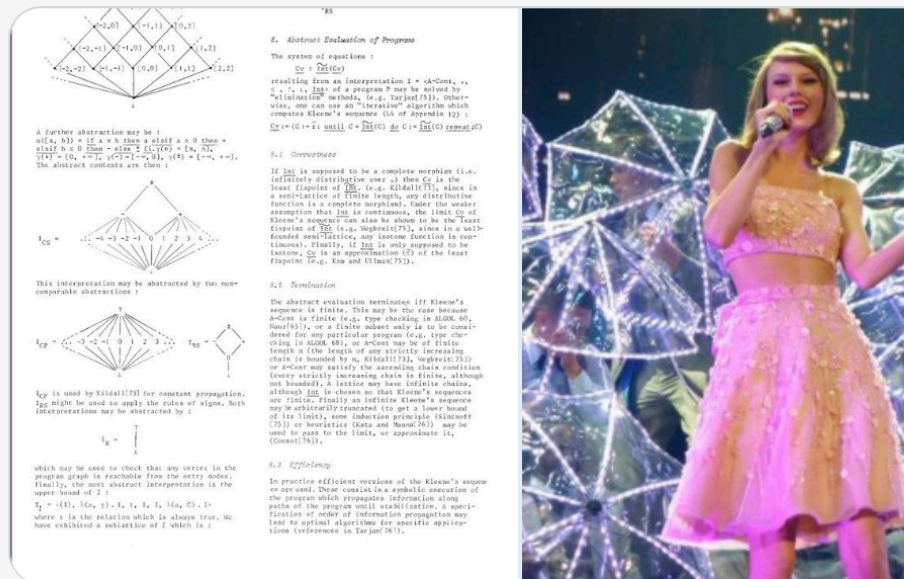


✨ Jean Yang ✨ @jeanqasaur · 11 Aug 2020

Replying to @jeanqasaur

"Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," Cousot & 1977. Introduced abstract interpretation for statically analyzing program properties. Now used by Airbus.

di.ens.fr/~cousot/COUSOT...



✨ Jean Yang ✨ @jeanqasaur · 11 Aug 2020

"From System F to Typed Assembly Language," @GMorrisett et al, 1998. Introduced a typed assembly language + type-preserving translation. Being able to prove properties of assembly through types is huge! POPL "Most Influential Paper" in 2008.

cs.cornell.edu/talc/papers/ta...

types	$\tau ::= a \mid \text{int} \mid \forall [x]. \tau \mid (\tau_1^1, \dots, \tau_n^m) \mid \text{box } \tau$
initialization flags	$\varphi ::= 0 \mid 1$
heap types	$\Psi ::= \{r_1, r_2, \dots, r_n, r_n\}$
register file types	$\Gamma ::= \{r_1, r_2, \dots, r_n, r_n\}$
type contexts	$\Delta ::= \emptyset$
registers	$r \in \{r_1, r_2, r_3, \dots\}$
word values	$w ::= d \mid i \mid \text{Tr} \mid \text{w}[r] \mid \text{pack } [r, w] \text{ as } v'$
small values	$v ::= r \mid w \mid \text{w}[r] \mid \text{pack } [r, w] \text{ as } v'$
heap values	$h ::= [r_1, \dots, r_n] \mid \text{code}[r] \mid S$
heaps	$H ::= \{r_1 \mapsto h_1, \dots, r_n \mapsto h_n\}$
register files	$R ::= \{r_1 \mapsto w_1, \dots, r_n \mapsto w_n\}$
instructions	$i ::= \text{add } r_1, r_2, v \mid \text{box } r, v \mid \text{ld } r_1, r_2 \mid \text{store } r_1[r], \text{sub } r_1, r_2, v \mid \text{unpack } [r, r_2], v$
instruction sequences	$S ::= \emptyset \mid S \mid \text{halt}[r]$
programs	$P ::= (H, R, S)$

Figure 6: Syntax of TAL

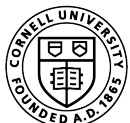
$\mathcal{R} \models$	$(H, R, S) \models P$ where
$\text{add } r_1, r_2, v, S$	$(H, R(r_1 \mapsto R(r_1) + R(r_2)), S')$
$\text{box } r, v, S$	(H, R, S')
$\text{ld } r_1, r_2, S$	when $R(r_1) = 0$
$\text{store } r_1[r], v, S$	when $R(r_1) = i$ and $i \neq 0$
$\text{unpack } [r, r_2], v, S$	when $R(r_1) = i$ and $R(r_2) = \text{code}[i]$
$\text{halt}[r], S$	when $R(r_1) = i$ and $R(r_2) = \text{code}[i]$
$\text{ld } r_1, r_2, S$	when $R(r_1) = i$ and $R(r_2) = \text{code}[i]$
$\text{store } r_1[r], v, S$	when $R(r_1) = i$ and $R(r_2) = \text{code}[i]$
$\text{unpack } [r, r_2], v, S$	when $R(r_1) = i$ and $R(r_2) = \text{code}[i]$

Figure 7: Operational Semantics of TAL



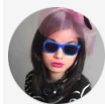
arithmetic is not the same as the arithmetic familiar to mathematicians, and it is necessary to exercise some care in selecting an appropriate set of axioms. For example, the axioms displayed in Table 1 are rather a small selection of axioms relevant to integers. From this incomplete set

- $\text{All } x. \neg \exists x. (x = \text{max} + 1)$ (strict interpretation)
- $\text{All } x. \text{max} + 1 = \text{max}$ (firm boundary)
- $\text{All } x. \text{max} + 1 = 0$ (modulo arithmetic)



Cornell Bowers C/IS
Computer Science

Find your style

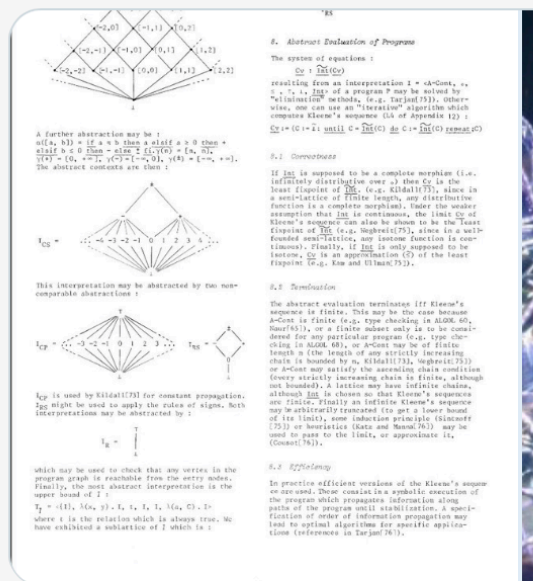


✨ Jean Yang ✨ @jeanqasaur · 11 Aug

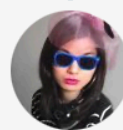
Replying to @jeanqasaur

"Abstract interpretation: a unified lattice programs by construction or approximation 1977. Introduced abstract interpretation properties. Now used by Airbus.

di.ens.fr/~cousot/COUSOT...



2 3



✨ Jean Yang ✨ @jeanqasaur · 11 Aug 2020

"From System F to Typed Assembly Language," @GMorrisett et al, 1998. Introduced a typed assembly language + type-preserving translation. Being able to prove properties of assembly through types is huge! POPL "Most Influential Paper" 2016.

✨ Jean Yang ✨ @jeanqasaur · 11 Aug 2020

"Formal Certification of a Compiler Back-end," Xavier Leroy (OCaml creator!), 2006. First paper to use interactive theorem-proving (with Coq) to program and formally verify a C compiler. Landmark paper in verification. POPL "Most Influential Paper" 2016.

xavierleroy.org/publi/compiler...

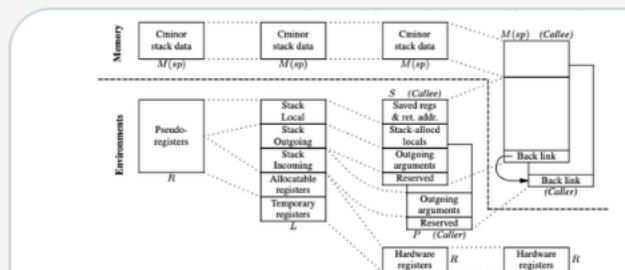


Figure 1. Overview of register allocation and introduction of activation records. For each intermediate language, the placement of function-local data is outlined, either in the memory-allocated activation record (top part) or in non memory-resident execution environments (bottom part).

```
getenv()
cond(cond, P, true)
return
```

```
Linear functions:
fn ::= fun() : sig
{ stack-allocated code }
```

The dynamic semantics of Linear, like those of RTL and LTL, is "mostly small-steps": each instruction is a transition in the semantics, but a call instruction transitions directly to the state at function return.

Mach The last intermediate language in our gentle descent towards PowerPC assembly is called Mach. It is a variant of Linear where the three infinite supplies of stack slots (local, incoming and outgoing) are mapped to actual memory locations in the stack frames of the caller (for local and outgoing slots) or the caller (for incoming slots).

```
Mach instructions:
setstack(r, τ, δ) register to stack move
getstack(r, δ, r) stack to register move
getparent(r, δ, r) caller's stack to register move
... as in Linear
```

In the three new move instructions, τ is the type of the data moved and δ is the word offset in the corresponding activation record. The semantics of call is also modified so that all hardware registers are global and shared between caller and callee: there is no automatic restoration of callee-save registers at function return; instead, the Mach code producer must produce appropriate `setstack` and `getstack` instructions to save and restore used callee-save registers at function prologues and epilogues.

The semantics for Mach is of the form $G, fn, sp \vdash \zeta, R, M \rightarrow \Gamma', R', M'$, where R is a mapping from hardware registers to values, and `setstack`, `getstack` and `getparent` are interpreted as

unconditional branch
conditional branch
function return

3.3 The target language: PowerPC macro-assembly
The target language for our compiler is abstract syntax for a subset of the PowerPC assembly language (90 instructions of the 2006-offered by the processor). The semantics is purely small-step and defines a transition over the state of registers and memory for every instruction. The registers modeled are: all general-purpose integer and float registers, the PC, LR and CTR special registers, and bits 0 to 3 of the condition register.

The semantics is (to the best of our knowledge) faithful to the actual behavior of PowerPC instructions, with one exception: the `fsadd` and `fsabs` instructions (combined multiply-add and multiply-sub over floats) are treated as producing the same results as a normal multiply followed by a normal addition or subtraction, ignoring the fact that `fsadd` and `fsabs` skip a rounding step on the result of the multiply. Depending on how the certification of the source program treats floating-point numbers (e.g. as IEEE floats or as intervals of real numbers), this infidelity can be semantically correct or not. It is however trivial to turn off the generation of `fsadd` and `fsabs` instructions in case exact results at the bit level are required.

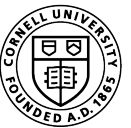
Our PowerPC assembly language features a handful of macro-instructions that expand to canned sequences of actual instructions during pretty-printing of the abstract syntax to concrete assembly syntax. These macro-instructions include allocation and deallocation of the stack frame (mapped to arithmetic on the stack pointer register), integer to float conversions (mapped to complicated bit-level manipulations of IEEE floats), and loading of a floating-point literal (mapped to a load from a memory-allocated constant). The reason for treating these operations as basic instructions is that it was deemed too difficult and not worthwhile to certify the correctness of the corresponding canned sequences. For instance, proving



Cornell Bowers C:IS
Computer Science

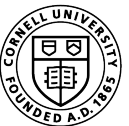


Find your style

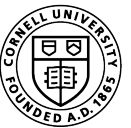


Find your style

Why is style important?

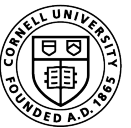


More practically...



More practically...

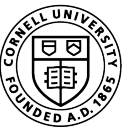
Read a lot (of introductions!)



More practically...

Read a lot (of introductions!)

YouTube

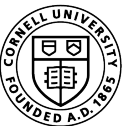


More practically...

Read a lot (of introductions!)

YouTube

Talk to people whose work you like



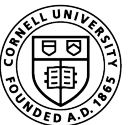
More practically...

Read a lot (of introductions!)

Talk to peers

YouTube

Talk to people whose work you like



More practically...

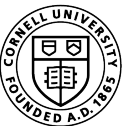
Read a lot (of introductions!)

Talk to peers

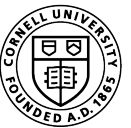
YouTube

Twitter

Talk to people whose work you like

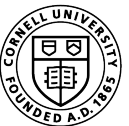


And when things go wrong?



And when things go wrong?

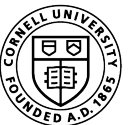
You do not like the topic



And when things go wrong?

You do not like the topic

Collaboration breakdown

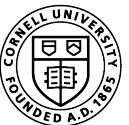


And when things go wrong?

You do not like the topic

Collaboration breakdown

Your work keeps getting rejected



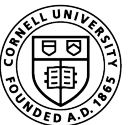
And when things go wrong?

You do not like the topic

Supervisor conflict

Collaboration breakdown

Your work keeps getting rejected



And when things go wrong?

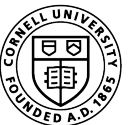
You do not like the topic

Supervisor conflict

Collaboration breakdown

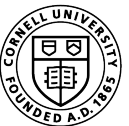
Mental health

Your work keeps getting rejected



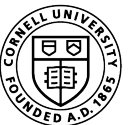
Some reflections...

Finding a matching topic is important —
for your daily happiness and for your
career



Some reflections...

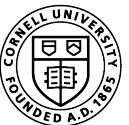
Finding a matching topic is important —
for your daily happiness and for your
career



Some reflections...

Finding a matching topic is important —
for your daily happiness and for your
career

Multidimensional decision
External Factors

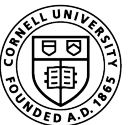


Some reflections...

Finding a matching topic is important —
for your daily happiness and for your
career

Multidimensional decision
External Factors

A topic is not forever



Some reflections...

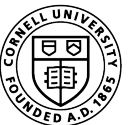
Finding a matching topic is important —
for your daily happiness and for your
career

Multidimensional decision
External Factors

A topic is not forever



Your research evolves with the years —
topic-wise and style-wise



Some reflections...

Finding a matching topic is important —
for your daily happiness and for your
career

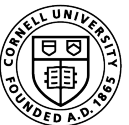


Multidimensional decision
External Factors

Your research evolves with the years —
topic-wise and style-wise

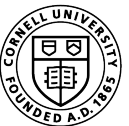
A topic is not forever

Sometimes the topic finds you



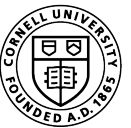
Research is a conversation

- *Stephanie Weirich, PLMW'22*



A photograph of the Cornell University Bowers Hall, a modern building with a glass and steel facade. The building is partially obscured by a large black rectangular overlay. Below the black overlay is a solid red horizontal bar. The foreground shows a series of concrete steps leading up to the building.

Questions?



Cornell Bowers C-IS
Computer Science