Wrap-up Lecture

CIS 5000: SOFTWARE FOUNDATIONS

Steve Zdancewic

Fall 2022

Announcements

- HW12: Sub.v
 - due: Monday, Dec. 12th at *midnight*
 - NOTE: See submission instructions on ED about the "short answer" questions.
- Final exam: Tuesday, Dec. 20th 9:00 – 11:00AM Location: Skirkanich Auditorium
- Course status:
 - We will be working on compiling the course grades, statistics, etc., during the next week.
 - If you have questions about how you stand in the course, please contact me.

Final Exam Coverage

- The exam is comprehensive, but emphasizes more recent material:
- General Coq principles:
 - Propositions, induction, and inductively-defined propositions
 - Functional programming
- Imp Semantics:
 - large- and small-step operational semantics
 - program equivalence
 - Hoare logic
- Simply-typed Lambda Calculus:
 - operational semantics of higher-order functions, substitution
 - type system and proofs: canonical forms, preservation, progress
 - variants: products, sums, fix, records, etc.
- Subtyping
 - properties of the subtyping relation <:
 - inversion lemmas
- Mutable state
 - references, store typing (details less emphasized since there was no HW)

We will provide definitions, reminders, etc., so there is less emphasis on memorization.

Standard vs Advanced [WPE I]

- As with the previous exams there will be a "standard" track and an "advanced" track.
- PhD Students and WPE-I exam students must take the advanced track
 - It will include some form of written proof
 - WPE-I exams often include more "synthesis" problems (i.e., explore a variant language, proofs, etc.)
- See the example final exams that are available on the course web page.

Remote Exam Option

- Some students with extenuating circumstances have asked to take the exam remotely.
- The remote option will be the same format.
- Released on Gradescope for a 24-hour duration starting at the same time as the in-person exam.
- There will be a 2-hour (+ a bit extra) window to download, complete, (scan if necessary), and upload the exam to Gradescope.

• Send mail to me before Thursday, December 15th if you wish to be considered for the remote option.

COURSE RECAP



CIS 5000

Foundations

- Functional programming
- Constructive logic
- Logical foundations
- Proof techniques for inductive definitions

Semantics

- Operational semantics
- Modeling imperative "While" programs
- Hoare logic for reasoning about program correctness

Type Systems

- Simply typed λ -calculus
- Type safety
- Subtyping
- Dependently-typed programming

Coq interactive theorem prover

– turns doing proofs & logic into programming — fun!

Subset Used in CIS 5000



Coq's Full Capabilities



Can Formal Methods Scale?

Academia

- **Bedrock** web; packet filters
- **CakeML** SML compiler
- **CertiKOS** certified OS kernel
- CompCert C compiler
- **EasyCrypt** crypto protocols
- Kami RISCV architecture
- **HS2Coq** Library validation
- **SEL4** OS microkernel
- Vellvm LLVM IR
- **VST** C software
- **Ynot** DBMS, web services



The Science of Deep Specifications





Deep Specifications

[deepspec.org]



- *Rich* expressive description
- Formal mathematical, machine-checked
- 2-Sided tested from both sides
- *Live* connected to real, executable code

Goal: Advance the reliability, safety, security, and cost-effectiveness of software (and hardware).



case study **SMART CONTRACTS**

Formal Methods for Blockchain

Academic Work:

A Survey of Smart Contract Formal Specification and Verification [Tolmach, et al. 2021]





blockchain



smart contract

- high-level
- human written
- programming language



blockchain



blockchain



contracts offer blockchain services

that can be invoked by other smart contracts



contracts offer blockchain services

that can be invoked by other smart contracts

Vulnerabilities?



[Atzei, et al. 2017]

```
contract SimpleDAO {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
  }
  function withdraw(uint amount) {
    if (credit[msg.sender] >= amount) {
       msg.sender.call.value(amount);
       credit[msg.sender] -= amount;
    }
  }
  function getCredit(address to) {...}
}
```

```
    check the credit
    transfer the amount
    deduct the amount
    from available credit
```

```
[Atzei, et al. 2017]
contract SimpleDAO {
 mapping (address \Rightarrow uint) public credit;
 function pay(address to) {
   credit[to] += msg.value
 }
 function withdraw(uint amount) {
   if (credit[msg sender] >= amount) {
      ms contract Mallory {
      cr
           SimpleDao public dao = SimpleDao(0x354...);
   }
 }
           address owner;
 functio
}
           function Mallory(){owner = msg.sender;}
           function (){ dao.withdraw(dao.getCredit(this)); }
           function stealItAll() { owner.send(this.balance); }
        }
```

```
[Atzei, et al. 2017]
contract SimpleDAO {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
  }
  function withdraw(uint amount) {
    if (credit[msg.sender] >= amount) {
       msg.sender.call.value(amount);
       credit[msg.sender] -= amount;
   }
  }
                          contract Mallory {
  function getCredit(add
                            SimpleDao public dao = SimpleDao(0x354...);
}
                            address owner;
                            function Mallory(){owner = msg.sender;}
                            function (){ dao.withdraw(dao.getCredit(this)); }
                            function stealItAll() { owner.send(this.balance); }
                          }
```

```
[Atzei, et al. 2017]
contract SimpleDAO {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
   }
   function withdraw(uint amount) {
     if (credit[msg.sender] >= amount) {
        msg.sender.call.value(amount);
        credit[msg.sender] -= amount;
    }
   }
                           contract Mallory {
   function getCredit(add)
                             SimpleDao public dao = SimpleDao(0x354...);
}
                             address owner;
                             function Mallory(){owner = msg.sender;}
after setup, invoke
                             function (){ dao.withdraw(dao.getCredit(this)); }
the fallback method
                             function stealItAll() { owner.send(this.balance); }
                           }
```



```
[Atzei, et al. 2017]
contract SimpleDA0 {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
  }
  function withdraw(uint amount) {
    if (credit[msg.sender] >= amount) {
       msg.sender.call.value(amount);
       credit[msg.sepder] -= amount;
    }
  }
                          contract Mallory {
                            SimpleDao public dao = SimpleDao(0x354...);
  function getCredit (add)
}
                            address owner;
   the call operation
                            function Mallory(){owner = msg.sender;}
   transfers some ether
   to Mallory, by calling
                            function (){ dao.withdraw(dao.getCredit(this)); }
   the fallback method
                            function stealItAll() { owner.send(this.balance); }
   again!
                          }
```



```
[Atzei, et al. 2017]
contract SimpleDA0 {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
  }
  function withdraw(uint amount) {
    if (credit[msg.sender] >= amount) {
       msg.sender.call.value(amount);
       credit[msg.sepder] -= amount;
    }
  }
                          contract Mallory {
                            SimpleDao public dao = SimpleDao(0x354...);
  function getCredit (add)
}
                            address owner;
   the call operation
                            function Mallory(){owner = msg.sender;}
   transfers some ether
   to Mallory, by calling
                            function (){ dao.withdraw(dao.getCredit(this)); }
   the fallback method
                            function stealItAll() { owner.send(this.balance); }
   again!
                          }
```



```
[Atzei, et al. 2017]
contract SimpleDAO {
  mapping (address \Rightarrow uint) public credit;
  function pay(address to) {
    credit[to] += msg.value
                                                simple fix:
  }
                                                decrement the credit before
  function withdraw(uint amount) {
                                                transferring the amount.
    if (credit[msg.sender] >= amount) {
       credit[msg.sender] -= amount;
       msg.sender.call.value(amount);
    }
  }
                          contract Mallory {
  function getCredit(add
                            SimpleDao public dao = SimpleDao(0x354...);
}
                            address owner;
                            function Mallory(){owner = msg.sender;}
                            function (){ dao.withdraw(dao.getCredit(this)); }
                            function stealItAll() { owner.send(this.balance); }
                          }
```

Formal Verification



Deep Specification

mathematical model

- hides implementation details
- describes the system behaviors
- implemented in an interactive theorem prover-





Correctness Definition

[Grischenko, et al. 2018]

For the DAO attack, the problem can be characterized as a failure of a property known as **call integrity**, reentrant code controlled by attacker.



Formal Verification


Formal Verification ⇒ Fewer Vulnerbilities



case study
VELLVM – VERIFIED LLVM IR

LLVM Compiler Infrastructure

[Lattner et al.]





LLVM IR = Control-flow Graphs: + Labeled blocks

- + Straight-line Code
- + Block Terminators
- + Static Single Assignment + Phi Nodes

Types: - i64 \Rightarrow 64-bit integers - i64* \Rightarrow pointer

Other LLVM IR Features

- C-style data values
 - ints, structs, arrays, pointers, vectors
- Type system
 - used for layout/alignment/padding
- Relaxed-memory concurrency
 primitives
- Intrinsics
 - extend the language malloc, bitvectors, etc.
- Transformations & Optimizations

Make targeting LLVM IR easy and attractive for developers!



But... it's complex

COMPILER INVASTOCIONE							
LL	.VM Hom	ie l	Documentation »	TV	De		
			 Pointer Typ 	be	pe		
LĽ	VM Lang		 Vector Typ 				
			 Label Type 				
	Abstra		 Token Type 		 'llvm.loop 	unroll and i	an, count' Metad
Ľ	Abstra		 Metadata Ty 		• '11vm.loop	unroll and i	am.disable' Meta
•	introd		 Aggregate I 		 'llym.loop 	unnell and i	an anabja' Mata
	• We		Array Typ Structure		 'llvm.loop 		fptosi to'ln
٠	Identi		Onaque		 'llvm.loop 		Syntax:
	Hiah L		Constants		 'llvm.mem' 		• Overview:
	0 Mo		 Simple Constar 		 'llvm.mem. 		Arguments:
	- 110		 Complex Const 		'irr_loop'		Semantics:
	• Lin		 Global Variable 		 'invariant 		uitofn to'lr
	 Cal 		 Undefined Valu 		 'type' Meta 		Syntax:
	 Vis 		 Poison Values 		 'associate 		Overview:
	• DLI		Addresses of B		 'prof' Meta 		Arguments:
	o The		Constant Expre		 branch_ 		Semantics:
	• •		Other values		Tunction		Example:
	• Rui		Inline Assembli		Module Flags Me	• •	sitofp to' In
	 Strip 		Output o		 Objective-C G 		Syntax:
	• No		 Input cor 		Metadata		Overview:
	o Glo		 Indirect i 		C type width !		Arguments:
	- Gio		Clobber	1	 Automatic Linker 		Semantics:
	- Fur		 Constrair 	ľ	ThinLTO Summa		ptrtoint to
	• Alia		 Supporte 		 Module Path S Clobal Value 6 		Syntax:
	• IFu		 Asm templa 		Giobal value S Eunction S		Overview:
	• Co		 Inline Asm N 		 Global Vari 		Arguments:
	o Na		Metadata		 Alias Sumn 		Semantics:
	- Dec		Specialized		 Function Fl 		Example:
	• Par		DiCompil		 Calls 	• •	inttoptr to
	• Gai		 DIFile 		 Refs 		Syntax:
	• Pre		 DIBasicTy 		 Typeldinfo 		• Overview:
	• Pro		 DISubrou 		 TypeTes 		Arguments:
	O Dor		 DIDerived 		TypeTe		Semantics:
	e Per		 DICompo 		 TypeTes 		bitcast to'
	• Att		 DISubran 		 TypeCh 		Syntax:
	 Fur 		 DIEnume 		 Type ID Summ 		Overview:
	 Glo 		 DITempla 	1	 Intrinsic Global V 		Arguments:
	• Op		 Dillempia 		 The 'llvm.use 		Semantics:
			 DiGloball 		• The livm.com		Example:
	_		 DISubpro 		The live.gld		addrspacecast .
	•		 DILexical 		Instruction Refer		Syntax:
	•		 DILexical 		Terminator In		Overview:
	• Mo		 DILocatic 		 'ret' Instru 		Semantics:
	• Dat		 DILocalVi 		Syntax:		Example:
	0 Tar		 DIExpres 		 Overvier 	• Oth	er Operations
	a Dui		 DIObjCPr 		 Argume 		icmp' Instruction
	• Poi		 Dllmport 		 Semanti 		Syntax:
	 Vol 		 DIMacro 		Example		Overview:
	• Me		thes' Meter		- Dr Instruc		Arguments:
	o Ato		Semantic		 Overvier 		Semantics:
	0 Ela		 Benresen 		 Argume 		Example:
	0 FIO		 'tbaa, struc 		 Semanti 		Suptax:
	• Fas		 'noalias' an 		 Example 		Overview:
	• Use		 'fomath' Met 		 'switch' In 		Arguments:
	• Sou		 'range' Meta 		 Syntax: 		Semantics:
	Type		 'absolute s 		Overvier		Example:
ĺ	o Mer		 'callees' Mo 		Argume	• •	phi' Instruction
	0 VOI		 'unpredicta 		 Jernanti Implementi 		Syntax:
	• Fur		• 'llvm.loop'		 Example 		Overview:
	• Firs		• '11vm.loop.		 'indirectb 		Arguments:
			'llvm.loop.		Syntax:		Semantics:
			• 'llvm.loop.		 Overvier 		Example:
			• '11vm.loop.		 Argume 		Select Instructi
			'llvm.loop.		 Semanti 		Overview:
			'llvm.loop.		 Implement Evample 		Arguments:
			• 'llvm.loop.		· 'invoke' in		Semantics:
			• 'llvm.loop.		 Syntax: 		Example:
			• '11vm.loop.		 Overvier 		call' Instruction

- OVCIVICW.	
 Arguments: 	
 Semantics: 	
 Example: 	
 'catchswitch' Instruction 	 'urem' Instruction
Syntax:	Syntax:
 Overview: 	 Overview:
 Arguments: 	 Arguments:
 Semantics: 	 Semantics:
 Example: 	 Example:
 'catchret' Instruction 	 'srem' Instruction
Syntax:	 Syntax:
 Overview: 	 Overview:
 Arguments: 	 Arguments:
 Semantics: 	 Semantics:
 Example: 	 Example:
 'cleanupret' Instruction 	 'frem' Instruction
Syntax:	Syntax:
 Overview: 	 Overview:
 Arguments: 	 Arguments:
 Semantics: 	 Semantics:
 Example: 	 Example:
 'unreachable' Instruction 	 Bitwise Binary Operati
Svntax:	 'sh1' Instruction
 Overview: 	Syntax:
 Semantics: 	 Overview:
Binary Operations	 Arguments:
 'add' Instruction 	 Semantics:
Svntax:	 Example:
 Overview: 	 '1shr' Instruction
 Arguments: 	Syntax:
_VM	Refe
$\pm \alpha$	

 Example: 	Arg
 'fsub' Instruction 	 Ser
Syntax:	• Exa
 Overview: 	 for' in
 Arguments: 	 Syn
 Semantics: 	 Ove
 Example: 	 Arg
 'mul' instruction 	 Ser
 Syntax: 	• Exa
Overview:	 'xor' li
 Arguments: 	 Syn
 Semantics: 	 Ove
 Example: 	 Arg
 'fmul' instruction 	 Sen
Syntax:	• Exa
 Overview: 	 Vector Op
 Arguments: 	 'extra
 Semantics: 	 Syn
 Example: 	• Ove
 'udiv' Instruction 	 Arg
Syntax:	 Sen
 Overview: 	• Exa
 Arguments: 	 'inser
 Semantics: 	 Syn
 Example: 	 Ove
 'sdiv' Instruction 	 Arg
 Syntax: 	 Ser
Overview:	• Exa
 Arguments: 	 'shuff
 Semantics: 	 Syn
 Example: 	• Ove
 'fdiv' Instruction 	• Arg
 Syntax: 	• Sen
Overview:	Exa
 Arguments: 	 Aggregat
 Semantics: 	'extra
 Example: 	• Syn
 "unon" Instruction 	• Ove
	Arg
	 Sen
	 Exa

	- (Canada - An	. I ta anno 11 an
	• TPLOSI to	
	• Syntax:	 Intrinsic Functi
	 Overview: 	 Variable Arç
Courses	Argument	 'llvm.va
internoty Access a	 Semantics 	 Synta
alloca mstri	 Example: 	 Overv
- Syntax.	• 'uitofp 1	 Argur
Argumontr	Syntax:	 Sema
 Arguments Somantics: 	 Overview: 	 '11vm.va
 Fyample: 	 Argument 	 Synta
 'load' instruct 	 Semantics 	 Overv
Syntax:	 Example: 	Argur
 Overview: 	 'sitofp 1 	 Sema
 Arguments 	Syntax:	 'llvm.va
 Semantics: 	 Overview: 	 Synta
Examples:	 Argument 	Overv
 'store' Instru 	 Semantics 	Arour
Syntax:	Example:	 Sema
 Overview: 	 'ptrtoint 	Accurate Ga
Arguments	Syntax:	 Experime
Semantics:	 Overview: 	• '11vm.ec
Example:	 Argument 	Synta
 'fence' Instru 	Semantics	Overv
Syntax:	Example:	Arour
 Overview: 	inttoptr	- Argui
 Arguments 	Syntax:	111vm ac
 Semantics: 	 Overview: 	Sunta
		 SVIIId

Sema

'llvm.ad

Synta
 Overv

Sema

• '11vm.fr

SyntaOverv

Argui

Sema

Intrinsics

Synta

Overv

Argur

Sema

• 'llvm.re

Intrinsics

Svnta

Overv

Sema

Synta

Overv

Sema

Sema

'llvm.ge

SyntaOverv

Sema

'llvm.pr
 Synta
 Overv

Argur

Sema

Masked V

· ilvm. Syn Ove Arg

Overview:

Semantics:

Syntax:

Overviev

Arguments:

'llvm.exp2.*' Intrinsic

Intrinsics

nsics

'11vm.st SyntaOverv

'llvm.st

'llvm.lo

erence Manu of contents

Arguments:	0	ragamen
emantics:	 Syntax: 	 Semantics
xample:	 Overview: 	 Example:
Instruction	 Arguments 	 Other Operation
yntax:	 Semantics: 	 'icmp' Instru
Overview:	 Example: 	Syntax:
rguments:	 Vector of p 	 Overview:
emantics:	 Conversion Oper 	 Argument
xample:	 'trunc to' 	 Semantics
'Instruction	 Syntax: 	Example:
yntax:	 Overview: 	 'fcmp' Instruct
Overview:	 Arguments 	Syntax:
rguments:	 Semantics: 	Overview
emantics:	 Example: 	Argument
xample:	· 'zext to'l	Semantics
Operations	Syntax:	- Example:
ractelement'	Overview:	inhi! Instruct
yntax:	Arguments	- phi mstruct
Overview:	 Semantics: 	• Syntax:
rguments:	Example:	Overview:
emantics:	Sext to i	 Argument
xample:	Syntax:	 Semantics
ertelement'in	- Overview.	 Example:
yntax:	- Arguments	 'select' Inst
Overview:	Evample:	Syntax:
rguments:	fettours 1	 Overview:
emantics:	Suntay:	 Argument
xample:	Overview:	 Semantics
fflevector' in	Arguments	Example:
yntax:	Semantics:	 'call' Instru
Overview:	Example:	 Syntax:
arguments:	'fpext to'	 Overview:
emantics:	Syntax:	 Argument
xample:	 Overview: 	Semantics
ate Operations	 Arguments 	Example:
ractvalue ins	 Semantics: 	'va_arg' Inst
yntax:	Example:	Syntax:
verview:	• 'fptoui to	Overview:
aguments:	Syntax:	Argument
emanucs:	• Overview:	
xample.	 Arguments 	

				e la			
• '11v	vm.exper	imental.vector.red • 'llvm.exp	eri	mental.v	ector.reduce.and.*'		
Intr	rinsic	Intrinsic					
	Syn	 'llvm.readcyclecounter' Intrinsic 					,
	Ove	Syntax:					
-	Arg	 Overview: 					
• '11v	vm.	 Semantics: 			r.reduce.or.*'		
Intr	rins	 'llvm.clear_cache' Intrinsic 					
	Syn Ow	 Syntax: 					
	Arc	Overview:					
111	vm.	 Semantics: 'llym_instance_instance 			n reduce von **		
Intr	rins	 Syntax: 			cuter		
	Syn	Overview:			'llvm.load.relative'	Intrir	sic
- (Ove	 Arguments: 			Syntax:		
	Arg	 Semantics: 			 Overview: 		
• '11v	vm.	 'llvm.instrprof.increment.step' Int 			11 um cideoffoct' int	incie	
Intr	rins	Syntax:			- Combany	msic	
	Syn	Overview:			 Syntax: Oversidents 		
	Arc	 Arguments: Semantics: 			Overview:		
111	vm.	 'llvm.instrprof.value.profile' intri 			 Arguments: 		
Intr	rins	Syntax:			 Semantics: 		
	Syn	Overview:		0	Stack Map Intrinsics		
- (Ove	 Arguments: 		0	lement Wise Atomic Mer	nory	Intrinsics
	Arg	 Semantics: 			'llvm.memcpy.element	. unor	rdered.atomic
- '11y	vm.	 'llvm.thread.pointer' Intrinsic 			Intrinsic		
	IS	 Syntax: Overview: 			Syntax:		
	<u>n</u>	 Semantics: 			 Overview: 		
		 Standard C Library Intrinsics 			 Arguments: 		
	9	 'llvm.memcpy' Intrinsic 			Semantics:		
	s	 Syntax: 			Lowering:		
	m	Overview:			'llvm.memmove.elemen	t.unc	ordered.atomic'
	<i>.</i>	Arguments:			Intrinsic		
	g	Semantics:			Syntax:		
	•	 Syntax: 			 Overview: 		
	IS	Overview:			 Arguments: 		
	Ove	 Arguments: 			Semantics:		
-	Arc	 Semantics: 			Lowering:		
- '11	vm.	 'llvm.memset.*' Intrinsics 			'llvm.memset.element	. unor	rdered.atomic'
Intr	rins	Syntax:			Intrinsic		
	Syn	Arguments:			Syntax:		
- (Ove	 Semantics: 			Overview:		
	Arg	'llvm.sqrt.*' Intrinsic			Arguments:		
Half Pr	reci	 Syntax: 			Semantics:		
	viii. Sun	 Overview: 			Lowering:		
	Ove	Arguments:					
- /	Arg	Semantics:		Abstract			
	Sen	Suntax:					
	Exa	Overview:		This do	cument is a reference m	anual	for the LLVM assembly
• '11v	vm.	 Arguments: 		langua	ge. LLVM is a Static Single	a Assi	ignment (SSA) based
-	Syn	 Semantics: 		represe	ntation that provides typ	e saf	ety, low-level opera-
	Arc	 'llvm.sin.*' Intrinsic 		tions, f	exibility, and the capabi	ity of	representing 'all'
	Sen	Syntax:					
- 1	Exa	 Arguments: 					
 Debug 	gei	Semantics:					
 Except 	tior	 'llvm.cos.*' Intrinsic 					
Tramp	oli	 Syntax: 					
11	vm.	Overview:			trinsic		
	Ove	Arguments: Somantics:					
	Arc	 Semanucs: 'llym.pow.*' Intrinsic 					
	Sen	 Svntax: 					
- '11	vm.	Overview:			Intrinsic		
• :	Syn	 Arguments: 					
- (Ove	Semantics:					
	Arg	 'llvm.exp.*' Intrinsic 					
		- SWITAX					

One Example: undef

The **undef** "value" represents an arbitrary, but indeterminate bit pattern for any type.

Used for:

- uninitialized registers
- reads from volatile memory
- results of some underspecified operations

What is the value of **%y** after running the following?

One plausible answer: 0 Not LLVM's semantics!

(LLVM is more liberal to permit more aggressive optimizations)

Partially defined values are interpreted *nondeterministically* as sets of possible values:

Interactions with Optimizations

Consider:

versus:



Interactions with Optimizations

Consider:

versus:

Upshot: if %x is **undef**, we can't optimize **mul** to **add**

What's the problem?

Bug List: (12 of 435) First Last Prev Next Show last search results					
Bug 33165 - Simpify* cannot distribute instructions for simplification due to une	def				
Status DEODENED	Benerted: 2017-05-25 02:12 DDT by Nune Longs				
Davide Italiano 2017-05-25 08:55:40 PDT	<u>Comment 6</u>				
cc Davide Italiano 2017-05-25 09:05:26 PDT	Comme				
nc (unless we want to give up on some undef tran but I'm afraid others might be affected too)	sformations, and special case sel ϵ				
B' John Regehr 2017-05-25 09:09:24 PDT	<u>Com</u> ı				
Yes, this is one of those test cases. There Nuno has been automatically filtering out c to be hard to fix but I guess he decided to	are so many optimization failures lasses of mistranslation that are take a closer look at some of the				
Soon I'll be able to include branches/phis branches due to a limitation in Alive.	in these test cases, but only forw				

LLVM Compiler Infrastructure

[Lattner et al.]



The Vellvm Project

[Zhao et al. POPL 2012, CPP 2012, PLDI 2013, Mansky et al. CAV2015]



Vellvm Framework



Vellvm Framework



VELLVM – Previous Results

- SoftBound [POPL 2012]
 - Memory Safety
- mem2reg [PLDI 2013]
 - Register promotion, defined in terms of a stack of "micro-optimizations"
- Verified dominator analysis [CPP 2012]
 - Cooper-Harvey-Kennedy Algorithm [2000]
- Better memory models
 - ptrtoint casts [PLDI 2015]
 - modular formalization [CAV 2015]
- Better representation for effectful programs in Coq [POPL2020]
- Improved LLVM IR semantics [ICFP 2021]



Modular LLVM IR Semantics

- Core: LLVM control-flow-graph interpreter
- Memory Model [CAV 15]
 - interprets loads/stores
 - support for casts [PLDI 15]
 - other effects





LLVM Interpreter in Coq



Vellvm Effects

- 1. External Calls
- 2. Intrinsics
- 3. Global Environment
- 4. Local Environment
- 5. Stack
- 6. Memory
- 7. Nondeterminism
- 8. Undefined Behavior
- 9. (Debugging)

Each layer of effects can be interpreted separately, making proofs *modular* and changes orthogonal.





Compositionality

DENOTATIONAL SEMANTICS

Compositional LLVM Semantics

- $[\![]\!]_{expr}$ expressions
- [-]_{instr} instructions
- [-]_{block} blocks

Meaning built up by induction on the structure of the syntax.

- open programs
- fixpoint combinators
- pure monadic computations

- [-]_{cfg} control-flow graphs
- [[]]_{IIvm} programs
 (mutually recursive functions)

Operational Semantics

- not defined purely by induction on syntax
- extra "bookkeeping": evaluation contexts, program counters, etc.

- 🛞

Compositional Reasoning

$$\begin{split} \underbrace{ \begin{array}{l} \operatorname{outputs}(cfg_2) \cap \operatorname{inputs}(cfg_1) = \emptyset & to \notin \operatorname{inputs}(cfg_1) \\ \hline \llbracket cfg_1 + cfg_2 \rrbracket_{\mathrm{cfg}}(f, to) \approx \llbracket cfg_2 \rrbracket_{\mathrm{cfg}}(f, to) \\ \hline \llbracket cfg_1 + cfg_2 \rrbracket_{\mathrm{cfg}}(f, to) \approx \llbracket cfg_1 \rrbracket_{\mathrm{cfg}}(f, to) ;; \\ \operatorname{match} x \text{ with } | \ \operatorname{inl} fto \Rightarrow \llbracket cfg_1 + cfg_2 \rrbracket_{\mathrm{cfg}} fto \\ | \ \operatorname{inr} v \Rightarrow \operatorname{ret} v \\ \\ \underbrace{ \begin{array}{l} \operatorname{independent_flows} cfg_1 cfg_2 & to \in \operatorname{inputs}(cfg_i) \\ \llbracket cfg_1 + cfg_2 \rrbracket_{\mathrm{cfg}}(f, to) \approx \llbracket cfg_i \rrbracket_{\mathrm{cfg}}(f, to) \\ \end{array} }_{ \end{tabular} Flow \\ \end{array}} \operatorname{Flow} \end{split}} \end{split}}$$

Rules to reason about control-flow graph composition.

Lemmas about "contextual program equivalences".

Exectuability
VELLVM DEMO

So What?

- Debugging
- Validate Vellvm Semantics against other implementations
 - test suite of ~140 "semantic tests"
 - e.g., integrate with QuickChick, Csmith, ALIVE
- Find bugs in the existing LLVM infrastructure
 - thinking hard about corner cases while formalizing is a good way to find real bugs
 - identify inconsistent assumptions about the LLVM compiler

Deep Specifications



Rich – expressive description
Formal – mathematical, machine-checked
2-Sided – tested from both sides
Live – connected to real, executable code

SPIRAL / HELIX

[Püschel, et al. 2005] [Franchetti et al., 2005, 2018] [Zaliva et al., 2015 2018, 2019]

DSL for high-performance numerical computing.



HELIX Compilation Pipeline



Compiler Correctness Proof



WHAT NEXT?

After CIS 5000

Research Conferences:

- Certified Programs and Proofs (CPP)
- Interactive Theorem Proving (ITP)
- Theorem Proving and Higher-order Logics (TPHOLS)
- Principles of Programming Languages (POPL)
- Programming Languages Design and Implementation (PLDI)
- International Conference on Functional Programming (ICFP)

- ...

Other Theorem Provers:

- LEAN
- Isabelle/HOL
- Agda
- F*

After CIS 5000

More Software Foundations Volumes

- written in the same style as SF and PLF



Thanks!

• To our three fantastic TAS:



Jessica Shi



Adam Stein



Joey Velez-Ginorio

And, thanks to all of you!