Setting the Stage: on the Mechanized Verification of a Compiler

Compilation and Program Analysis (#11)

2024/2025

Yannick Zakowski







Introduction

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow:

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow: "We will bury you"

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow: "We will bury you"

It appears that the original Russian sentence something closer to "We shall outlive you"

- In 1956, Nikita Khrushchev is addressing western diplomats in Moscow: "We will bury you"
 - It appears that the original Russian sentence something closer to "We shall outlive you"

Khrushchev had a buggy translator!

¹ The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

A nightmare scenario...

Our algorithm satisfies the specification

¹ The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

A nightmare scenario...

Our algorithm satisfies the specification Better! Our implementation satisfies the Specification

- A nightmare scenario...

Our algorithm satisfies the specification Better! Our implementation satisfies the Specification But the **COMPILE** has changed the meaning of my program

- A nightmare scenario...

Our algorithm satisfies the specification Better! Our implementation satisfies the Specification But the **COMPILE** has changed the meaning of my program

- A nightmare scenario...
- The executable code does not satisfy the Specification

Our algorithm satisfies the specification

Better! Our implementation satisfies the Specification

But the **COMPILER** has changed the meaning of my program

Natural languages are hard. But when it comes to programming languages, can we guarantee that our translators won't betray us?

- A nightmare scenario...
- The executable code does not satisfy the Specification





Note: gcc is composed of roughly 15 millions line of codes...





input

Note: gcc is composed of roughly 15 millions line of codes...



How to generate inputs?

input



Valid C program fit to stress test your compiler

Note: gcc is composed of roughly 15 millions line of codes...



How to generate inputs?



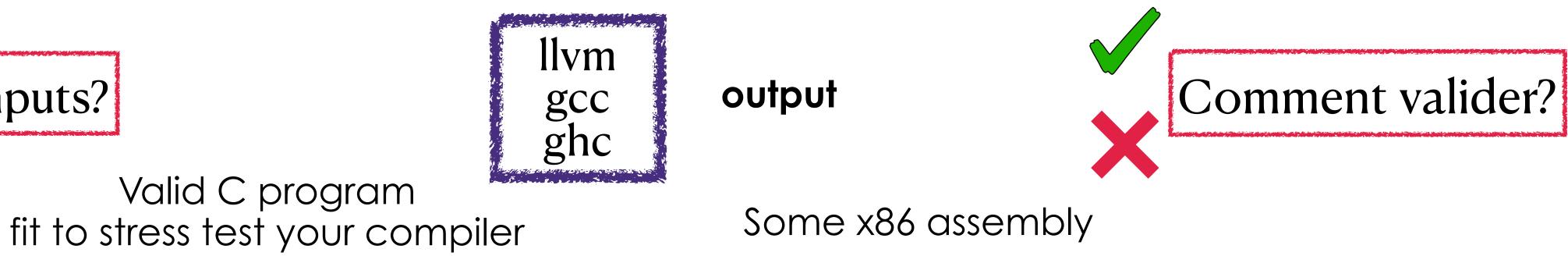
Note: gcc is composed of roughly 15 millions line of codes...

A compiler is a program, and we want it to behave: can't we just test them?



output

How to generate inputs?



Note: gcc is composed of roughly 15 millions line of codes...

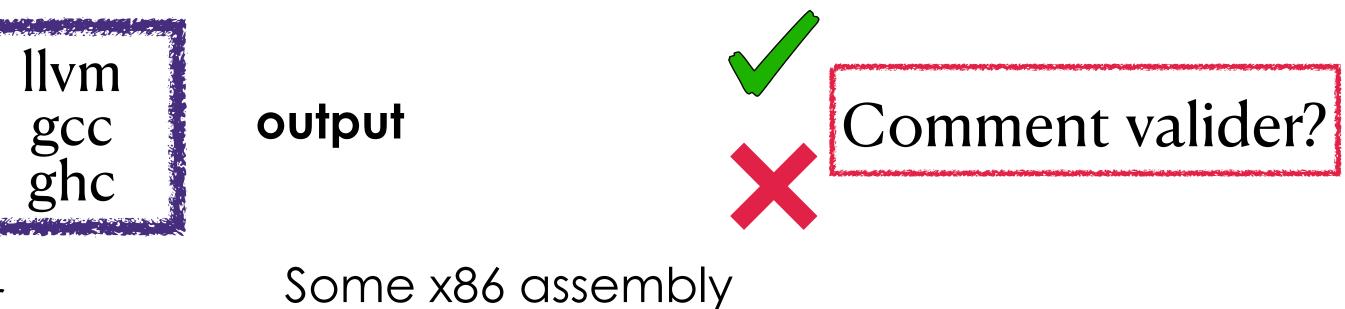
How to generate inputs?



Valid C program fit to stress test your compiler

Note: gcc is composed of roughly 15 millions line of codes...

A compiler is a program, and we want it to behave: can't we just test them?



Simplest solution: differential testing. Have your compilers argue!

It's hard work, but it can be done! Finding and Understanding Bugs in C Compilers

Xuejun YangYang ChenEric EideJohn Regehr

Write a C program random generator (CSmith)
 Have several (>= 3) C compilers run the programs and vote on the result

But C is no ML: a syntactically correct program is likely no C! Undefined behaviours: null pointer dereference, array access out-of-bound, etc... Your random generator must be paired with complex static analyses

It's hard work, but it's worth it! Finding and Understanding Bugs in C Compilers

Xuejun YangYang ChenEric EideJohn Regehr

"Every compiler we tested was found **to crash** and also **to silently generate wrong code** when presented with valid input."

It's hard work, but it's worth it! Finding and Understanding Bugs in C Compilers

Xuejun YangYang ChenEric EideJohn Regehr

"Every compiler we tested was found **to crash** and also **to silently generate wrong code** when presented with valid input."

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers **are absent**."

Here enters the hero of our story: the Verified compiler

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

Introduction 1

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Pre-history

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

Introduction 1

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Pre-history

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

Introduction 1

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language. The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that

compilers are correct.

Pre-history

Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch Computer Science Department Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history

Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch Computer Science Department Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history

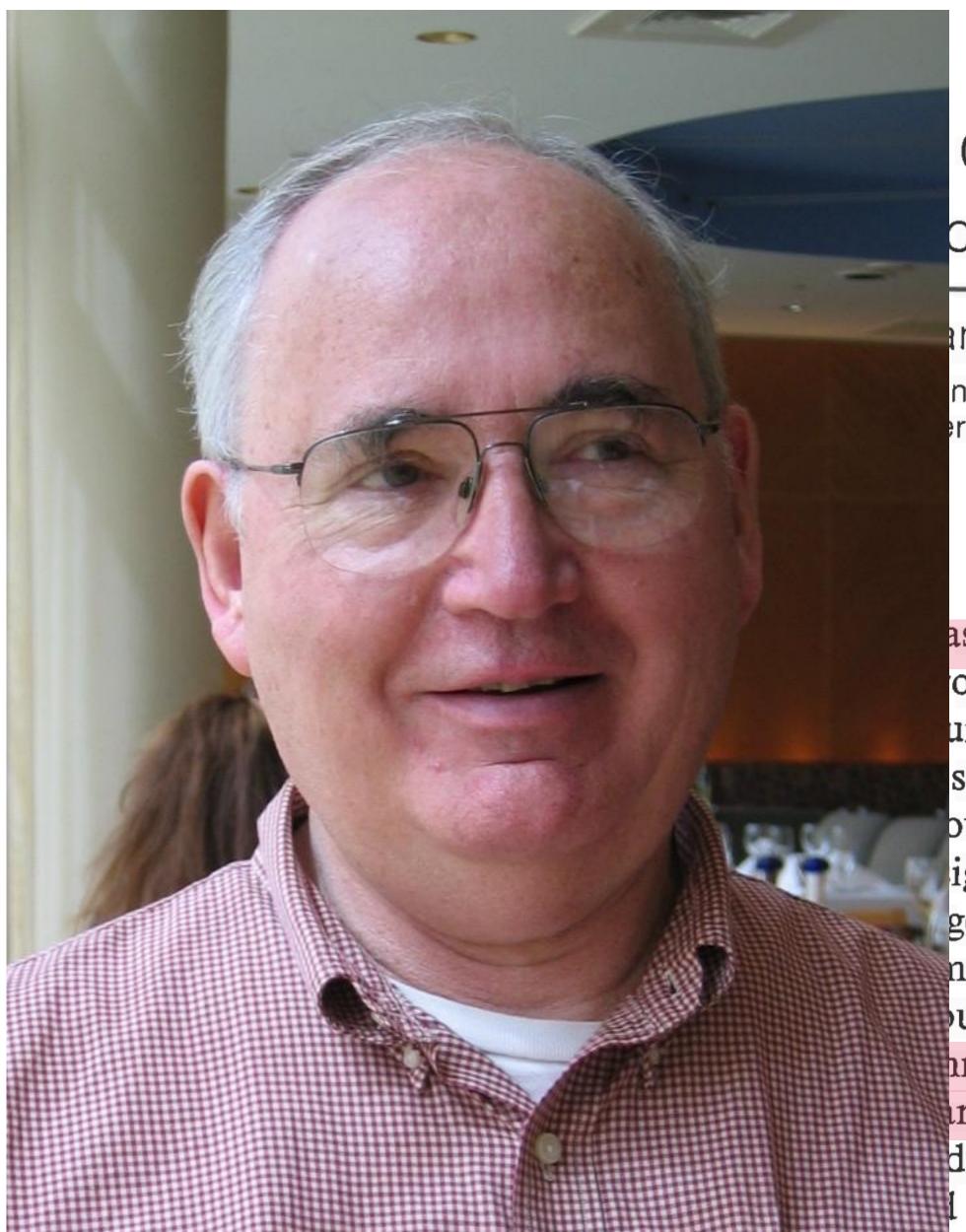
Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch Computer Science Department Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history



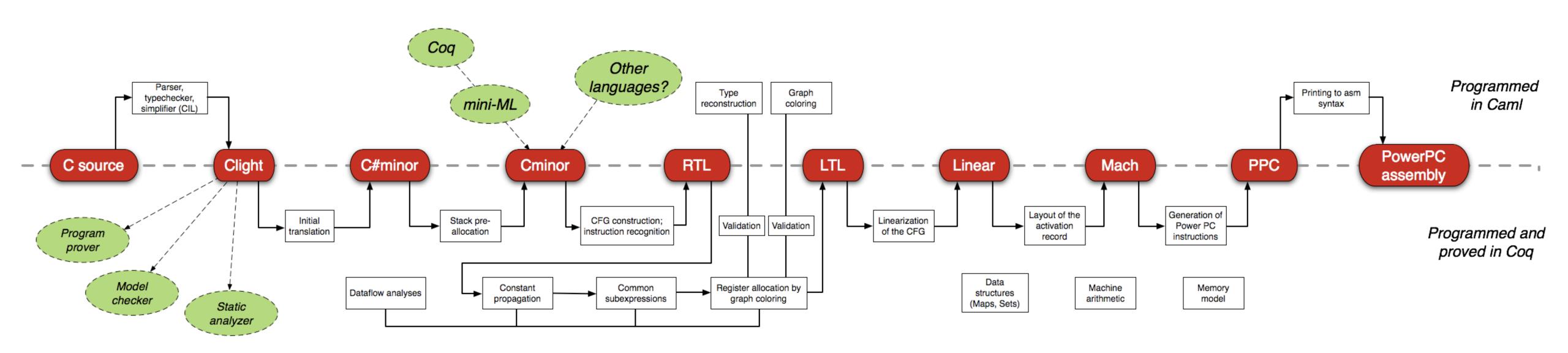
Pre-history Compiler Correctness chanized Logic

and R. Weyhrauch nce Department ersity

ask of machine-checking the proof of a simple compiling oof-checking program is LCF, an implementation of a logic unctions due to Dana Scott, in which the abstract syntax semantics of programming languages can be naturally ource language in our example is a simple ALGOL-like ignments, conditionals, whiles and compound statements. ge is an assembly language for a machine with a pushdown nethods are used to give structure to the proof, which is utline. However, we present in full the expression-compiling hm. More than half of the complete proof has been machine inticipate no difficulty with the remainder. We discuss our ducting the proof, which indicates that a large part of it to reduce the human contribution.

Turned 90 Yesterday! 1976's Turing Prize (with Rabin)

CompCert (2009-) : a Verified C99 Optimising Compiler



If CompCert successfully compiles a C source program p down to a PowerPC assembly program <mark>asm</mark>, then « <mark>asm</mark> and p behave the same »

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

Why does CompCert interest so much these industries?

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

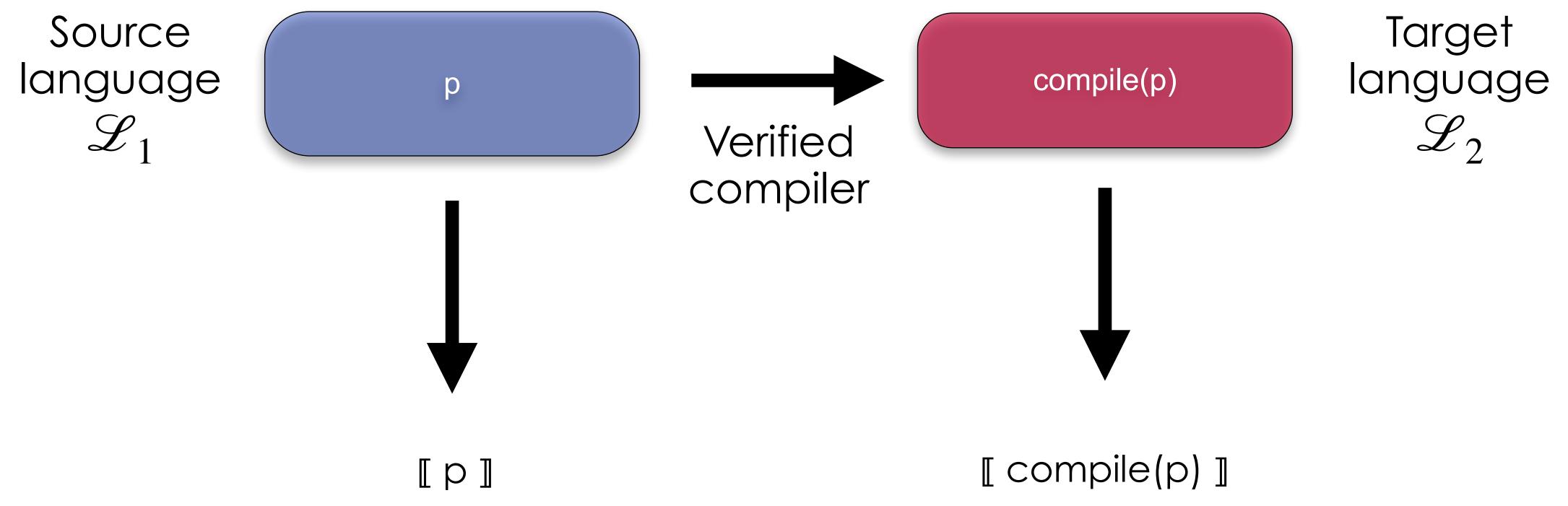
- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

Why does CompCert interest so much these industries?

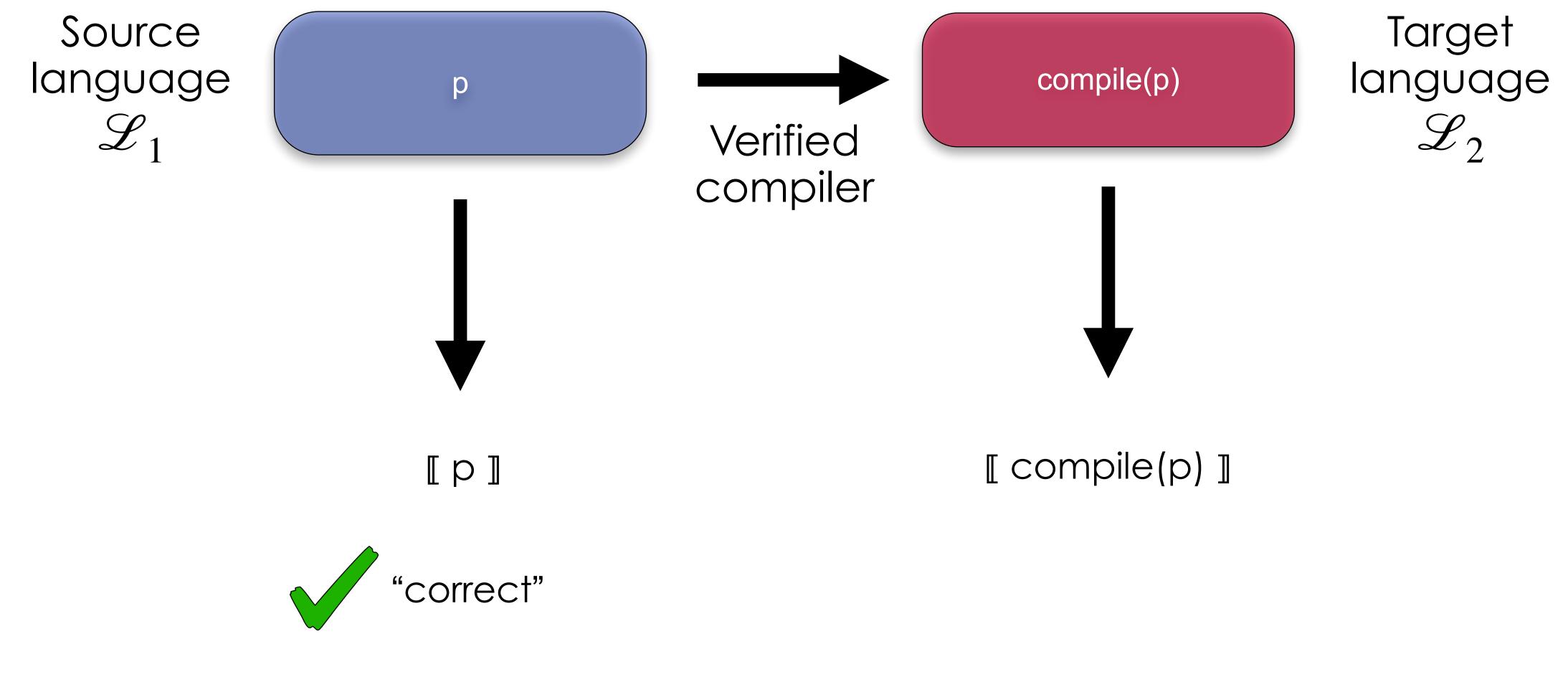
Paradoxically, not so much to increase trust than to improve performances!

"With CompCert it is possible to decrease the execution time of our flight control algorithms by a significant amount" (TUM)

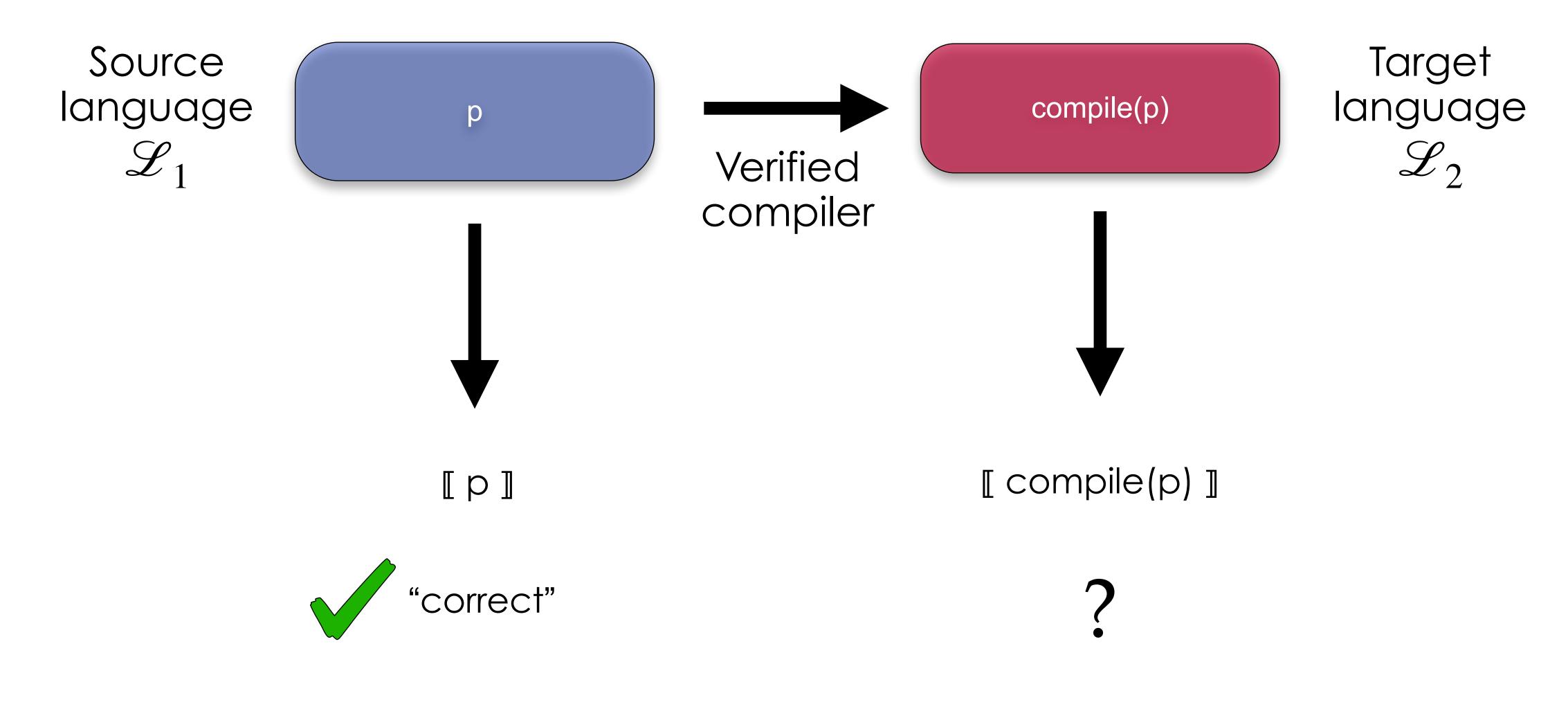
The standards for certification are extremely stricts for such fields: optimisations were usually completely ruled out!



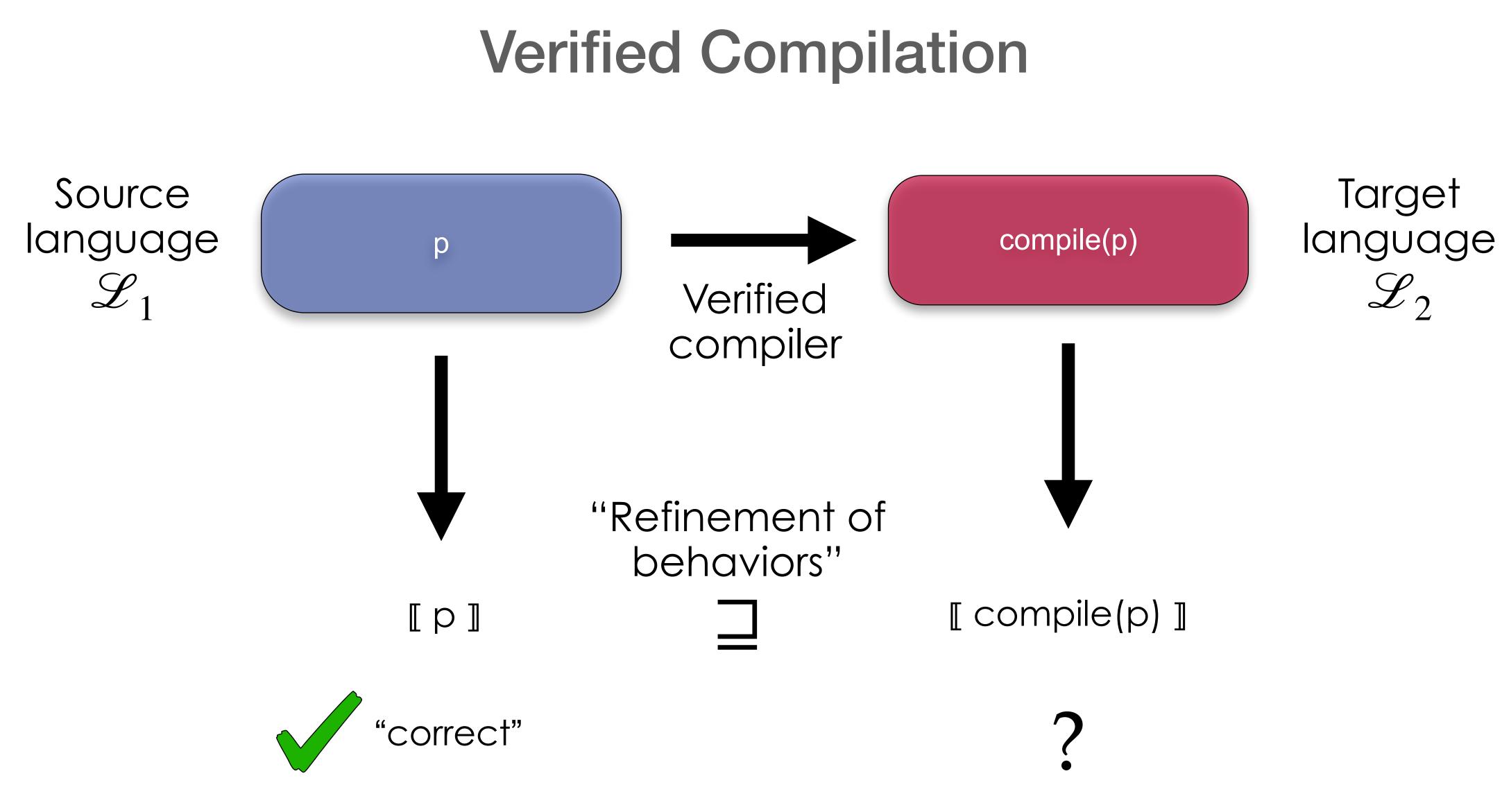
Verified Compilation

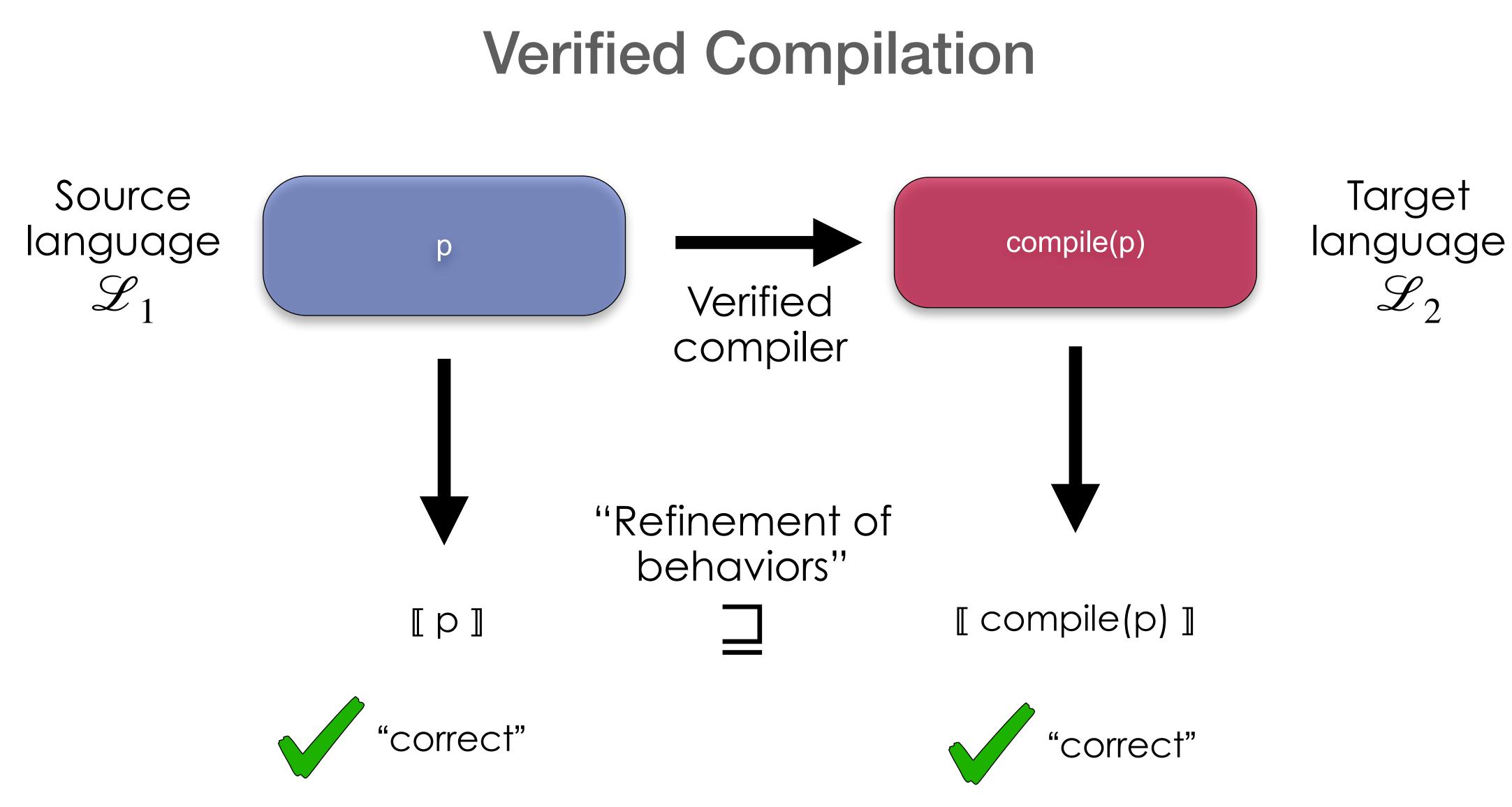


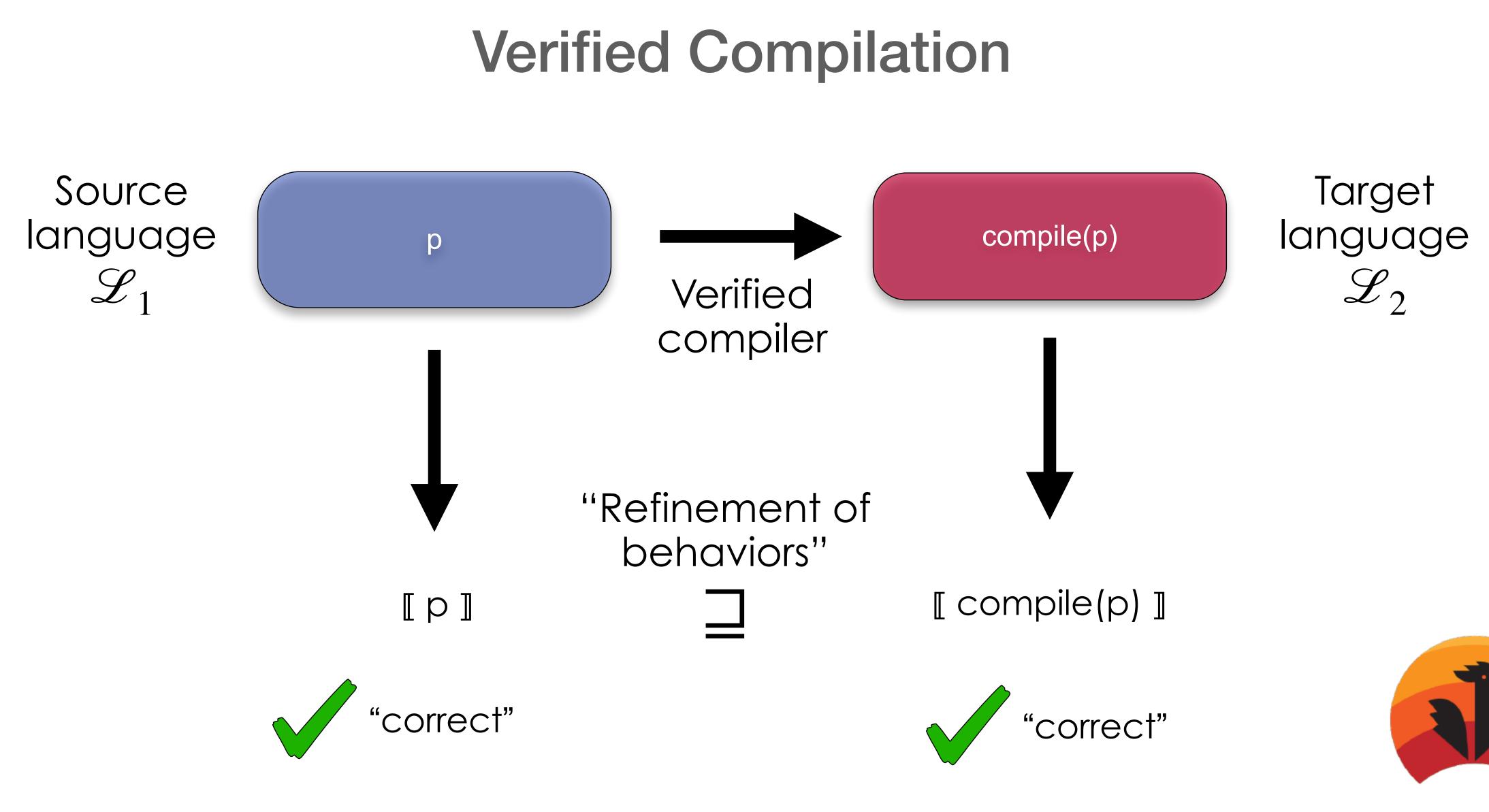
Verified Compilation



Verified Compilation

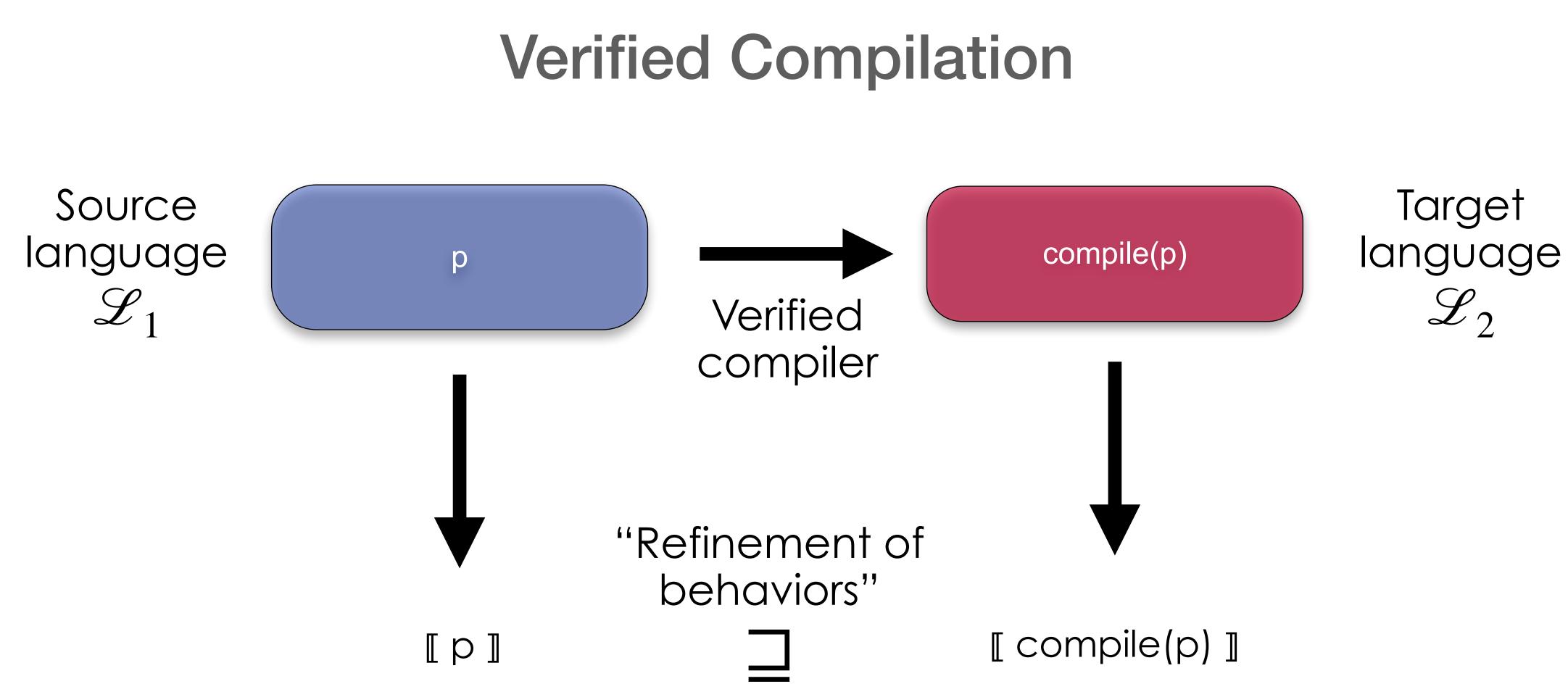




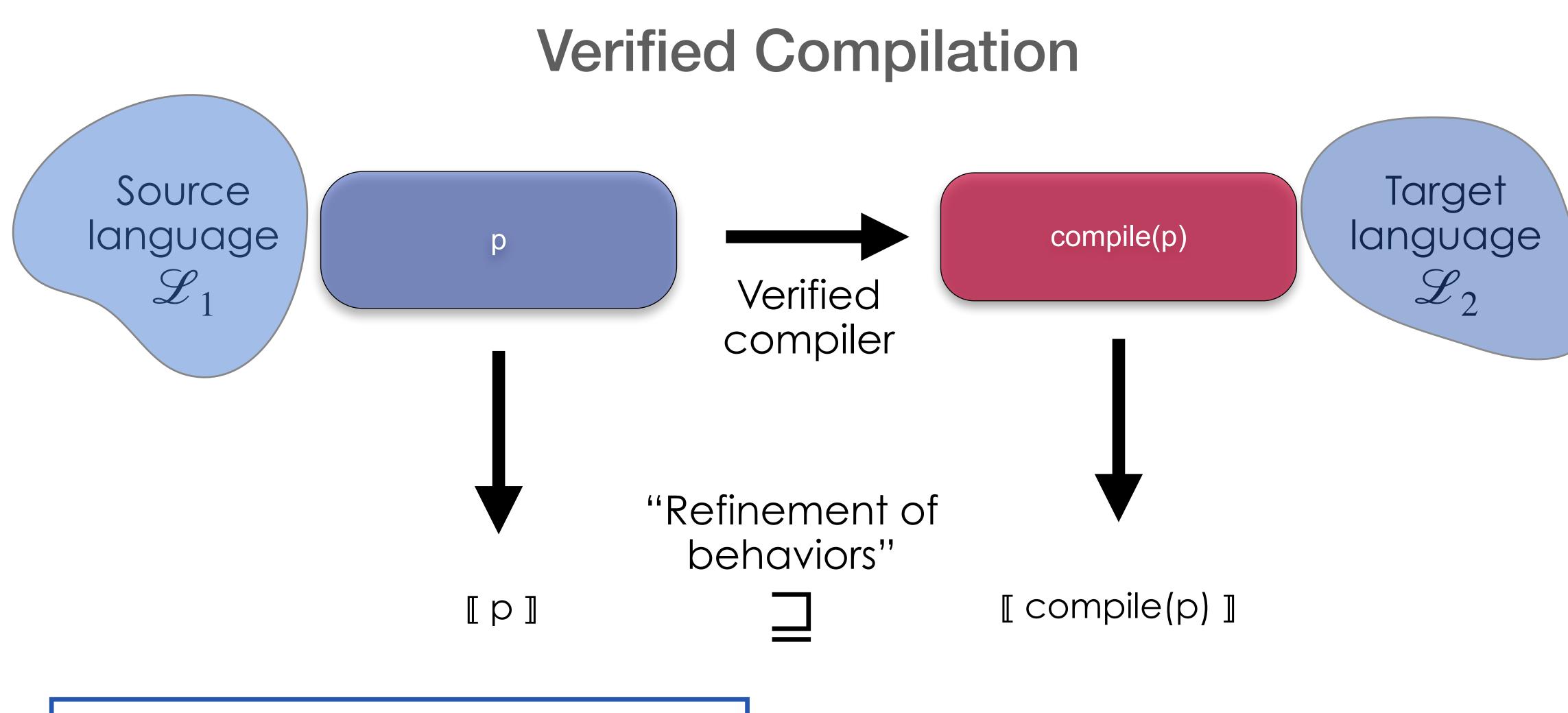


The compiler is written and formally proved correct in a Proof Assistant





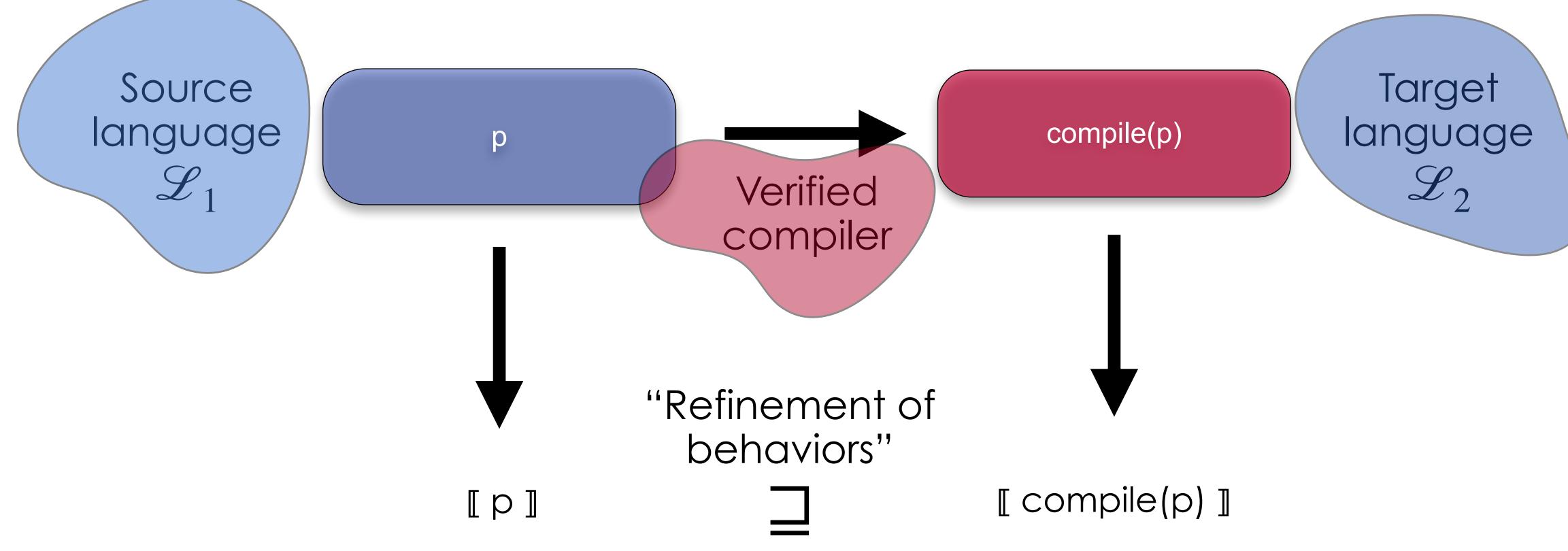




New languages, new constructions





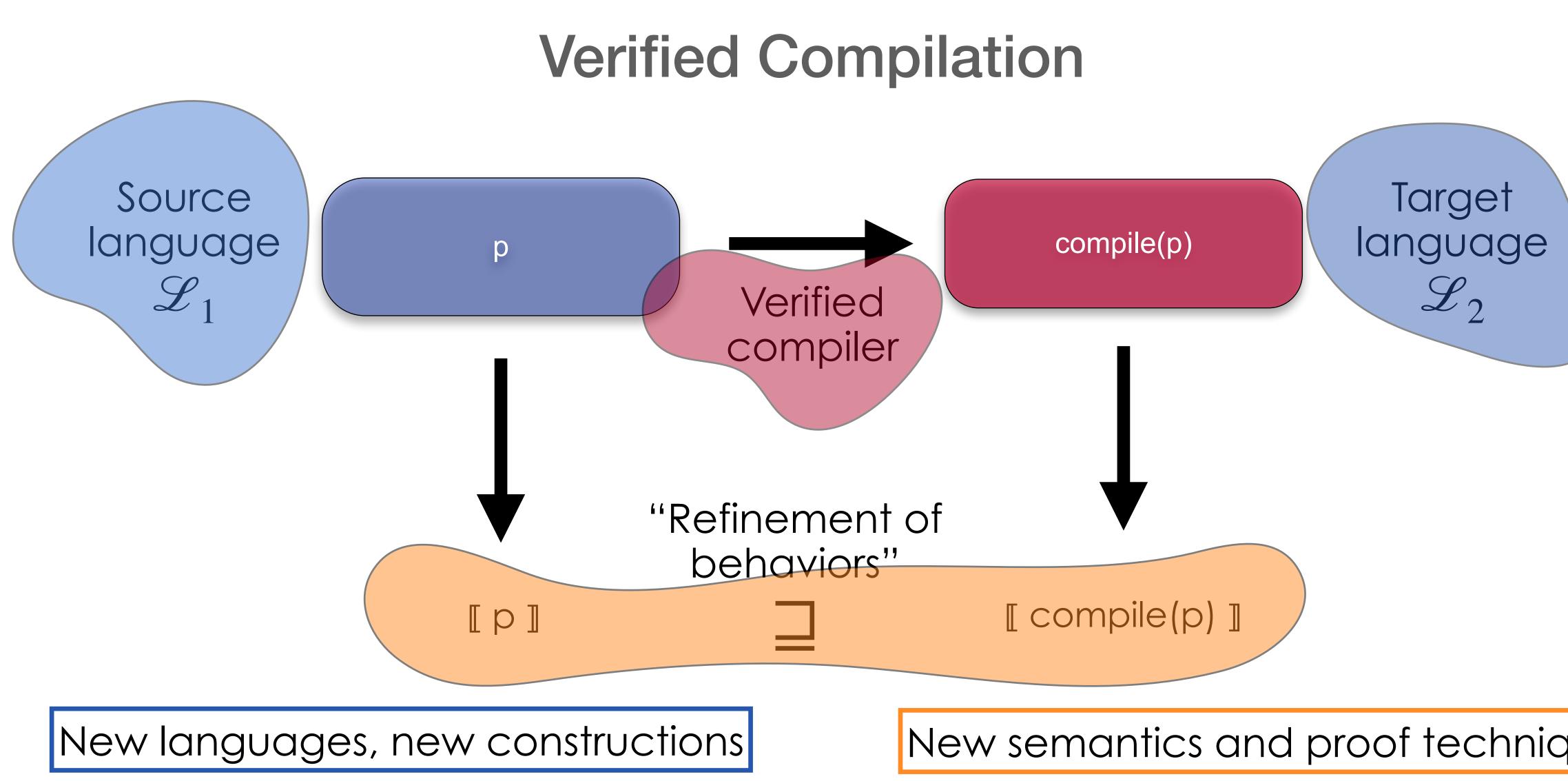


New languages, new constructions

Supporting better/more optimizations

Verified Compilation





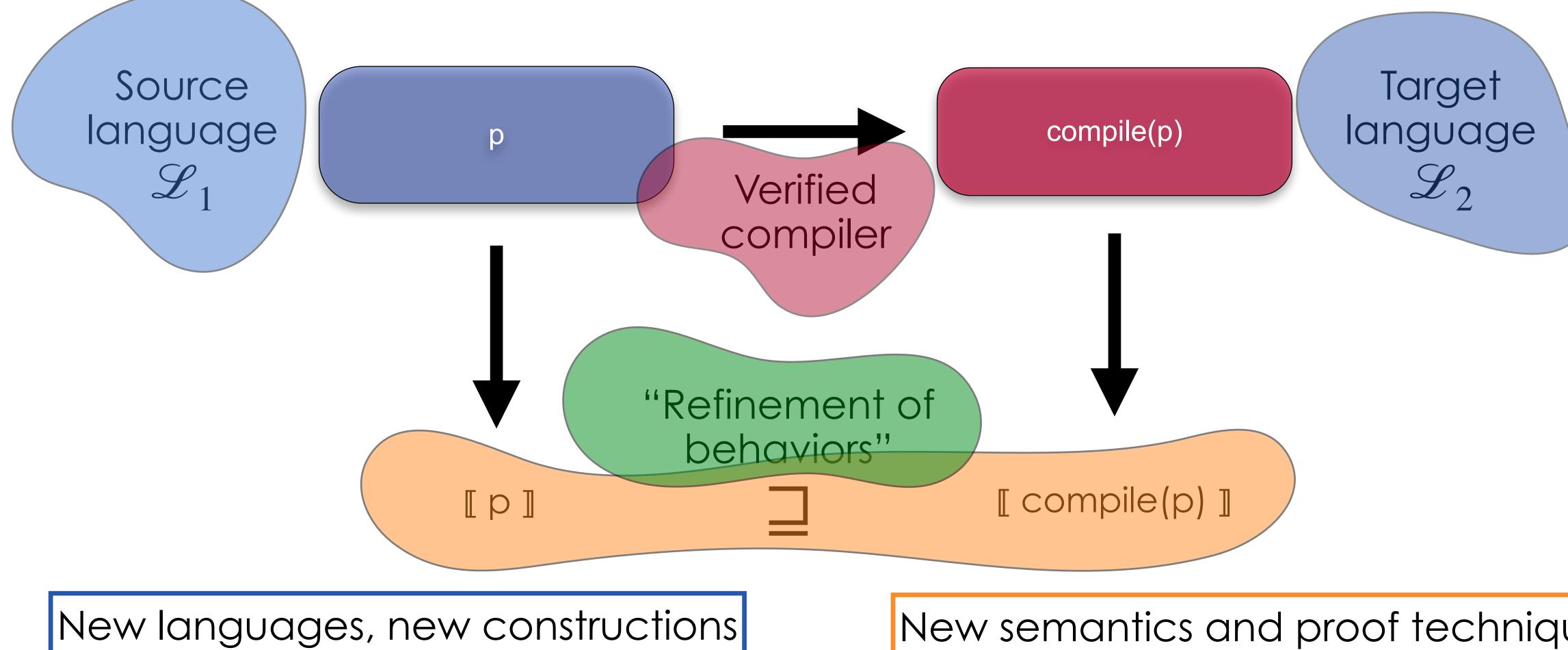
Supporting better/more optimizations

New semantics and proof techniques









Supporting better/more optimizations

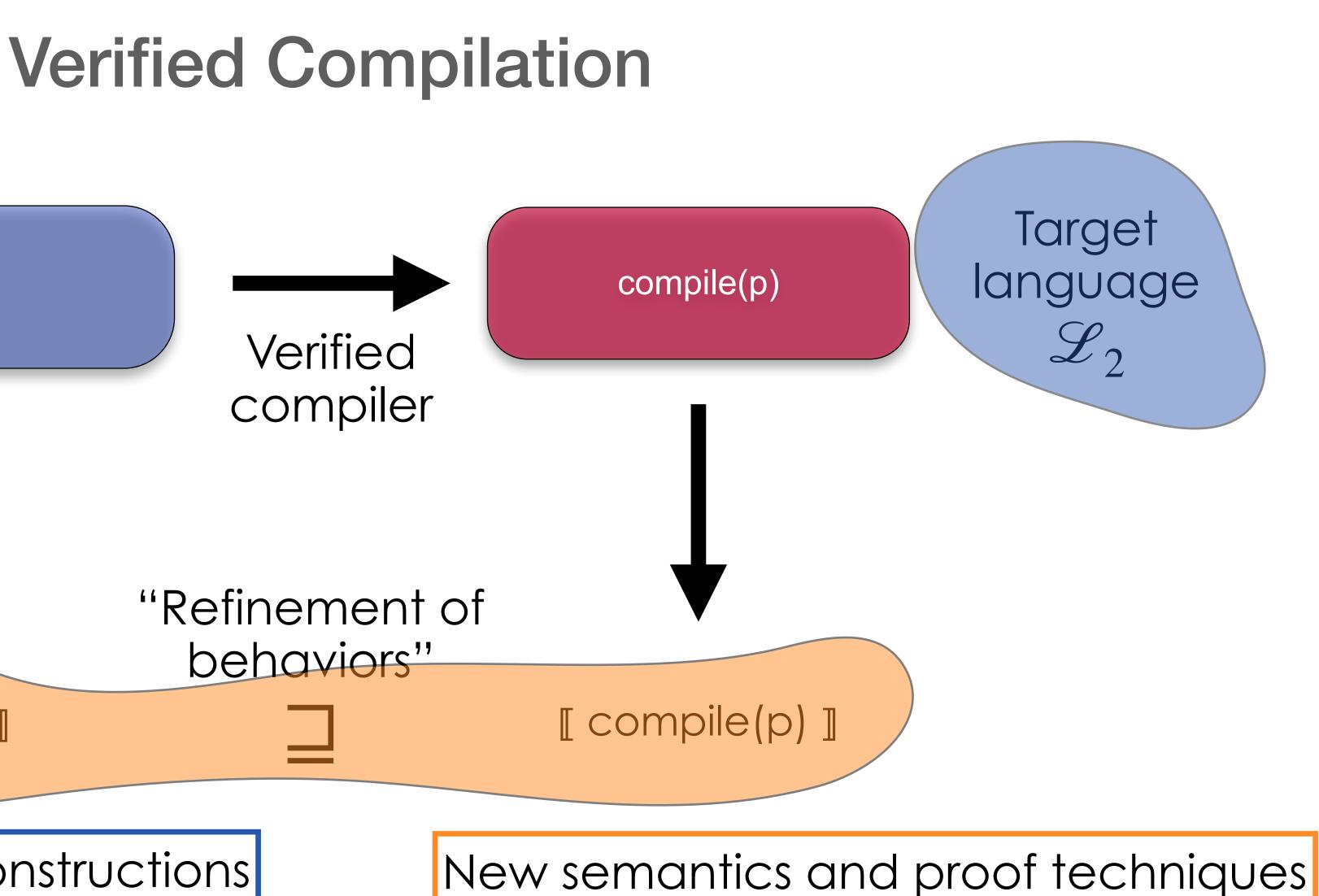
Verified Compilation

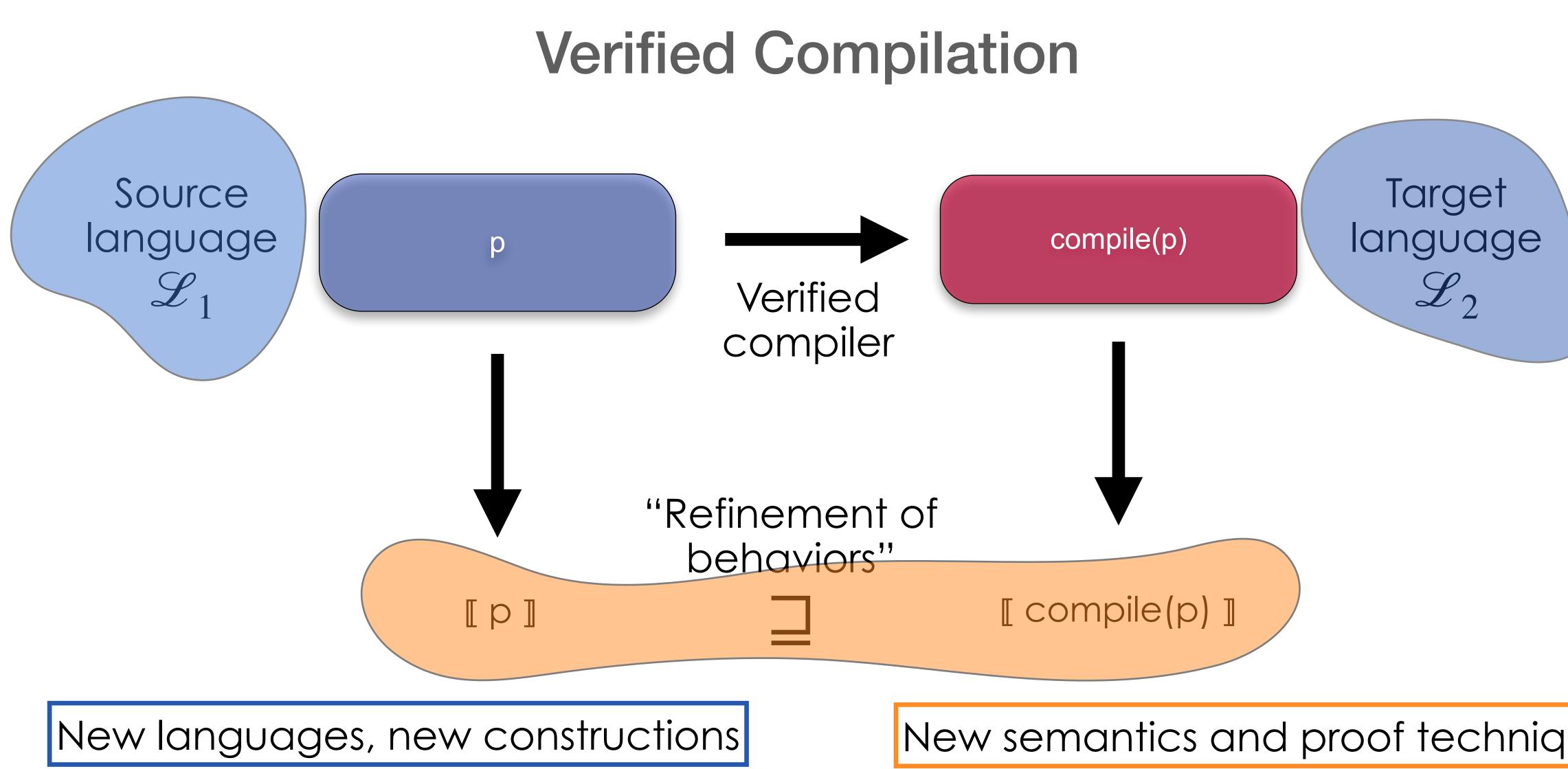
New semantics and proof techniques

Stronger results (security, ...)







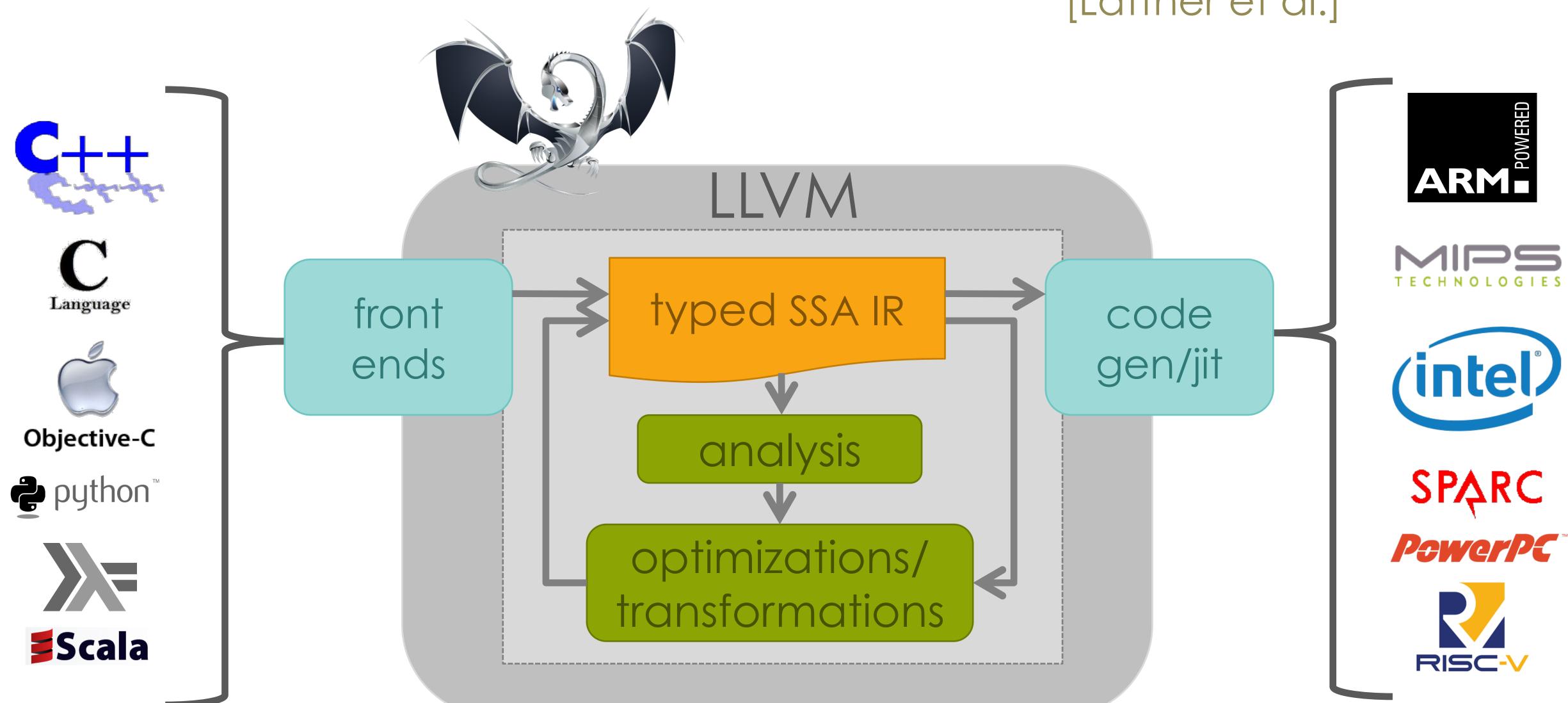


A modular, compositional, and executable semantics for LLVM IR





LLVM Compiler Infrastructure [Lattner et al.]



LLVM Reference Manual table of contents

The LLVM IR







LLVM Home	Documentation»	VDA		
	 X86_mmx Type Pointer Type 			
LLVM Lang	 Vector Type 			
	 Label Type 			
	 Token Type 	<pre> 'llvm.loop.un</pre>	roll_and_jam'	
 Abstra 	 Metadata Ty 	<pre> 'llvm.loop.un</pre>	<pre>roll_and_jam.count' Meta</pre>	
Introd	 Aggregate T 	-	<pre>roll_and_jam.disable' M</pre>	
• We	 Array Typ 	<pre> 'llvm.loop un</pre>	roll and jam enable' Me	
	 Structure 	<pre> 'llvm.loop</pre>	'fptosi to'	
 Identif 	Opaque 5	<pre> 'llvm.loop</pre>	Syntax:	
High L	Constants	<pre>'llvm.mem'</pre>	Overview:	
• Mo	 Simple Constar 	<pre> 'llvm.mem.</pre>	 Arguments: 	
	 Complex Const 	<pre> 'irr_loop'</pre>	 Semantics: Example: 	
• Lin	 Global Variable 	• 'invariant	Example: 'uitofp to'	
• Cal	 Undefined Valu 	 'type' Meta 	Syntax:	
• Vis	Poison Values	 'associate 	 Overview: 	
• DLI	 Addresses of Ba 	 'prof' Meta 	 Arguments: 	
	 Constant Expre 	branch_	 Semantics: 	
• Thr	Other Values	 functior 	 Example: 	
• Rur	• Inline Assemble	• VP	 'sitofp to' 	
• Stri	Inline Asm C	Module Flags Me	 Syntax: 	
	Output ce	 Objective-C G 	 Overview: 	
• Noi	Input cor	Metadata	 Arguments: 	
• Glo	 Indirect in 	 C type width I Automatic Linker 	 Semantics: 	
• Fur	 Clobber (Automatic Linker ThinLTO Summar 	Example:	
• Alia	 Constrair 	Module Path S	• 'ptrtoint t	
	 Supporte 	Global Value	Syntax:	
• IFu	 Asm templar 	 Function St 	Overview:	
• Cor	 Inline Asm N 	 Global Vari 	 Arguments: 	
• Nar	Metadata	 Alias Sumn 	 Semantics: 	
	 Metadata Node Specialized I 	 Function Fl 	Example:	
• Par	opecianzea	Calls	inttoptr t	
• Gar	DICompilDIFile	 Refs 	Syntax:	
• Pre	 DIFile DIBasicTy 	 TypeldInfo 	 Overview: 	
• Pro	 DISubrou 	TypeTe:	 Arguments: 	
	 DIDubiou DIDerived 	 TypeTes 	 Semantics: 	
• Per	 DICompo 	 TypeCh TypeTage 	 Example: 	
• Att	 DISubran 	TypeTe:TypeCh	• 'bitcast to	
• Fur	 DIEnume 	 Type ID Summ 	Syntax:	
	 DITempla 	 Intrinsic Global V 	Overview:	
• Glo	 DITempla 	• The 'llvm.use	 Arguments: 	
• Op	 DINames 	• The 'llvm.com	 Semantics: 	
	DIGlobal	• The 'llvm.glo	 Example: 	
	 DISubpro 	• The 'llvm.glo	 'addrspacecast Syntax: 	
_	 DILexical 	Instruction Refer	Syntax:Overview:	
•	 DILexical 	Terminator In	 Arguments: 	
• Mo	 DILocatio 	 'ret' Instru 	 Arguments. Semantics: 	
• Dat	 DILocalV; 	Syntax:	 Example: 	
	 DIExpress 	 Overvier 	Other Operations	
• Tar	 DIObjCPr 	 Argume 	 'icmp' Instruction 	
• Poi	 Dllmport 	 Semanti 	 Syntax: 	
• Vol	 DIMacro 	 Example 	 Overview: 	
• Me	 DIMacroF 	 'br' Instruc 	 Arguments: 	
	 'tbaa' Metac 	Syntax:	 Semantics: 	
• Ato	 Semantic 	Overvier	Example:	
• Floa	 Represen 	Argume	'fcmp' Instruction	
• Fas	'tbaa.struc	 Semanti 	Syntax:	
	 'noalias' an 	 Example 'switch' In 	Overview:	
• Use	 'fpmath' Met 		 Arguments: 	
ο Sou	 'range' Meta 	Syntax:Overview	 Semantics: 	
Type S	'absolute_s'	- Overvier	Example:	

		The	
	 Arguments: 		
_	 Arguments: Semantics: 		
_	 Example: 		
_	 'catchswitch' Instruction 	• 'urem' Instruction	
	Syntax:	Syntax:	
	 Overview: 	Overview:	
_	 Arguments: 	 Arguments: 	0
_	 Semantics: 	 Semantics: 	
_	 Example: 	Example:	
	 'catchret' Instruction 	 'srem' Instruction 	
Metadata	Syntax:	 Syntax: 	
e' Metadata	 Overview: 	Overview:	
' Metadata	Arguments:	 Arguments: Semantics: 	
to' Instru	 Semantics: Example: 	Example:	
	 • 'cleanupret' Instruction 	• 'frem' Instruction	
W:	Syntax:	Syntax:	
ents:	 Overview: 	Overview:	
ics: e:	 Arguments: 	 Arguments: 	
to' Instru	 Semantics: 	 Semantics: 	
to mstru	Example:	Example:	
	 'unreachable' Instruction 	Bitwise Binary Operati	
w: ents:	Syntax:	 'shl' Instruction 	
ics:	 Overview: 	Syntax:	
e:	 Semantics: 	Overview:	
to' Instru	 Binary Operations 	 Arguments: 	
	 'add' Instruction 	 Semantics: 	
co mstru		- Examples	
	 Syntax: 	Example:	
w:	Overview:	• '1shr' Instruction	
w: ents: ics: e:	Overview:Arguments:	 '1shr' Instruction Syntax: 	
w: ents: ics:	Overview: Arguments:	• '1shr' Instruction	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w:	 Overview: Arguments: 	 'Ishr' Instruction Syntax: 	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents:	 Overview: Arguments: 	 'Ishr' Instruction Syntax: Refere Ole of Arguments: 	
w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics:	 Overview: Arguments: 	 'Ishr' Instruction Syntax: Refere Ole of Arguments: Semantics: 	
w: ents: ics: e: •• to'l w: ents: ics: e: •• to'l w: ents: ics: e: e:	 Overview: Arguments: 	 'Ishr' Instruction Syntax: Refere Ole of Arguments: 	
w: ents: ics: e: •• to'l w: ents: ics: e: •• to'l w: ents: ics: e: e:	 Overview: Arguments: Image: Arguments of the second se	 'Ishr' Instruction Syntax: Refere	
w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: to' li	 Overview: Arguments: 	 'Ishr' Instruction Syntax: Refere Ole of optimization Arguments: Semantics: Example: 'or' Instruction 	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents: ics: e: •• to' listri w: w:	 Overview: Arguments: 	 'Ishr' Instruction Syntax: Reference Reference Reference	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents: ics: e: •• to' lnstri w: ents: ics: e: •• to' lnstri	 Overview: Arguments: Example: 'fsub' Instruction Syntax: Overview: Arguments: Semantics: 	 'Ishr' Instruction Syntax: Reference Reference Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: 	•
w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' Instri	 Overview: Arguments: Example: 'fsub' Instruction Syntax: Overview: Arguments: Semantics: Example: 	 'Ishr' Instruction Syntax: Reference Reference Reference Reference Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: The semantics: Semantics: Example: 	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents: ics: e: •• to' lnstri w: ents: ics: e: •• to' lnstri	 Overview: Arguments: Example: 'fsub' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction 	 'Ishr' Instruction Syntax: Reference Reference Reference	
w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents: ics: e: •• to' listri w: ents: ics: e: •• to' listri w: ents: ics: e: •• to' li w: ents: ics: e: •• to' li w: ents: ics: •• to' li w: •• to' li w: •• to' li w: •• to' l	 Overview: Arguments: Image: Overview: Example: 'fsub' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: 	 'Ishr' Instruction Syntax: Reference Reference Reference Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: 'vor' Instruction Semantics: Example: 'xor' Instruction Syntax: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' Instri w: ents: ics: e: . to' li w: ents: ics: e: . to' li w: ents: e: . to' li w: ents: e: . to' li w: . to' li . to' li . to' li . to' li . to' . to' li . to' . to' li . to' . to' to' . to' . to' to' . to</pre>	 Overview: Arguments: Image: Overview: Example: 'fsub' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: 	 'Ishr' Instruction Syntax: Readed and the second syntax Readed and the second syntax Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' Instri w: ents: ics: e: . to' li w: ents: ics: e: . to' li ents: ics: e: . to' li ents: ics: . to' li ents: ics: . to' li ents: ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !</pre>	 Overview: Arguments: EXAMPLE: fsub'Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul'Instruction Syntax: Overview: Arguments: Semantics: Semantics: Example: 'mul'Instruction Syntax: Overview: Arguments: Semantics: Example: Marguments: Semantics: Example: 	 'Ishr' Instruction Syntax: Reference Reference Reference Reference Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri e: . to' instri . to' instri e: . to' instri . to' i</pre>	 Overview: Arguments: Interpret text of the second state of the second stat	 'Ishr' Instruction Syntax: Reference Reference Reference	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri e: . to' instri . to' instri e: . to' instri . to' instri e: . to' instri . t</pre>	 Overview: Arguments: Interpret terms of the second sec	 'Ishr' Instruction Syntax: Reference of Content of Syntax Arguments: Semantics: Semantics: Semantics: Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Syntax: Overview: Arguments: Semantics: Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Semantics: Semantics: Semantics: Semantics: Semantics: Semantics: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri e: . to' instri . to' instri e: . to' instri . to' i</pre>	 Overview: Arguments: EVALUATE: Arguments: Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: 	 'Ishr' Instruction Syntax: Reference Reference Reference	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: to' Instri w: ents: ics: e: to' instri e: to' instri to' instri to' instri e: to' instri to' instri e: to' instri to' i</pre>	 Overview: Arguments: Interpret Provided Hyperbolic Arguments: Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Semantics: Semantics: Arguments: Overview: Arguments: Semantics: Semantics: Semantics: Arguments: Semantics: Arguments: Arguments: Arguments: Arguments: Arguments: Arguments: Arguments: 	 'Ishr' Instruction Syntax: Readed and the second syntax Readed and the second syntax Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: Semantics: Example: Vector Operations 	
<pre>w: ents: ics: e: to' l w: ents: ics: e: to' l w: ents: ics: e: to' Instri w: ents: ics: e: to' w: ents: ics: e: to' w: ents: ics: e: to' w: ents: ics: e: to' w: ents: ics: e: to' w: ents: ics: e: to' w: e: to' to' </pre>	 Overview: Arguments: Interpret int	 'Ishr' Instruction Syntax: Redefectors Redefectors Redefectors	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri e: . to . to . to . to . to . to . to . to</pre>	 Overview: Arguments: Interpret Provided Hyperbolic Arguments: Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Semantics: Semantics: Arguments: Overview: Arguments: Semantics: Semantics: Semantics: Arguments: Semantics: Arguments: Arguments: Arguments: Arguments: Arguments: Arguments: Arguments: 	 'Ishr' Instruction Syntax: Readed and Control Syntax Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: Semantics: Example: 'xor' Instruction Syntax: Overview: Arguments: Semantics: Example: Vector Operations 'extractelement' In Syntax: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri w: ents: ics: e: . to' li w: ents: ics: e: . to' li w: ents: . to'</pre>	 Overview: Arguments: Interpret int	 'Ishr' Instruction Syntax: Reader of Content State Arguments: Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Semantic	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: to' linstri w: ents: ics: e: to' linstri e: to' to' </pre>	 Overview: Arguments: Interpret Provided Hyperbolic Arguments: Syntax: Overview: Arguments: Semantics: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: 	 'Ishr' Instruction Syntax: Readed and Control of Syntax : Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Syntax: Overview: Arguments: Semantics: 	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: . to' Instri w: ents: ics: e: . to' Instri w: ents: ics: e: . to' instri w: ents: ics: e: . to' li w: ents: ics: e: . to' li w: ents: . to'</pre>	 Overview: Arguments: Interpret int	 'Ishr' Instruction Syntax: Readea Control Contro Control Control Control Control Control Control Cont	
<pre>w: ents: ics: e: to' li w: ents: ics: e: to' li w: ents: ics: e: to' linstri w: ents: ics: e: to' linstri e: to' to' </pre>	 Overview: Arguments: Instruction Syntax: Overview: Arguments: Semantics: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'mul' Instruction Syntax: Overview: Arguments: Semantics: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Example: 'fmul' Instruction Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Syntax: Overview: Arguments: Syntax: Overview: Arguments: Semantics: Semantics: Semantics: Syntax: Overview: Arguments: Syntax: Overview: Youtiv' Instruction Syntax: Overview: 	 'Ishr' Instruction Syntax: Readed and Control of Syntax : Arguments: Semantics: Example: 'or' Instruction Syntax: Overview: Arguments: Semantics: Syntax: Overview: Arguments: Semantics: 	

Example:

Syntax:

'sdiv' Instruction

LLVM IR

слатріс.

Memory Access a

'alloca' Instru

Syntax:

Overview:

Arguments

Semantics:

Example:

'load' Instruct

Overview:

Arguments

Semantics:

Examples:

'store' Instruction

Syntax:

Overview:

Arguments

Semantics:

Example:

'fence' Instruct

Syntax:

Overview:

Arguments

Semantics:

Syntax:

Overview:

Arguments:

Semantics:

Example

Syntax:

 'fptosi to' ins 	struction
Syntax:	Intrinsic Funct
 Overview: 	 Variable Ar
 Argument 	• '11vm.va
 Semantics 	 Synta
Example:	 Over
'uitofp t	 Argu
Syntax:	 Sema
 Overview: 	 'llvm.v
Argument	Synta
Semantics	 Over
Example:	 Argu
<pre> 'sitofp t</pre>	 Argu Sema
Syntax:	- '11vm.v
 Overview: 	Synta
Argument	 Over
 Semantics 	Argu
Example:	 Argu Sema
<pre> 'ptrtoint</pre>	Accurate G
Syntax:	 Accurate d Experim
 Overview: 	 '11vm.g
Argument	
Semantics	SyntaOver
Example:	
<pre>inttoptr</pre>	 Argu Sema
Syntax:	- '11vm.g
 Overview: 	- Synta
	- Synta

tosi to' Inst	truction
Syntax:	Intrinsic Functi
Overview:	 Variable Arg
Argument	 'llvm.va
Semantics	 Synta
Example:	 Overv
tofp t	 Argur
Syntax:	 Semai
Overview:	• '11vm.va
Argument	 Synta
Semantics	 Overv
Example:	 Argur
tofp t	 Semai
Syntax:	• 'llvm.va
Overview:	 Synta
Argument	 Overv
Semantics	 Argur
Example:	 Sema
rtoint	• Accurate Ga
Syntax:	 Experime
Overview:	<pre>'llvm.gc</pre>
Argument	 Synta:
Semantics	 Overv
Example:	 Argur
ttoptr	 Semal
Syntax:	<pre> 'llvm.gc</pre>
Overview:	 Synta

ence Manual contents

0		- Argur	
Syntax:	 Semantics 	 Sema 	
Overview:	Example:	'llvm.ad	
 Arguments 	 Other Operatior 	 Synta: 	
Semantics:	 'icmp' Instruct 	 Overv 	
Example:	Syntax:	 Sema 	
 Vector of p 	 Overview: 	• 'llvm.fr	
Conversion Oper	 Argument 		
'trunc to'	 Semantics 	- Synta	0
Syntax:	Example:	over v	
 Overview: 	 'fcmp' Instruct 	Argur	
 Arguments 	 Syntax: 	 Sema 	
Semantics:	 Overview: 	• 'llvm.lo	
Example:		Intrinsics	
• 'zext to'	Argument Sementiar	 Synta: 	
Syntax:	 Semantics 	 Overv 	
Overview:	 Example: 	 Argur 	
 Arguments 	• 'phi' Instruct	 Semai 	
Semantics:	Syntax:	'llvm.re	
Example:	 Overview: 	Intrinsics	
• 'sext to'l	 Argument 	Synta:	
Syntax:	 Semantics 	 Overv 	
 Overview: 	 Example: 	 Semal 	0
 Arguments 	 'select' Inst 	• 'llvm.st	0
Semantics:	Syntax:	 Synta: 	0
Example:	 Overview: 	 Overv 	
 'fptrunc t 	Argument	 Semai 	

Argument

Sema

Sema

		LVM.
		trins
		Syn
	-	Ονε
	•	Arg
	• '1	lvm.
	In	trins
	•	Syn
		Ονε
	•	Arg
þ	Half	Preci
	• '1	lvm.
	-	Syn
		Ove
		Arg
		Sen
		Exa
	• '1	lvm.
		Syn
		Ονε
		Arg
		Sen
	•	Exa
þ	Debu	iggei
þ	Exce	
5	Tram	poli
	• '1	lvm.
		Syn

			Intrinsi
•	'llvm.readcy	clec	ounter' Intrinsic
	Syntax:		

Overview:

'llvm.experimental.vector.red

Intrinsic

Syn

Ove

Arg

Intrins

Syn

Ove

Arg

Intrins

Syn

Ove

Arg

Intrins

Syn

Ove

Arg

Intrins

Syn

Ove

Arg

ins

Ove

Ove

'11vm.

'llvm.

'llvm.

'llvm.

'llvm.

- Semantics:
- 'llvm.clear_cache' Intrinsic
- Syntax: Overview:
- Semantics:
- 'llvm.instrprof.increment' Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.instrprof.increment.step' Intr
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.instrprof.value.profile' Intri
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.thread.pointer' Intrinsic
- Syntax:
- Overview:
- Semantics:
- Standard C Library Intrinsics
- 'llvm.memcpy' Intrinsic
- Syntax:
- Overview: Arguments:
- Semantics:
- 'llvm.memmove' Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.memset.*' Intrinsics
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.sqrt.*' Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.powi.*' Intrinsic
- Svntax:
- Overview:
- Arguments:
- Semantics:
- 'llvm.sin.*' Intrinsic
- Syntax:
- Overview:
- Arguments: Semantics:
- 'llvm.cos.*' Intrinsic
- Syntax: Overview:
- Arguments:
- Semantic



- r.reduce.xor.*'
- 'llvm.load.relative' Intrinsic
- Syntax:

'llvm.experimental.vector.reduce.and.*'

Intrinsic

- Overview:
- 'llvm.sideeffect' Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- Stack Map Intrinsics
- Element Wise Atomic Memory Intrinsics 'llvm.memcpy.element.unordered.atomic'
 - Intrinsic
 - Syntax:
 - Overview:
 - Arguments:
 - Semantics:
- Lowering: 'llvm.memmove.element.unordered.atomic' Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- Lowering:
- 'llvm.memset.element.unordered.atomic'
- Intrinsic
- Syntax:
- Overview:
- Arguments:
- Semantics:
- Lowering:

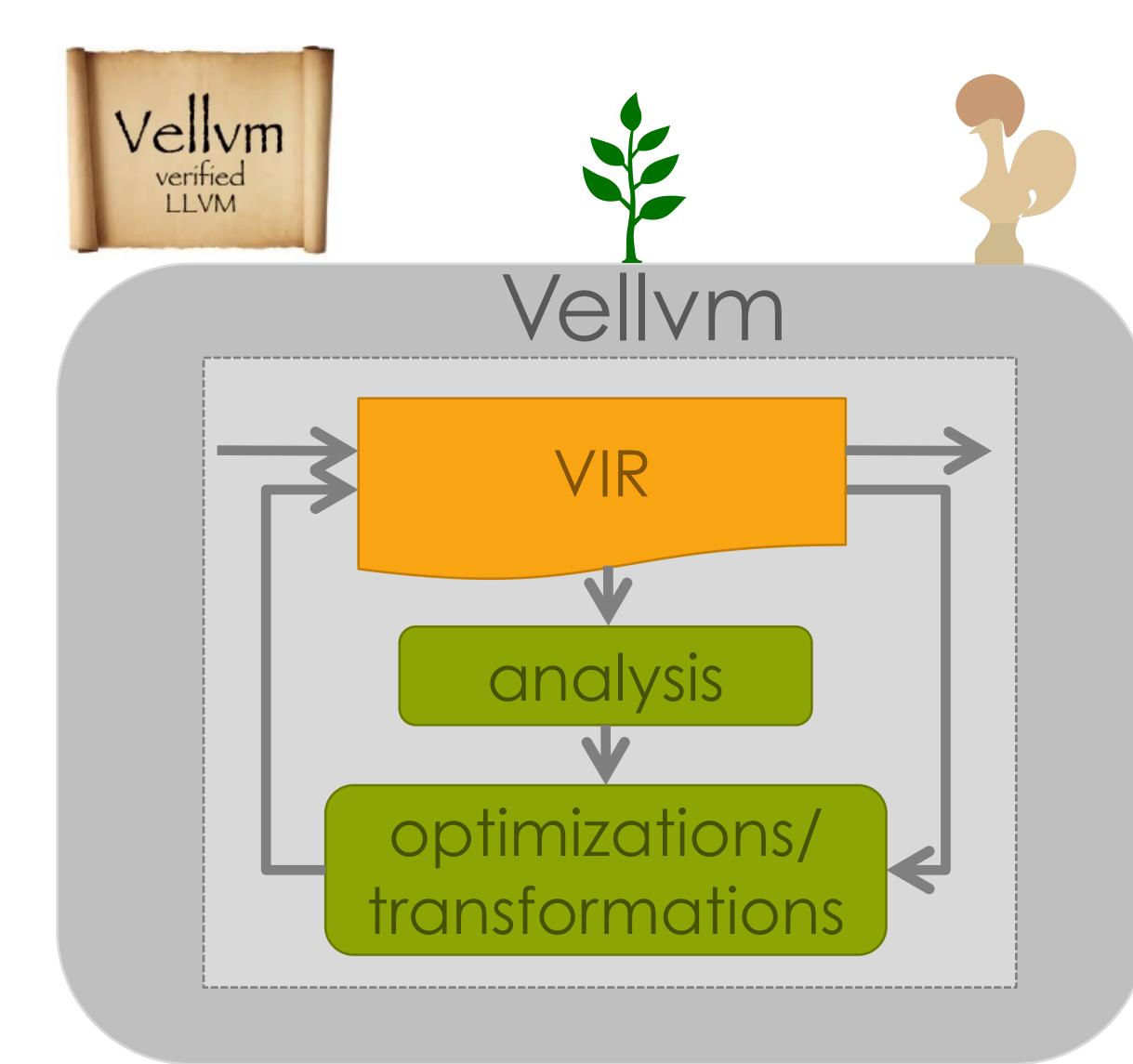
Abstract

This document is a reference manual for the LLVM assembly language. LLVM is a Static Single Assignment (SSA) based representation that provides type safety, low-level operations, flexibility, and the capability of representing 'all'

trinsic

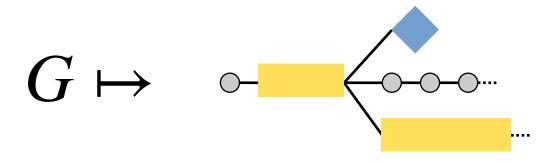


The Vellvm Project

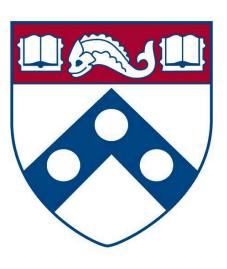


Operational style

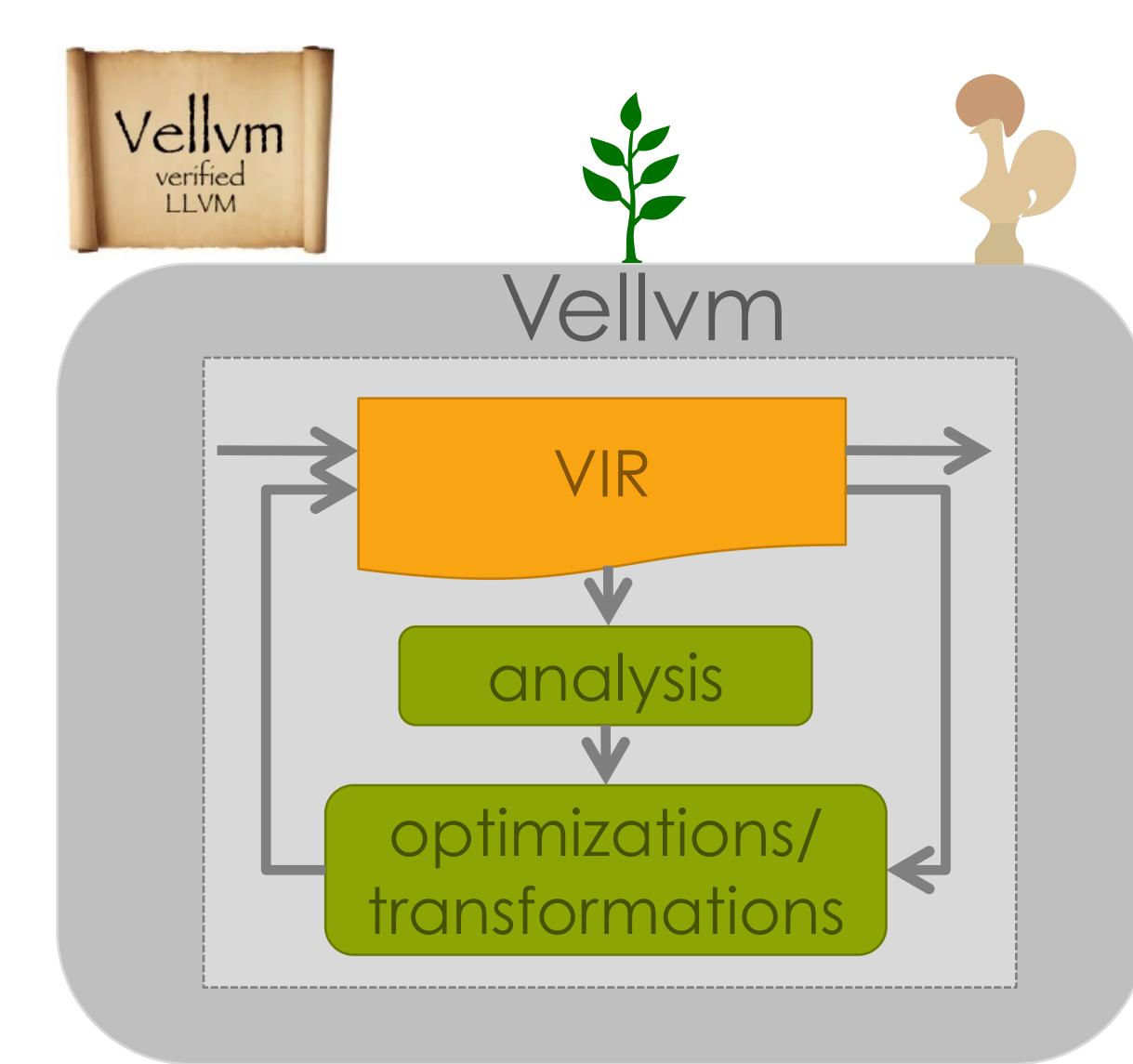
Monadic, denotational



Project in collaboration with the University of Pennsylvania

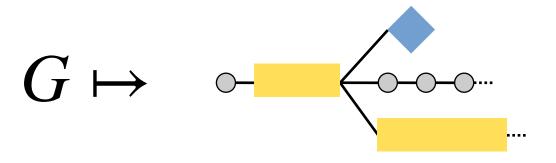


The Vellvm Project

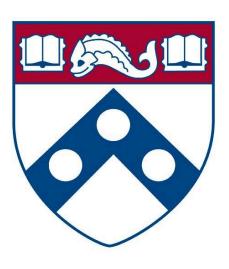


Operational style $G \vdash pc, mem \rightarrow pc', mem'$

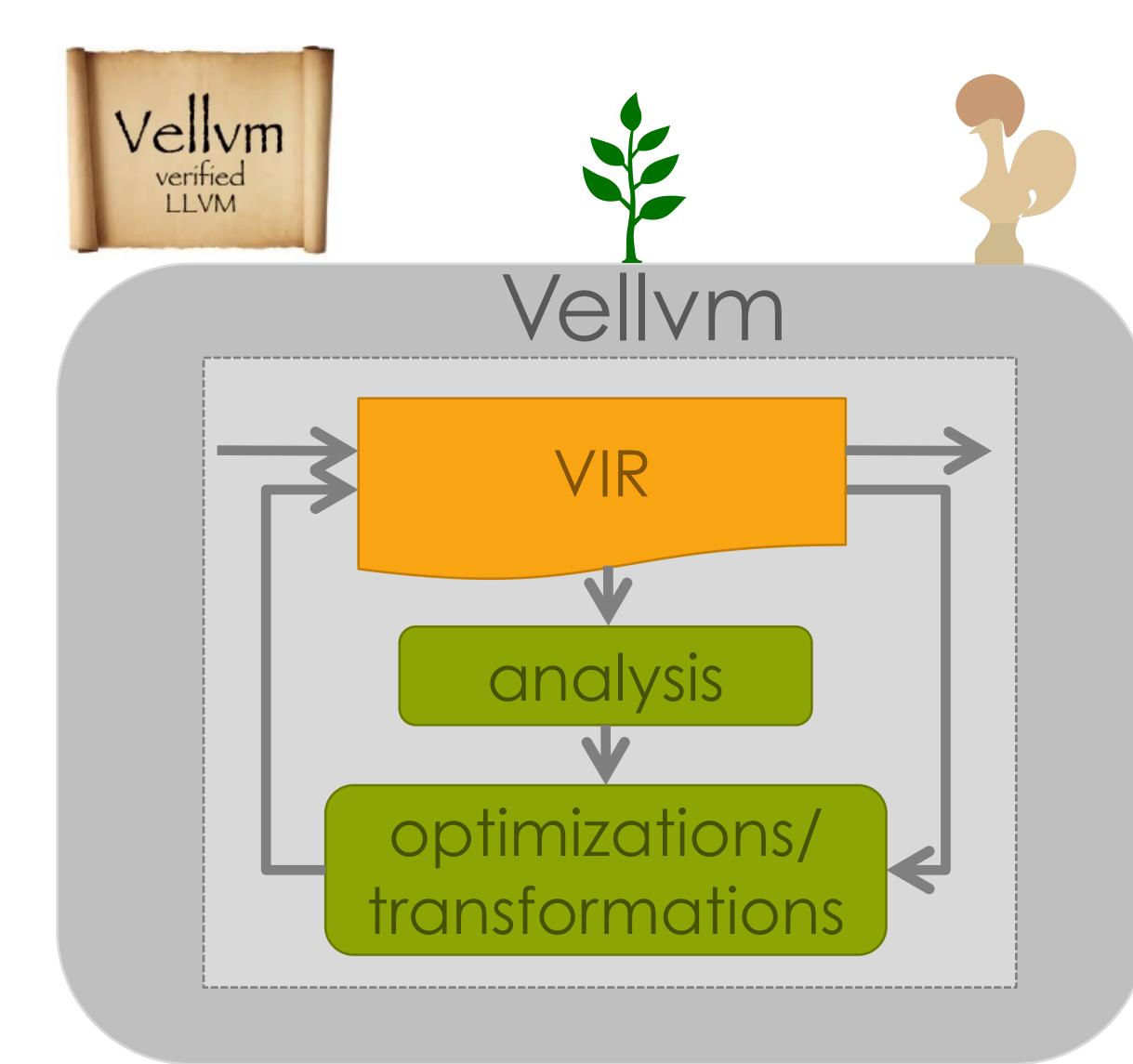
Monadic, denotational



Project in collaboration with the University of Pennsylvania

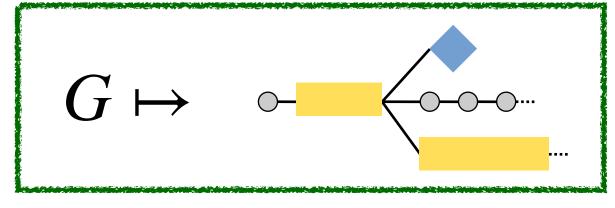


The Vellvm Project

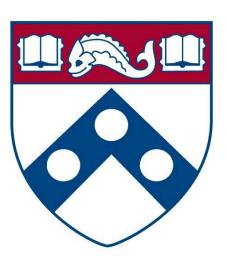


Operational style $G \vdash pc, mem \rightarrow pc', mem'$

Monadic, denotational



Project in collaboration with the University of Pennsylvania



(Operational) Semantics of an Imperative Language

Moving to the black board