THE NEXT 700 COMPILER CORRECTNESS THEOREMS (FUNCTIONAL PEARL)

Daniel Patterson & Amal Ahmed, Northeastern University
ICFP 2019

Priya Srikumar, Cornell University
Summer PLDG 2020

A history

1967 – Proving a compiler for arithmetic expressions correct

1973 – Statement of whole-program compiler correctness theorem

2006 – CompCert C Compiler

2011 – CompCert has no miscompilation errors; GCC/LLVM do

…and we're off!

What is compiler correctness?

$$s \rightsquigarrow t \quad \Longrightarrow \quad s \approx t$$

Semantics preserving

What is compiler correctness?

$$C_T^S(e_S)$$

Compiling from source to target

What is compiler correctness?

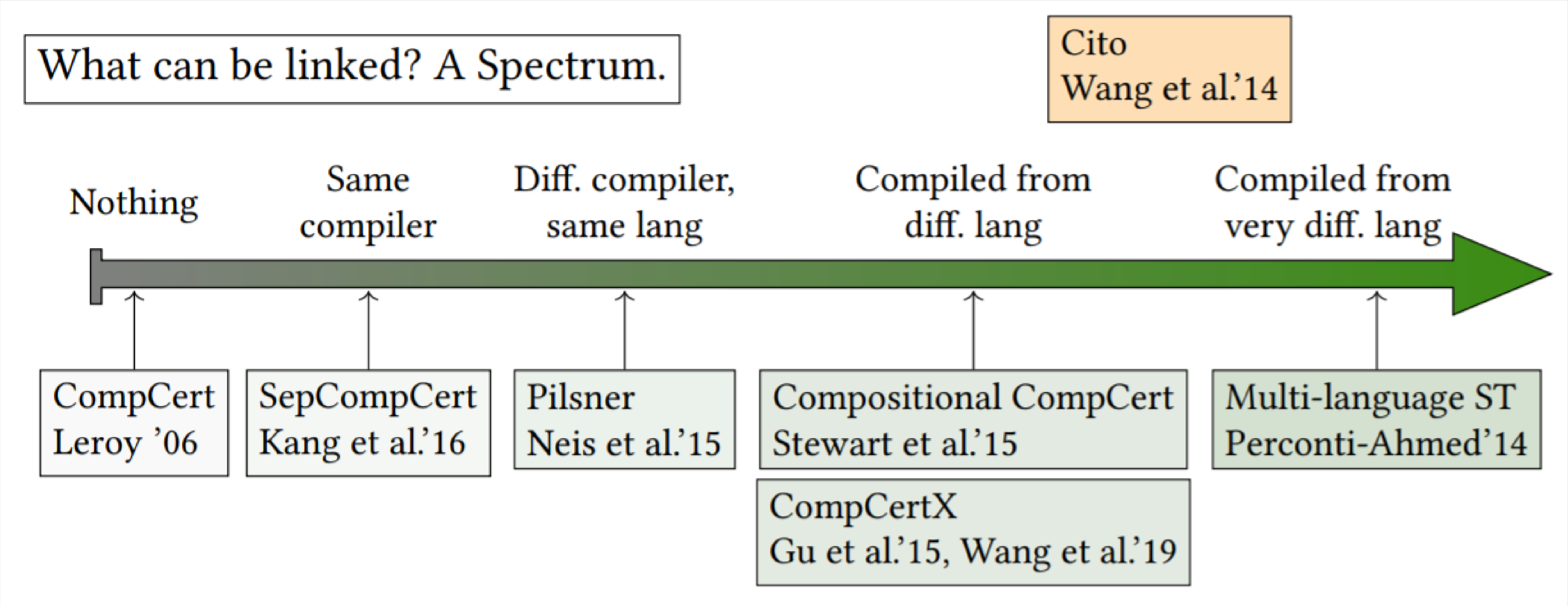
preserves the behavior of

refines

$T \sqsubseteq S$ defined for whole programs only

$$\forall e_S \in S.\ C_T^S(e_S)\ {}_T\sqsubseteq_S\ e_S$$

Whole program compiler correctness


What can be linked? A Spectrum.
Cito
Wang et al.'14
Nothing
Same compiler
Diff. compiler, same lang
Compiled from diff. lang
Compiled from very diff. lang
CompCert
Leroy '06
SepCompCert
Kang et al.'16
Pilsner
Neis et al.'15
Compositional CompCert
Stewart et al.'15
CompCertX
Gu et al.'15, Wang et al.'19
Multi-language ST
Perconti-Ahmed'14

What makes linking hard to verify?

What does relatedness look like?

What does linking look like?

How do we know a theorem is well-stated?

SepCompCert

CompCert compiler doesn't restrict to whole programs

But its correctness theorem does!

SepCompCert reveals bugs in CompCert

CCC theorems need to reflect actual compiler use

Pilsner

PILS (parametric inter-language simulation) relation

When are target modules related to source modules?

Provably transitive between stages of multi-pass compiler

Pilsner

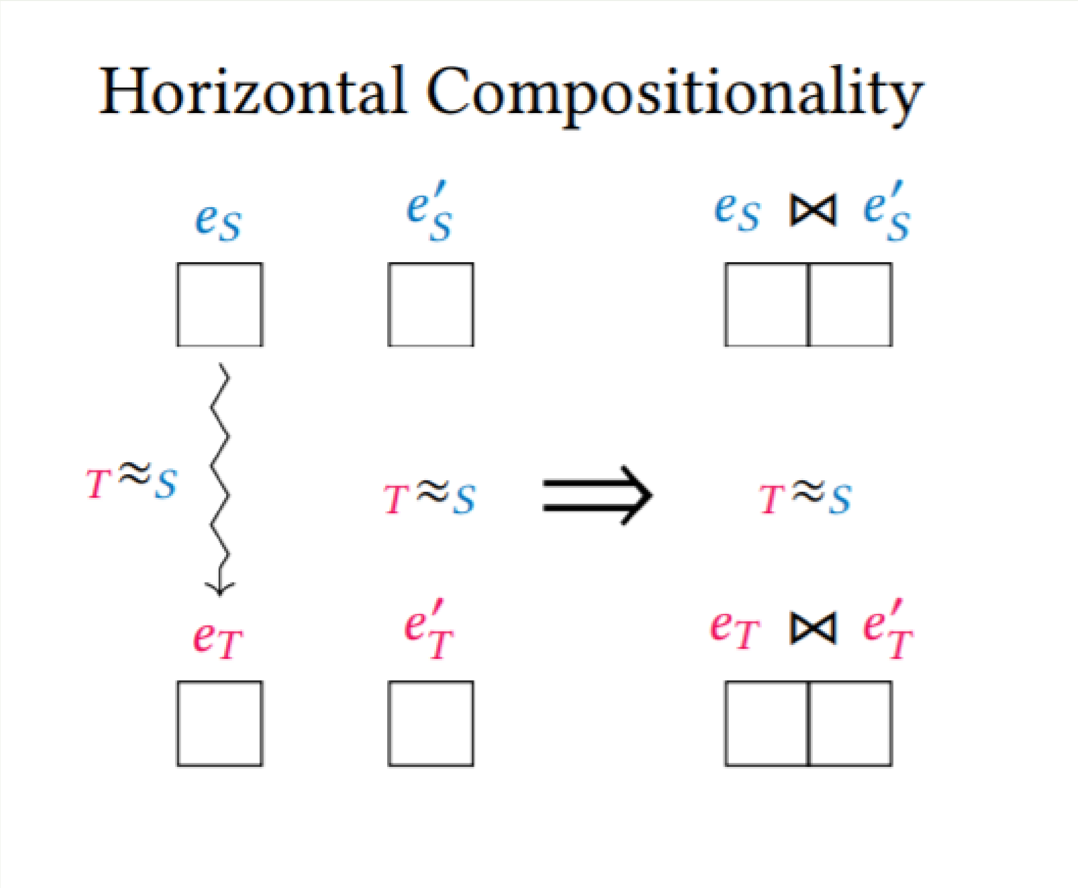
Need to find source module which is related to given target module

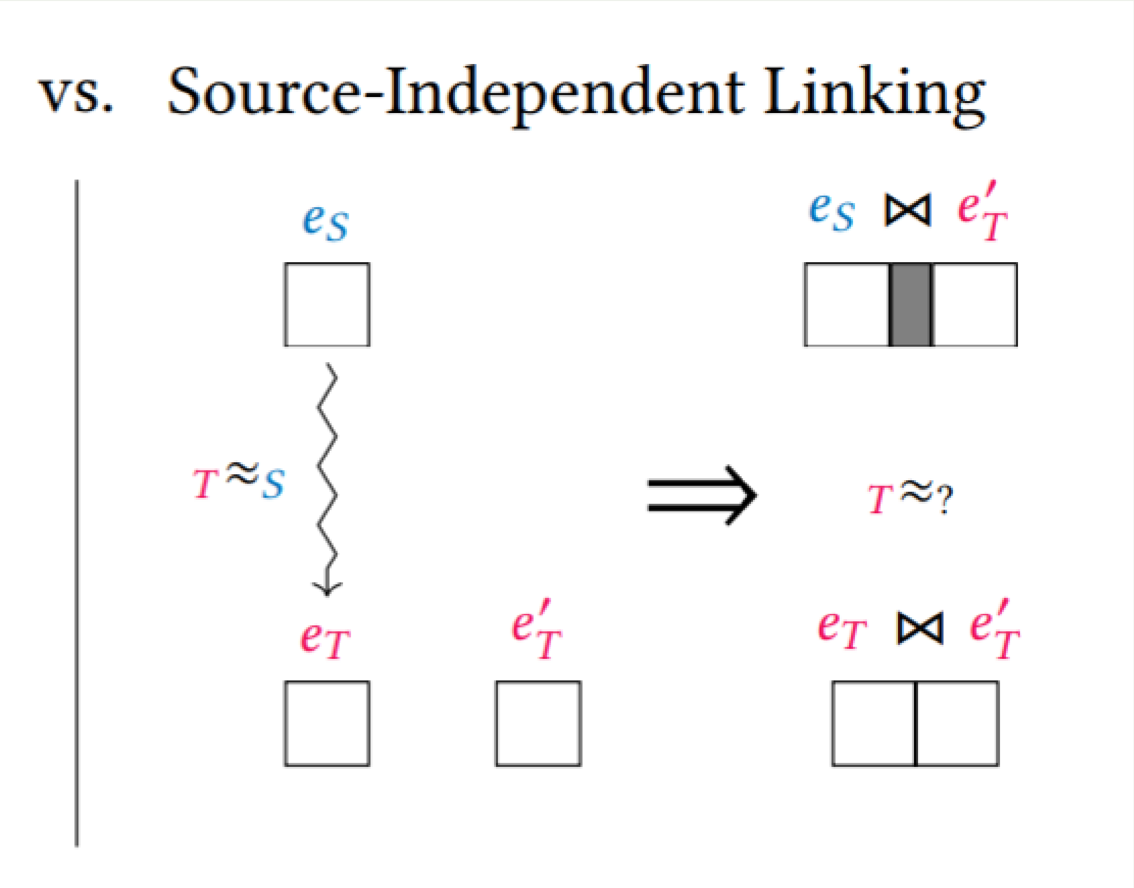
Hard to do! (hand-compilation?)

Limited to source language's representability

Generalizes over compilers with the same PILS relation

Not likely to have many of these





SepCompCert, Pilsner

Compositional CompCert

Interaction semantics for “language independent linking”

Allows linking with code in any CompCert language

Interaction protocol (requires same memory model)

Contextual equivalence in terms of interaction semantics

Expressed directly in Coq ($\widehat{S}$)

Semantic multi-language

Source-Target Multi-language

Syntactic multi-language

add ***boundary terms*** *which embed S terms in T contexts*

$$\mathcal{TS}(\cdot)$$

Compiler correctness then becomes

$$C_T^S(e_S) \approx \mathcal{TS}(e_S)$$

What makes a good CCC theorem?

Encompasses a variety of realistic compilers

Backwards-compatible with past (C)CC work

Straightforward to understand

What are the pieces of a CCC theorem?

What can we link with?

What language are we linking in?

How do we transform target code to this language?

How does linking operate?

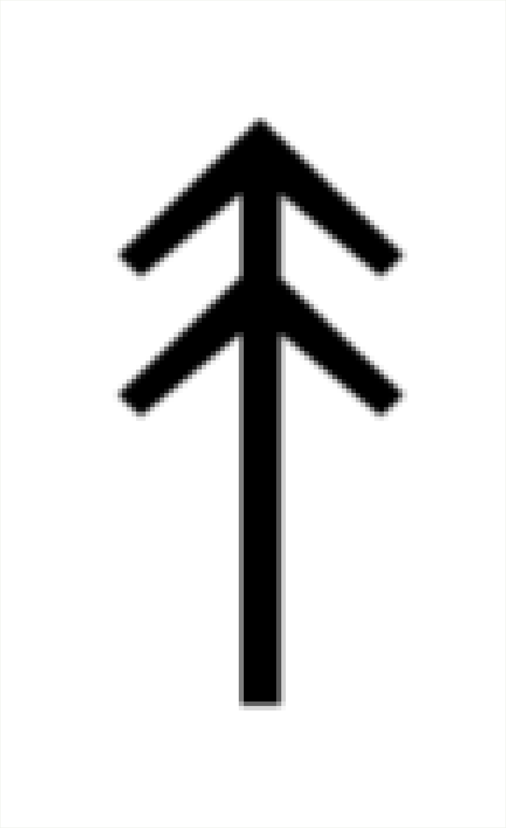
$\mathcal{L}$

Linking set

Elements are (e'_T, φ)
target component
and witness pairs

ST Linking Medium

language for linking
source component with
a component with behavior
equivalent to
target component



Lift function

$$\mathcal{L} \rightarrow \hat{S}$$

can use φ to generate

$$S \bowtie \widehat{S}$$

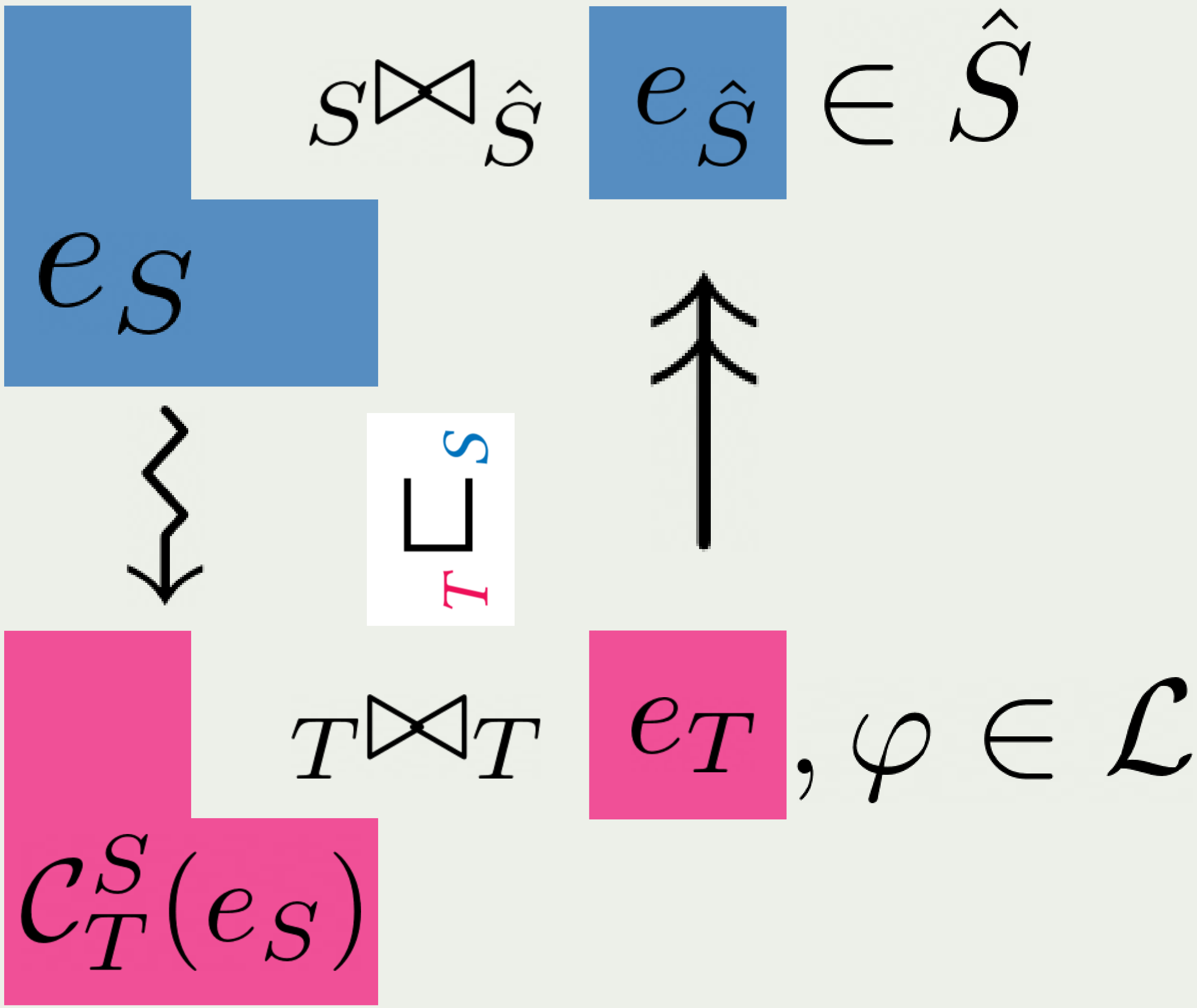
$$T \bowtie T$$

SepCompCert CCC Instantiation

$\mathcal{L}$	$\{(e_T, \varphi) \mid \varphi =$ source component e_S that was compiled by the SepCompCert compiler to $e_T\}$
$\widehat{S}$	unchanged source language S
$_{\widehat{S}}\bowtie_{S}$	unchanged source language linking $_S\bowtie_S$
$_{\widehat{S}}\sqsubset_{S}$	source language (whole program) refinement $_S\sqsubset_S$
$\Uparrow(\cdot)$	$\Uparrow(e_T, e_S) = e_S$

GOAL:

Define partial program refinement in terms of $T \sqsubset S$


$S \bowtie \hat{S}$
$e_{\hat{S}} \in \hat{S}$
e_S
$T \sqsubset S$
$T \bowtie T$
$e_T, \varphi \in \mathcal{L}$
$c_T^S(e_S)$

$$\exists \Uparrow.\ \forall e_S \in S.\ \forall (e_T, \varphi) \in \mathcal{L}.\ e_T\ {}_T\!\bowtie_T C_T^S(e_S)\ {}_T\!\sqsubseteq_{\widehat{S}}\ \Uparrow(e_T, \varphi)\ {}_{\widehat{S}}\!\bowtie_S e_S$$

Compositional compiler correctness

$$\exists \Uparrow.\ \forall e_S \in S.\ \forall (e_T, \varphi) \in \mathcal{L}.\ e_T \ {}_T\!\bowtie_T C_T^S(e_S) \ {}_T\!\sqsubset_{\widehat{S}} \ \Uparrow(e_T, \varphi) \ {}_{\widehat{S}}\!\bowtie_S e_S$$

There exists a lift function,

for any source component

and any linkable target component, s.t.

linking the target component

with a compiled source component

refines*

lifting the target component

and linking it with that source component.

$$\exists \Uparrow.\ \forall e_S \in S.\ \forall (e_T, \varphi) \in \mathcal{L}.\ e_T \;{}_T\!\bowtie_T\; C_T^S(e_S) \;{}_T\!\sqsubseteq_{\widehat{S}}\; \Uparrow(e_T, \varphi) \;{}_{\widehat{S}}\!\bowtie_S\; e_S$$

refines*: linking is a partial function that can fail

If language linking validation doesn't fail,
and we get a whole program,
we get *semantics preservation*
from the source component
linked with the lifted target component
to the compiled source component
linked with the target component.

$$(\varnothing_T, \varphi_\varnothing) \in \mathcal{L}$$

The empty component can be linked.

$$\forall e_S.\ \exists \varphi.\ (C_T^S(e_S), \varphi) \in \mathcal{L}$$

Anything the compiler outputs can be linked.

$$\Uparrow(\varnothing_T, \varphi_\varnothing) = \varnothing_{\hat{S}}$$

The lift function ensures that the
empty component in the target language
is lifted to the
empty component in the ST linking medium.

$$\forall e_S.\ \emptyset_{\hat{S}}\ \hat{S} \bowtie_S e_S\ \ \hat{S} \sqsubseteq_S\ e_S$$

Linking a program with the empty component preserves the program's semantics.

Lifting is the inverse of compiling.

$$\forall (e_T, \varphi) \in \mathcal{L}.\, \forall e_S.\, (\forall c_T.\, c_T \;{}_T{\bowtie}_T\; e_T \;{}_T{\sqsubset}_T\; c_T \;{}_T{\bowtie}_T\; C_T^S(e_S)) \implies$$

$$(\forall c_S.\, c_S \;{}_S{\bowtie}_{\hat{S}}\; \Uparrow(e_T, \varphi) \;{}_{\hat{S}}{\sqsubset}_S\; c_S \;{}_S{\bowtie}_S\; e_S)$$

If a target component refines
a compiled source component,
then the lift of the target component should refine
the source component.

SepCompCert CCC Instantiation

$\mathcal{L}$	$\{(e_T, \varphi) \mid \varphi =$ source component e_S that was compiled by the SepCompCert compiler to $e_T\}$
$\widehat{S}$	unchanged source language S
$_{\widehat{S}}\bowtie_S$	unchanged source language linking $_S\bowtie_S$
$_{\widehat{S}}\sqsubseteq_S$	source language (whole program) refinement $_S\sqsubseteq_S$
$\Uparrow(\cdot)$	$\Uparrow(e_T, e_S) = e_S$

SepCompCert correctness ⟶ CCC

$$\forall e_S \in S.\ \forall (C^S_T(e'_S), e'_S) \in \mathcal{L}.\ C^S_T(e'_S)\ {}_T{\bowtie}_T\ C^S_T(e_S)\ \ {}_T{\sqsubseteq}_S\ \ e'_S\ {}_S{\bowtie}_S\ e_S$$

$$\begin{aligned}&\forall (C^S_T(e'_S), e'_S) \in \mathcal{L}.\ \forall e_S.\ (\forall c_T.\ c_T\ {}_T{\bowtie}_T\ C^S_T(e'_S)\ \ {}_T{\sqsubseteq}_T\ \ c_T\ {}_T{\bowtie}_T\ C^S_T(e_S)) \implies \\ &\qquad (\forall c_S.\ c_S\ {}_S{\bowtie}_S\ {\Uparrow}(C^S_T(e'_S), e'_S)\ \ {}_S{\sqsubseteq}_S\ \ c_S\ {}_S{\bowtie}_S\ e_S)\end{aligned}$$

What is needed to understand CCC?

There's an upper bound on how much formalism we can take

Explicit parameters

Good for users!

Good for researchers!