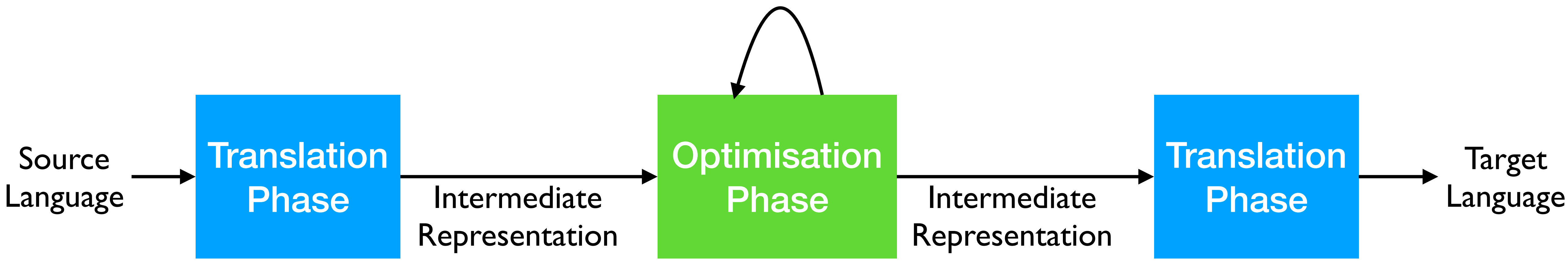
Compiler Correctness

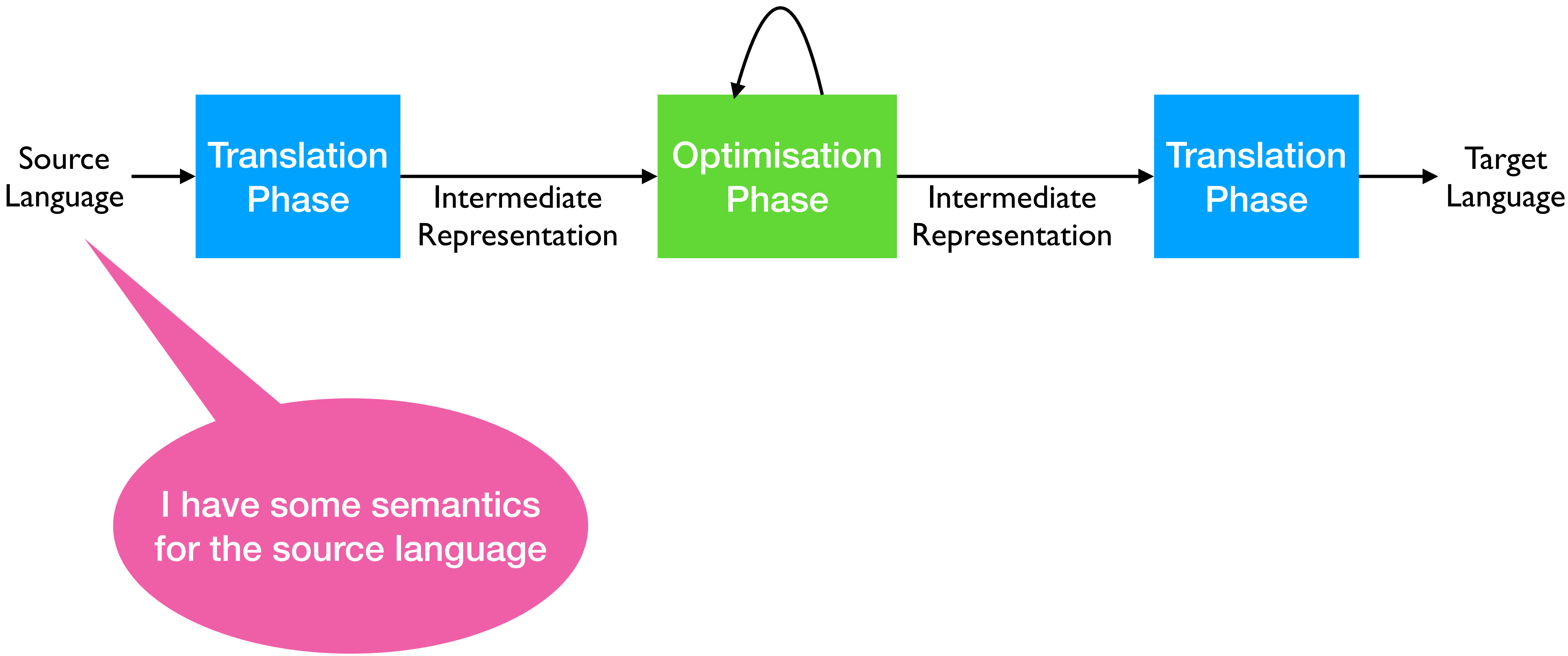
KC Sivaramakrishnan

Spring 2021

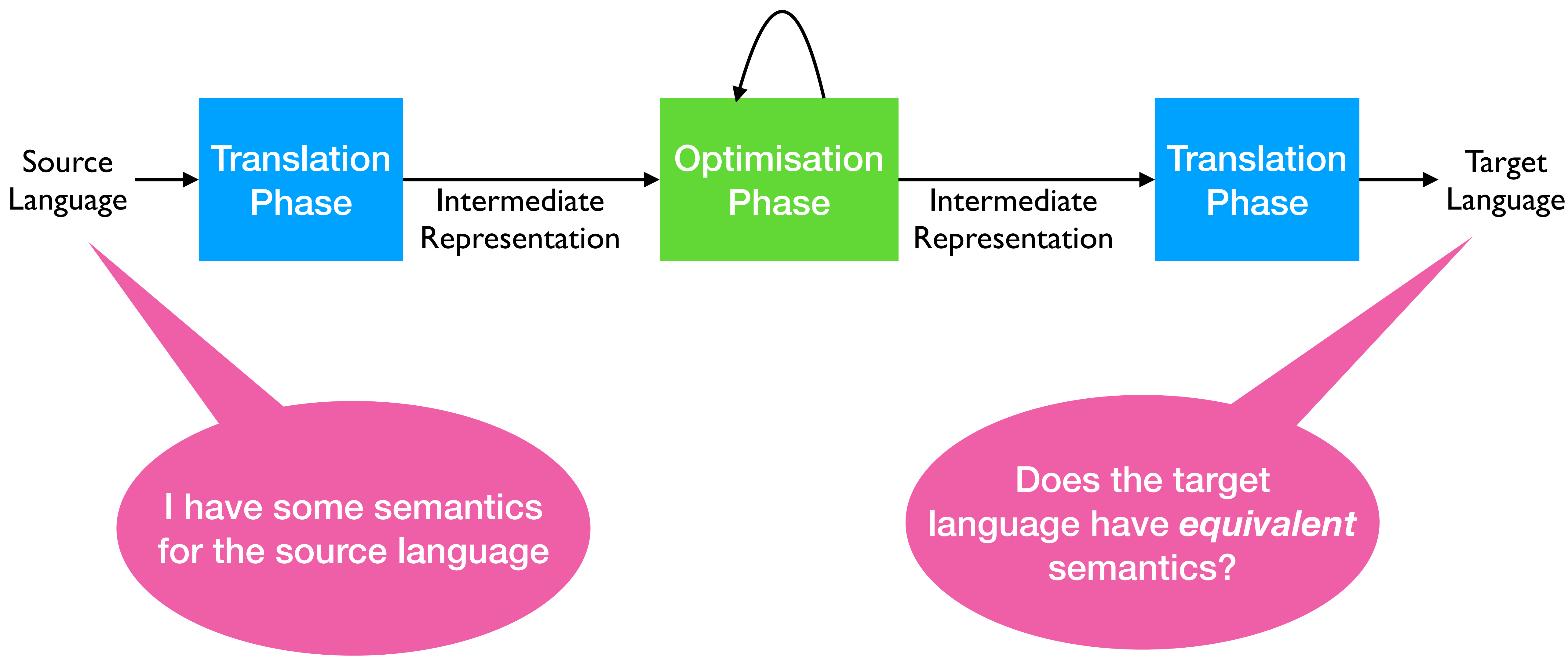
Why prove compilers correct?



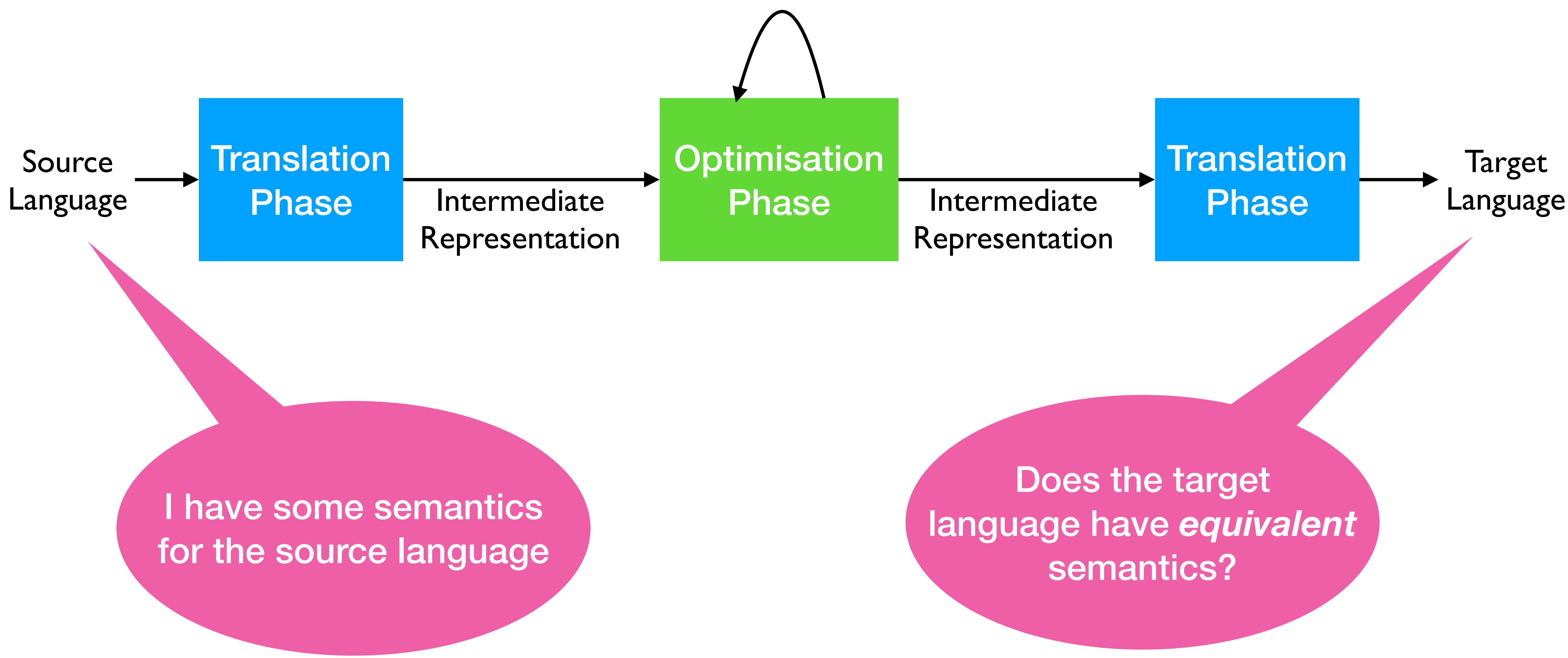
Why prove compilers correct?



Why prove compilers correct?



Why prove compilers correct?



Idea: Prove each step of the translation and optimisation correct through *Simulation*

Language

$$
\begin{array}{rccl}
\text{Numbers} & n & \in & \mathbb{N} \\
\text{Variables} & x & \in & \mathsf{Strings} \\
\text{Expressions} & e & ::= & n \mid x \mid e + e \mid e - e \mid e \times e \\
\text{Commands} & c & ::= & \mathsf{skip} \mid x \leftarrow e \mid c; c \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c \mid \mathsf{while}\ e\ \mathsf{do}\ c \mid \underline{\mathsf{out}(e)}
\end{array}
$$

Language

$$
\begin{array}{rccl}
\text{Numbers} & n & \in & \mathbb{N} \\
\text{Variables} & x & \in & \mathsf{Strings} \\
\text{Expressions} & e & ::= & n \mid x \mid e + e \mid e - e \mid e \times e \\
\text{Commands} & c & ::= & \mathsf{skip} \mid x \leftarrow e \mid c; c \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c \mid \mathsf{while}\ e\ \mathsf{do}\ c \mid \underline{\mathsf{out}(e)}
\end{array}
$$

- An optimising compiler should preserve the behaviour of the program in terms of the output *traces* generated.

Language

$$
\begin{array}{rccl}
\text{Numbers} & n & \in & \mathbb{N} \\
\text{Variables} & x & \in & \mathsf{Strings} \\
\text{Expressions} & e & ::= & n \mid x \mid e + e \mid e - e \mid e \times e \\
\text{Commands} & c & ::= & \mathsf{skip} \mid x \leftarrow e \mid c; c \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c \mid \mathsf{while}\ e\ \mathsf{do}\ c \mid \underline{\mathsf{out}(e)}
\end{array}
$$

- An optimising compiler should preserve the behaviour of the program in terms of the output *traces* generated.
- Equivalence of traces is fundamental correctness property
  - ✦ Invariants — safety
  - ✦ Trace equivalence — liveness (not only for terminating programs)

Labelled transition semantics

$$\frac{}{(v, \mathsf{out}(e)) \xrightarrow{[\![e]\!]v}_0 (v, \mathsf{skip})}$$

$$\frac{}{(v, x \leftarrow e) \xrightarrow{\epsilon}_0 (v[x \mapsto [\![e]\!]v], \mathsf{skip})} \qquad \frac{}{(v, \mathsf{skip}; c_2) \xrightarrow{\epsilon}_0 (v, c_2)}$$

$$\frac{[\![e]\!]v \neq 0}{(v, \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \xrightarrow{\epsilon}_0 (v, c_1)} \qquad \frac{[\![e]\!]v = 0}{(v, \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \xrightarrow{\epsilon}_0 (v, c_2)}$$

$$\frac{[\![e]\!]v \neq 0}{(v, \mathsf{while}\ e\ \mathsf{do}\ c_1) \xrightarrow{\epsilon}_0 (v, c_1; \mathsf{while}\ e\ \mathsf{do}\ c_1)} \qquad \frac{[\![e]\!]v = 0}{(v, \mathsf{while}\ e\ \mathsf{do}\ c_1) \xrightarrow{\epsilon}_0 (v, \mathsf{skip})}$$

$$\frac{(v, c) \xrightarrow{\ell}_0 (v', c')}{(v, C[c]) \xrightarrow{\ell}_{\mathsf{c}} (v', C[c'])}$$

Traces

- Finite sequences of outputs and termination events

$$\frac{}{\cdot \in \mathsf{Tr}(s)} \quad \frac{}{\mathsf{terminate} \in \mathsf{Tr}((v, \mathsf{skip}))} \quad \frac{s \xrightarrow{\epsilon}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{t \in \mathsf{Tr}(s)} \quad \frac{s \xrightarrow{n}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{\mathsf{out}(n) \bowtie t \in \mathsf{Tr}(s)}$$

- A trace is allowed to end even if the program hasn't terminated

Traces

- Finite sequences of outputs and termination events

$$\frac{}{\cdot \in \mathsf{Tr}(s)} \qquad \frac{}{\mathsf{terminate} \in \mathsf{Tr}((v, \mathsf{skip}))} \qquad \frac{s \xrightarrow{\epsilon}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{t \in \mathsf{Tr}(s)} \qquad \frac{s \xrightarrow{n}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{\mathsf{out}(n) \bowtie t \in \mathsf{Tr}(s)}$$

- A trace is allowed to end even if the program hasn't terminated

DEFINITION 9.1 (Trace inclusion). For commands c_1 and c_2, let $c_1 \preceq c_2$ iff $\mathsf{Tr}(c_1) \subseteq \mathsf{Tr}(c_2)$.

DEFINITION 9.2 (Trace equivalence). For commands c_1 and c_2, let $c_1 \simeq c_2$ iff $\mathsf{Tr}(c_1) = \mathsf{Tr}(c_2)$.

Traces

- Finite sequences of outputs and termination events

$$\frac{}{\cdot \in \mathsf{Tr}(s)} \qquad \frac{}{\mathsf{terminate} \in \mathsf{Tr}((v, \mathsf{skip}))} \qquad \frac{s \xrightarrow{\epsilon}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{t \in \mathsf{Tr}(s)} \qquad \frac{s \xrightarrow{n}_{\mathsf{c}} s' \quad t \in \mathsf{Tr}(s')}{\mathsf{out}(n) \bowtie t \in \mathsf{Tr}(s)}$$

- A trace is allowed to end even if the program hasn't terminated

DEFINITION 9.1 (Trace inclusion). For commands c_1 and c_2, let $c_1 \preceq c_2$ iff $\mathsf{Tr}(c_1) \subseteq \mathsf{Tr}(c_2)$.

DEFINITION 9.2 (Trace equivalence). For commands c_1 and c_2, let $c_1 \simeq c_2$ iff $\mathsf{Tr}(c_1) = \mathsf{Tr}(c_2)$.

$$\begin{aligned} & c_1 \simeq c2 \\ \iff & \mathrm{Tr}(c_1) = \mathrm{Tr}(c_2) \\ \iff & \mathrm{Tr}(c_1) \subseteq \mathrm{Tr}(c_2) \wedge \mathrm{Tr}(c_2) \subseteq \mathrm{Tr}(c_1) \\ \iff & c_1 \preceq c_2 \wedge c_2 \preceq c_1 \end{aligned}$$

Constant-folding

- Optimisation is
  - Find all maximal program subexpressions that don’t contain variables
  - Replace each subexpression with its known constant value
- The optimisation only changes variable free-expressions
  - The original and the optimised program match at each *step*

Basic Simulation Relation

Definition 9.3 (Simulation relation). We say that binary relation R over states of our object language is a *simulation relation* iff:

(1) Whenever $(v_1, \mathsf{skip})\ R\ (v_2, c_2)$, it follows that $c_2 = \mathsf{skip}$.

(2) Whenever $s_1\ R\ s_2$ and $s_1 \xrightarrow{\ell}_{\mathsf{c}} s_1'$, there exists s_2' such that $s_2 \xrightarrow{\ell}_{\mathsf{c}} s_2'$ and $s_1'\ R\ s_2'$.

$$\begin{tikzcd}
s_1 \arrow[r, "R"] \arrow[d, "\forall \xrightarrow{\ell}_{\mathsf{c}}"] & s_2 \arrow[d, "\exists \xrightarrow{\ell}_{\mathsf{c}}"] \\
s_1' & s_2' \arrow[l, "R^{-1}"]
\end{tikzcd}$$

Theorem 9.4. *If there exists a simulation R such that $s_1\ R\ s_2$, then $s_1 \simeq s_2$.*

Simulation for Constant Folding

THEOREM 9.5. *For any v and c,* $(v, c) \simeq (v, \mathsf{cfold}_1(c))$.

PROOF. By a simulation argument using this relation:

$$(v_1, c_1)\ R\ (v_2, c_2) \quad = \quad v_1 = v_2 \wedge c_2 = \mathsf{cfold}_1(c_1)$$