Lecture 2: Proving Full Abstraction (+ Question)

CS350

Marco Patrignani

Some Answers: CEQ with Randomisation

- Assume the language has `rand`

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- $$\frac{\mathtt{n} \in \mathbb{N}}{\mathtt{rand} \to \mathtt{n}}\ \text{(Rand)}$$

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- $$\frac{n \in \mathbb{N}}{\texttt{rand} \to n}\ (\text{Rand})$$

```
public Int random(){return rand;}  // P1
```

```
public Int random(){rand; return rand;}  // P2
```

```
public Int random(){return < rand;,rand;> }  // P3
```

```
public Int random(){x=rand; return < x,x> ;}  // P4
```

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- $$\frac{n \in \mathbb{N}}{\texttt{rand} \to n}\ \text{(Rand)}$$

```
public Int random(){return rand;}  // P1
```

```
public Int random(){rand; return rand;}  // P2
```

```
public Int random(){return < rand;,rand;> }  // P3
```

```
public Int random(){x=rand; return < x,x> ;}  // P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- (Rand)
  $$\frac{n \in \mathbb{N}}{\texttt{rand} \to n}$$

```
publi
```

```
publi
```

Q: are $P1$ and $P2$ equivalent?

```
public Int random(){return < rand;,rand;> }  // P3
```

```
public Int random(){x=rand; return < x,x> ;}  // P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- (Rand)
  $$\frac{n \in \mathbb{N}}{\mathtt{rand} \to n}$$

```
publi
```

```
publi
```

Q: are $P1$ and $P2$ equivalent?

Should they be?

```
public Int random(){return < rand;,rand;> }  // P3
```

```
public Int random(){x=rand; return < x,x> ;}  // P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has `rand`
- $$\frac{n \in \mathbb{N}}{O; n \triangleright \text{rand} \rightarrow O \triangleright \text{n}}\ (\text{Rand})$$
- Oracles: infinite lists of random numbers

```
public Int random(){return rand;}  // P1
```

```
public Int random(){rand; return rand;}  // P2
```

```
public Int random(){return < rand;,rand;> }  // P3
```

```
public Int random(){x=rand; return < x,x> ;}  // P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has `rand`

CEQ:

$$P_1 \simeq_{ctx} P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}. \mathfrak{C}[P_1]\downarrow \iff \mathfrak{C}[P_2]\downarrow$$

```
publi
```

```
publi
```

```
publi                                                            3
```

```
publi                                                           P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has rand

-

-

1 publi

1 publi

1 publi

1 publi

CEQ:

$$P_1 \simeq_{ctx} P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}. \mathfrak{C}[P_1]\downarrow \iff \mathfrak{C}[P_2]\downarrow$$

CEQ-with-rand, try 1:

$$P_1 \simeq_{ctx} ? P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O. O \triangleright \mathfrak{C}[P_1]\downarrow \iff O \triangleright \mathfrak{C}[P_2]\downarrow$$

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has rand

CEQ:

$$P_1 \simeq_{ctx} P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}. \mathfrak{C}[P_1]\downarrow \iff \mathfrak{C}[P_2]\downarrow$$

CEQ-with-rand, try 1:

$$P_1 \simeq_{ctx}? P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O. O \rhd \mathfrak{C}[P_1]\downarrow \iff O \rhd \mathfrak{C}[P_2]\downarrow$$

No!

P_1 and P_2 are not equivalent with this definition (but they should be)

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has rand
-
-

```
publi
publi
publi                                                       3
publi                                                       P4
```

Contextual Preorder:
(not an eq, not symmetric)

$$P_1 \sqsubseteq P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O_1 . O_1 \triangleright \mathfrak{C}[P_1]\downarrow \Rightarrow \exists O_2 . O_2 \triangleright \mathfrak{C}[P_2]\downarrow$$

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

Contextual Preorder:

(not an eq, not symmetric)

$$P_1 \sqsubseteq P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O_1. O_1 \rhd \mathfrak{C}[P_1]\downarrow \Rightarrow \exists O_2. O_2 \rhd \mathfrak{C}[P_2]\downarrow$$

For $P_1 \simeq_{ctx} P_2$, O_2 is O_1 with every other element interleaved with random numbers

- A…
-
- (…

```
publi…
```

```
publi…
```

```
publi…  …3
```

```
publi…  …P4
```

Intui…

Some Answers: CEQ with Randomisation

- Assume the language has `rand`

Contextual Preorder:
(not an eq, not symmetric)

$$P_1 \sqsubseteq P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O_1 . O_1 \rhd \mathfrak{C}[P_1]\downarrow \Rightarrow \exists O_2 . O_2 \rhd \mathfrak{C}[P_2]\downarrow$$

Must also include $P_2 \sqsubseteq P_1$ otherwise P_3 and P_4 are also equivalent (and they should not be)

```
publi
```

```
publi
```

```
publi                                      3
```

```
publi                                      P4
```

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Some Answers: CEQ with Randomisation

- Assume the language has rand

1 publi

1 publi

1 publi ... 3

1 publi ... P4

Contextual Preorder:

(not an eq, not symmetric)

$$P_1 \sqsubseteq P_2 \stackrel{\text{def}}{=} \forall \mathfrak{C}, \forall O_1 . O_1 \triangleright \mathfrak{C}[P_1]\downarrow \Rightarrow \exists O_2 . O_2 \triangleright \mathfrak{C}[P_2]\downarrow$$

Must also include $P_2 \sqsubseteq P_1$ otherwise P_3 and P_4 are also equivalent (and they should not be)

$$P_1 \simeq_{ctx} P_2 \stackrel{\text{def}}{=} P_1 \sqsubseteq P_2 \cap P_2 \sqsubseteq P_1$$

Intuitively $P_1 \simeq_{ctx} P_2$ and $P_3 \not\simeq_{ctx} P_3$

Other Equivalences

- Contextual equivalence is not the only notion of program equivalence

Other Equivalences

- Contextual equivalence is not the only notion of program equivalence
- Any semantics defines its notion of equivalence

Other Equivalences

- Contextual equivalence is not the only notion of program equivalence
- Any semantics defines its notion of equivalence
- Any notion of equivalence can be used in the statement of fully abstract compilation

Other Equivalences

- Contextual equivalence is not the only notion of program equivalence
- Any semantics defines its notion of equivalence
- Any notion of equivalence can be used in the statement of fully abstract compilation
- Trace semantics or bisimilarity are widely used

Fully Abstract Compilation

$$[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} \text{ is FAC} \overset{\text{def}}{=} \forall \mathsf{P}_1, \mathsf{P}_2$$

$$\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$$

Fully Abstract Compilation

$\llbracket \cdot \rrbracket^{\mathsf{S}}_{\mathbf{T}}$ is FAC $\overset{\text{def}}{=} \forall \mathsf{P}_1, \mathsf{P}_2$

$$\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff \llbracket \mathsf{P}_1 \rrbracket^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} \llbracket \mathsf{P}_2 \rrbracket^{\mathsf{S}}_{\mathbf{T}}$$

- break the $\iff$:
  1. $\Rightarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \Rightarrow \llbracket \mathsf{P}_1 \rrbracket^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} \llbracket \mathsf{P}_2 \rrbracket^{\mathsf{S}}_{\mathbf{T}}$
  2. $\Leftarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ \llbracket \mathsf{P}_1 \rrbracket^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} \llbracket \mathsf{P}_2 \rrbracket^{\mathsf{S}}_{\mathbf{T}} \Rightarrow \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2$
- point 2 (should) follow from compiler correctness

Fully Abstract Compilation

$$[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} \text{ is FAC} \stackrel{\text{def}}{=} \forall \mathsf{P}_1, \mathsf{P}_2$$

$$\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$$

- break the $\iff$:
  1. $\Rightarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \Rightarrow [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$
  2. $\Leftarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}} \Rightarrow \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2$
- point 2 (should) follow from compiler correctness
- point 1 is tricky, because of $\simeq_{ctx}$ and its $\forall \mathfrak{C}$

Fully Abstract Compilation

$[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FAC $\stackrel{\text{def}}{=} \forall \mathsf{P}_1, \mathsf{P}_2$

$$\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$$

- break the $\iff$:
  1. $\Rightarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \Rightarrow [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$
  2. $\Leftarrow$: $\forall \mathsf{P}_1, \mathsf{P}_2.\ [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}} \Rightarrow \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2$
- point 2 (should) follow from compiler correctness
- point 1 is tricky, because of $\simeq_{ctx}$ and its $\forall \mathfrak{C}$
  This structure is called a backtranslation

Trace Semantics

- we replace $\simeq_{ctx}$ with something equivalent

Trace Semantics

- we replace $\simeq_{ctx}$ with something equivalent
- but simpler to reason about

Trace Semantics

- we replace $\simeq_{ctx}$ with something equivalent
- but simpler to reason about
- a semantics that abstracts from the context (observer)

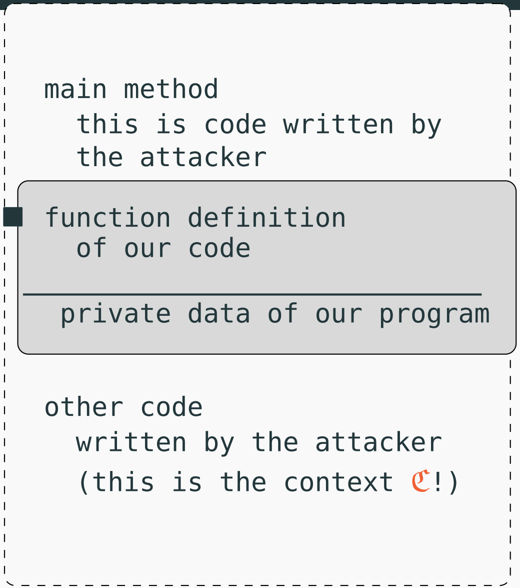
Trace Semantics

- we replace $\simeq_{ctx}$ with something equivalent
- but simpler to reason about
- a semantics that abstracts from the context (observer)
- and still describes the behaviour of a program precisely

Trace Semantics

- we replace $\simeq_{ctx}$ with something equivalent
- but simpler to reason about
- a semantics that abstracts from the context (observer)
- and still describes the behaviour of a program precisely
- a trace semantics

Traces for PMA

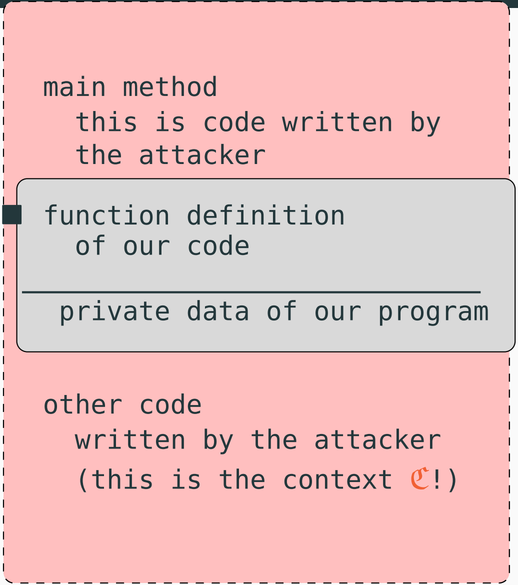


- interest in the behaviour of our code (component)

Traces for PMA

- interest in the behaviour of our code (component)
- need to consider the *rest*

Traces for PMA



- interest in the behaviour of our code (component)
- need to consider the *rest*

Trace Semantics for Our Program

```
main method
  this is code written by
  the attacker

function definition
  of our code
_______________________________
 private data of our program

other code
  written by the attacker
  (this is the context ℭ!)
```

- disregard the rest

Trace Semantics for Our Program

- disregard the rest

Trace Semantics for Our Program

- disregard the rest
- abstract its behaviour from the component perspective:

Trace Semantics for Our Program

- disregard the rest
- abstract its behaviour from the component perspective:
    1. jump to an entry point ■

Trace Semantics for Our Program

- disregard the rest
- abstract its behaviour from the component perspective:
  1. jump to an entry point ■
- abstract the component behaviour from the rest perspective:

Trace Semantics for Our Program

- disregard the rest
- abstract its behaviour from the component perspective:
  1. jump to an entry point ■
- abstract the component behaviour from the rest perspective:
  1. call/return

Trace Semantics

- semantics for partial programs (component)

Trace Semantics

- semantics for partial programs (component)
- relies on the operational semantics

Trace Semantics

- semantics for partial programs (component)
- relies on the operational semantics
- denotational: describes the behaviour of a component as sets of traces

Trace Semantics

- semantics for partial programs (component)
- relies on the operational semantics
- denotational: describes the behaviour of a component as sets of traces
- a trace is (typically) a sequence of actions that describe how a component interacts with an observer

Trace Semantics

- semantics for partial programs (component)
- relies on the operational semantics
- denotational: describes the behaviour of a component as sets of traces
- a trace is (typically) a sequence of actions that describe how a component interacts with an observer
- without needing to specify the observer

Trace Semantics

- semantics for partial programs (component)
- relies on the operational semantics
- denotational: describes the behaviour of a component as sets of traces
- a trace is (typically) a sequence of actions that describe how a component interacts with an observer
- without needing to specify the observer
- indicated as $\mathrm{TR}(C) = \left\{ \overline{\alpha} \;\middle|\; C \overset{\overline{\alpha}}{\Longrightarrow} \_ \right\}$

Trace Actions

$$\begin{aligned} \textit{Labels} \quad & L ::= a \mid \epsilon \\ \textit{Observable actions} \quad & \alpha ::= \surd \mid g? \mid g! \\ \textit{Actions} \quad & g ::= \texttt{call}\ p\ (r) \mid \texttt{ret}\ p\ r(\texttt{r}_0) \end{aligned}$$

Traces for PMA

We need to define:

- trace states (almost program states)
- labels that make traces
- rules for generating labels and traces $\cdots$
- the traces of a component $\mathsf{TR}(C) = \cdots$

Trace Equivalence

- all semantics yield a notion of equivalence

Trace Equivalence

- all semantics yield a notion of equivalence
- the operational semantics gives us contextual equivalence

$$C_1 \simeq_{ctx} C_2$$

Trace Equivalence

- all semantics yield a notion of equivalence
- the operational semantics gives us contextual equivalence

$$C_1 \simeq_{ctx} C_2$$

- trace semantics gives us trace equivalence

$$C_1 \stackrel{\mathrm{T}}{=} C_2$$

Trace Equivalence

- all semantics yield a notion of equivalence
- the operational semantics gives us contextual equivalence

$$C_1 \simeq_{ctx} C_2$$

- trace semantics gives us trace equivalence

$$\mathsf{TR}(C_1) = \mathsf{TR}(C_2)$$

the traces of C_1 are the same of those of C_2

Trace Equivalence

- all semantics yield a notion of equivalence
- the operational semantics gives us contextual equivalence

$$C_1 \simeq_{ctx} C_2$$

- trace semantics gives us trace equivalence

$$\left\{\overline{\alpha} \;\middle|\; C_1 \stackrel{\overline{\alpha}}{\Longrightarrow} \_\right\} = \left\{\overline{\alpha} \;\middle|\; C_2 \stackrel{\overline{\alpha}}{\Longrightarrow} \_\right\}$$

the traces of C_1 are the same of those of C_2

Proofs about Trace Semantics

- any trace semantics won’t just work
- it needs to be
  correct and complete

Proofs about Trace Semantics

- any trace semantics won't just work
- it needs to be
  correct and complete

$$C_1 \simeq_{ctx} C_2 \iff C_1 \overset{\mathrm{T}}{=} C_2$$

Proofs about Trace Semantics

- any trace semantics won't just work
- it needs to be
  correct ($\Leftarrow$) and complete ($\Rightarrow$)

$$C_1 \simeq_{ctx} C_2 \iff C_1 \overset{\mathrm{T}}{=} C_2$$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \Rightarrow [\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}}$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \Rightarrow \forall \mathbf{C}.\ \mathbf{C}\left[[\![\mathsf{C}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \right]\!\downarrow \iff \mathbf{C}\left[[\![\mathsf{C}_2]\!]^{\mathsf{S}}_{\mathbf{T}} \right]\!\downarrow$
- unfold $\simeq_{ctx}$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\exists \mathbf{C}.\ \mathbf{C}\left[[\![\mathsf{C}_1]\!]^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \not\iff \mathbf{C}\left[[\![\mathsf{C}_2]\!]^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \Rightarrow \mathsf{P}_1 \not\simeq_{ctx} \mathsf{P}_2$
- unfold $\simeq_{ctx}$
- contrapositive

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\exists \mathbf{C}.\ \mathbf{C}\left[\llbracket \mathsf{C_1} \rrbracket^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \not\Longleftrightarrow \mathbf{C}\left[\llbracket \mathsf{C_2} \rrbracket^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \Rightarrow$ $\exists \mathsf{C}.\mathsf{C}\left[\mathsf{C_2}\right]\downarrow \not\Longleftrightarrow \mathsf{C}\left[\mathsf{C_2}\right]\downarrow$
- unfold $\simeq_{ctx}$
- contrapositive
- unfold $\simeq_{ctx}$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\exists \mathbf{C}.\ \mathbf{C}\left[[\![\mathsf{C_1}]\!]^{\mathsf{S}}_{\mathbf{T}} \right]\downarrow \not\Longleftrightarrow \mathbf{C}\left[[\![\mathsf{C_2}]\!]^{\mathsf{S}}_{\mathbf{T}} \right]\downarrow \Rightarrow$
    $\exists \mathsf{C}.\mathsf{C}\left[\mathsf{C_2}\right]\downarrow \not\Longleftrightarrow \mathsf{C}\left[\mathsf{C_2}\right]\downarrow$
- unfold $\simeq_{ctx}$
- contrapositive
- unfold $\simeq_{ctx}$
- backtranslation!

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\exists \mathbf{C}.\ \mathbf{C}\left[\llbracket \mathsf{C_1} \rrbracket^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \not\Leftrightarrow \mathbf{C}\left[\llbracket \mathsf{C_2} \rrbracket^{\mathsf{S}}_{\mathbf{T}}\right]\downarrow \Rightarrow$
    $\exists \mathsf{C}.\mathsf{C}\left[\mathsf{C_2}\right]\downarrow \not\Leftrightarrow \mathsf{C}\left[\mathsf{C_2}\right]\downarrow$
- generate C based on $\mathbf{C}$

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $[\![\mathsf{P}_1]\!]^{\mathsf{S}}_{\mathbf{T}} \not\simeq_{ctx} [\![\mathsf{P}_2]\!]^{\mathsf{S}}_{\mathbf{T}} \Rightarrow \exists \mathsf{C}. \mathsf{C}[\mathsf{C}_2]\downarrow \not\Longleftrightarrow \mathsf{C}[\mathsf{C}_2]\downarrow$
- generate C based on $\mathbf{C}$
- if complex, apply Traces (folding $\simeq_{ctx}$)

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $[\![\mathsf{P_1}]\!]^{\mathsf{S}}_{\mathbf{T}} \not\simeq^{\mathsf{T}} [\![\mathsf{P_2}]\!]^{\mathsf{S}}_{\mathbf{T}} \Rightarrow \exists \mathsf{C}.\mathsf{C}[\mathsf{C_2}]\downarrow \not\iff \mathsf{C}[\mathsf{C_2}]\downarrow$
- generate C based on $\mathbf{C}$
- if complex, apply Traces (folding $\simeq_{ctx}$)

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\mathsf{TR}(\mathbf{C_1}) \neq \mathsf{TR}(\mathbf{C_2}) \Rightarrow \exists \mathsf{C}.\mathsf{C}[\mathsf{C_2}]\downarrow \not\Longleftrightarrow \mathsf{C}[\mathsf{C_2}]\downarrow$
- generate C based on $\mathbf{C}$
- if complex, apply Traces (folding $\simeq_{ctx}$)

Fully Abstract Compilation & Target Traces

- we have:
  - $\mathbf{C_1} \simeq_{ctx} \mathbf{C_2} \iff \mathsf{TR}(\mathbf{C_1}) = \mathsf{TR}(\mathbf{C_2})$
- we need to prove
  - $\exists \alpha \in \mathsf{TR}(\mathbf{C_1}), \alpha \notin \mathsf{TR}(\mathbf{C_2}) \Rightarrow$
    $\exists \mathsf{C}.\mathsf{C}[\mathsf{C_2}]\downarrow \not\Longleftrightarrow \mathsf{C}[\mathsf{C_2}]\downarrow$
- generate C based on $\mathbf{C}$
- if complex, apply Traces (folding $\simeq_{ctx}$)