Lecture 4: Translating the Meaning of Safety

CS350

Marco Patrignani

Example (FAC and safety preservation)

- S has booleans

Example (FAC and safety preservation)

- S has booleans
- $\mathbb{T}$ has naturals

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true
- coding $\mathsf{true} \approx \mathbf{1}$, $\mathsf{false} \approx \mathbf{0}$

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true
- coding $\mathsf{true} \approx \mathbf{1}$, $\mathsf{false} \approx \mathbf{0}$
- Compiler $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ compiles C into $\boldsymbol{\lambda}\mathbf{x}.\,\mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{0}$

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true
- coding true $\approx \mathbf{1}$, false $\approx \mathbf{0}$
- Compiler $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ compiles C into $\lambda\mathbf{x}.\,\mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{0}$
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FA

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true
- coding true $\approx \mathbf{1}$, false $\approx \mathbf{0}$
- Compiler $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ compiles C into $\lambda\mathbf{x}.\,\mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{0}$
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FA
- Safety property in S: never output false

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true
- coding true $\approx \mathbf{1}$, false $\approx \mathbf{0}$
- Compiler $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ compiles C into $\lambda\mathbf{x}.\,\mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{0}$
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FA
- Safety property in S: never output false
- Safety property in $\mathbf{T}$: never output $\mathbf{0}$

Example (FAC and safety preservation)

- S has booleans
- $\mathbf{T}$ has naturals
- Any C always return true

A FA compiler does not preserve simple safety
under an intuitive translation

- Safety property in S: never output false
- Safety property in $\mathbf{T}$: never output $\mathbf{0}$

Example (FAC and safety preservation)

- S has booleans
- T has naturals
- Any C always return true

A FA compiler does not preserve simple safety
under an intuitive translation
because of responses to invalid input

- Safety property in S: never output false
- Safety property in T: never output 0

Example (FAC and declassification)

- Same S and $\mathbf{T}$

Example (FAC and declassification)

- Same S and $\mathbf{T}$
- C has a secret. For 9 interactions it behaves like $\lambda x.\, x$, then it returns the secret

Example (FAC and declassification)

- Same S and $\mathbf{T}$
- C has a secret. For 9 interactions it behaves like $\lambda \mathsf{x}.\, \mathsf{x}$, then it returns the secret
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ takes any C and returns $\lambda \mathbf{x}.\, \mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else\ leak\ secret}$

Example (FAC and declassification)

- Same S and $\mathbf{T}$
- C has a secret. For 9 interactions it behaves like $\lambda \mathsf{x}.\, \mathsf{x}$, then it returns the secret
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ takes any C and returns $\lambda \mathbf{x}.\, \mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{leak}\ \mathbf{secret}$
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FA

Example (FAC and declassification)

- Same S and $\mathbf{T}$
- C has a secret. For 9 interactions it behaves like $\lambda x.\, x$, then it returns the secret
- $[\![\cdot]\!]^{S}_{\mathbf{T}}$ takes any C and returns $\boldsymbol{\lambda x.\,\textbf{if } x < 2 \textbf{ then } [\![C]\!]\ x \textbf{ else leak secret}}$
- $[\![\cdot]\!]^{S}_{\mathbf{T}}$ is FA
- Property in S (and $\mathbf{T}$): Do not output the secret until the 10th input

Example (FAC and declassification)

- Same S and $\mathbf{T}$
- C has a secret. For 9 interactions it behaves like $\lambda x.\, x$, then it returns the secret
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ takes any C and returns $\lambda \mathbf{x}.\, \mathbf{if}\ \mathbf{x} < \mathbf{2}\ \mathbf{then}\ [\![\mathsf{C}]\!]\ \mathbf{x}\ \mathbf{else}\ \mathbf{leak}\ \mathbf{secret}$
- $[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}}$ is FA
- Property in S (and $\mathbf{T}$): Do not output the secret until the 10th input
- timing is key to failure here

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbf{T}$

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbf{T}$
- Often, this is not the case, so $\mathtt{t}$ and $\mathbf{t}$ can have different forms

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbf{T}$
- Often, this is not the case, so t and $\mathbf{t}$ can have different forms
  e.g., t deals with high-level values and $\mathbf{t}$ with low-level addresses

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbf{T}$
- Often, this is not the case, so t and $\mathbf{t}$ can have different forms
  e.g., t deals with high-level values and $\mathbf{t}$ with low-level addresses
- Assume the alphabet of $\mathbf{t}$ is larger than the one for t
  (otherwise no correct compilation is possible)

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbf{T}$
- Often, this is not the case, so t and $\mathbf{t}$ can have different forms
  e.g., t deals with high-level values and $\mathbf{t}$ with low-level addresses
- Assume the alphabet of $\mathbf{t}$ is larger than the one for t
  (otherwise no correct compilation is possible)
  call the set extra target actions $\eta = \{\eta_1 \cdots \eta_n\}$

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbb{T}$
- Often, this is not the case, so t and t can have different forms

Q1: how do we bridge this gap?

- (otherwise no correct compilation is possible)

  call the set extra target actions $\eta = \{\eta_1 \cdots \eta_n\}$

Different Trace Alphabets

- So far traces had the same alphabet between S and T
- Often, this is not the case, so t and t can have different forms

e

w

Q1: how do we bridge this gap?

Q2: how do we preserve the meaning of properties across languages?

- A … ne

one for t

(otherwise no correct compilation is possible)

call the set extra target actions $\eta = \{\eta_1 \cdots \eta_n\}$

Different Trace Alphabets

- So far traces had the same alphabet between S and $\mathbb{T}$
- Often, this is not the case, so t and t can have different forms

Recommended reading:

Patrignani and Garg. 2017. Secure Compilation as Hyperproperty Preservation.

(otherwise no correct compilation is possible)

call the set extra target actions $\eta = \{\eta_1 \cdots \eta_n\}$

Cross-language Relation

- Source property π, target property π

Cross-language Relation

- Source property π, target property π
- We indicate that they capture the same property as $\pi \approx \pi$

Cross-language Relation

- Source property π, target property π
- We indicate that they capture the same property as $\pi \approx \pi$
- What does this mean?

Cross-language Relation

- Source property π, target property π
- We indicate that they capture the same property as $\pi \approx \pi$
- What does this mean?

- We only know for safety and hypersafety

Safety Preservation with TPC

- safety = traces where something bad does not happen

Safety Preservation with TPC

- safety = traces where something bad does not happen

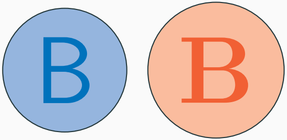
dually

Safety Preservation with TPC

- safety = traces where something bad does not happen

dually

- safety = set of bad prefixes



Safety, Visually

Safety, Visually

B

$\gamma_1?\gamma_2!$

$\beta_1?\beta_2!\beta_3?\beta_4!\beta_5?\beta_6!$

$\alpha_1?\alpha_2!\alpha_3?\alpha_4!$

Safety, Visually

B

$\gamma_1?\gamma_2!$

$\beta_1?\beta_2!\beta_3?\beta_4!\beta_5?\beta_6!$

$\alpha_1?\alpha_2!\alpha_3?\alpha_4!$

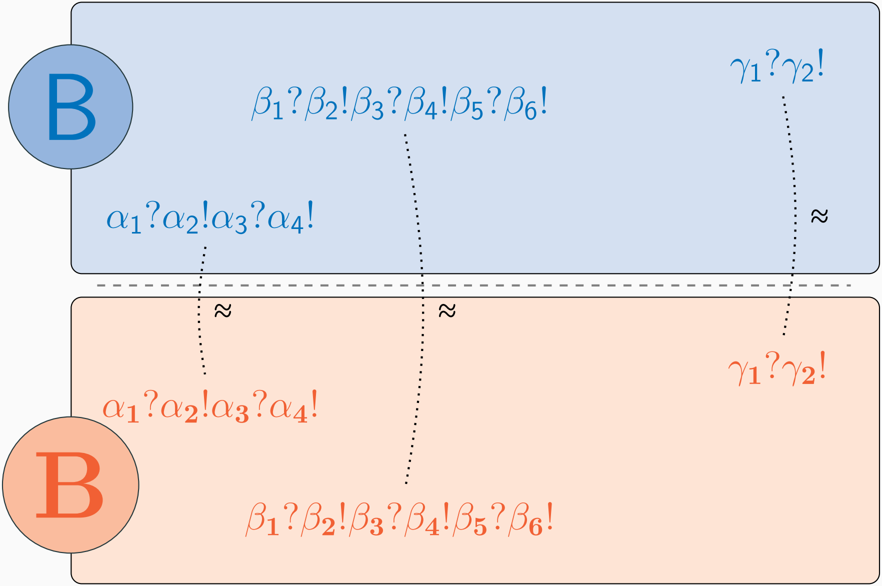
$\gamma_{\mathbf{1}}?\gamma_{\mathbf{2}}!$

$\alpha_{\mathbf{1}}?\alpha_{\mathbf{2}}!\alpha_{\mathbf{3}}?\alpha_{\mathbf{4}}!$

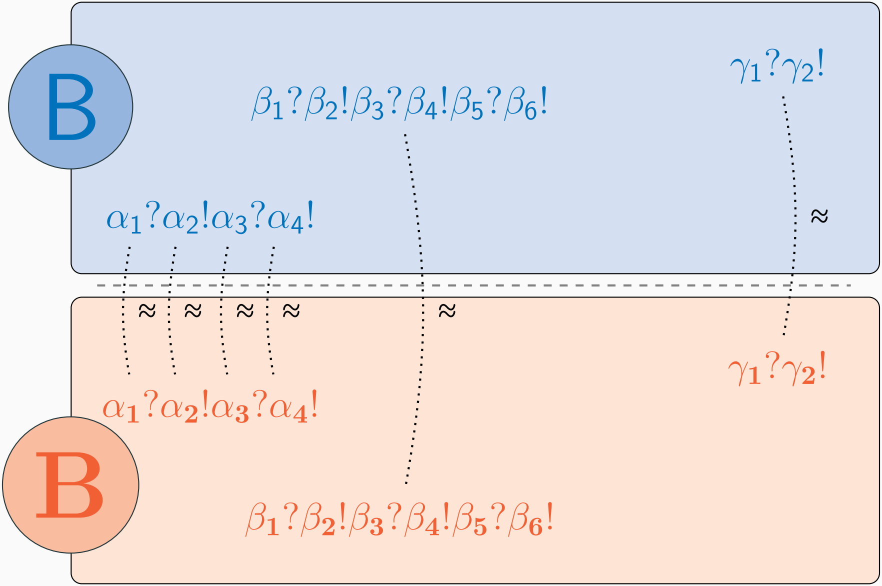
B

$\beta_{\mathbf{1}}?\beta_{\mathbf{2}}!\beta_{\mathbf{3}}?\beta_{\mathbf{4}}!\beta_{\mathbf{5}}?\beta_{\mathbf{6}}!$

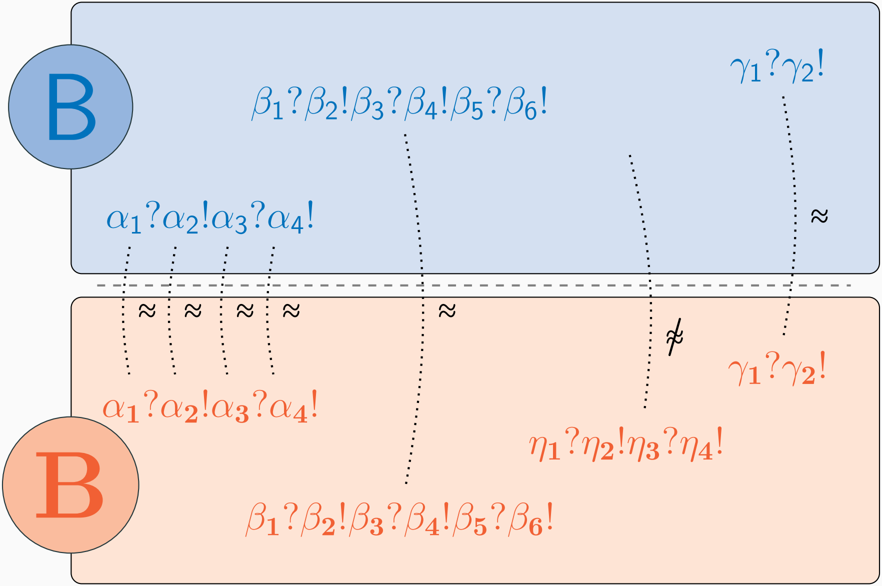
Safety, Visually



Safety, Visually



Safety, Visually



Extra Target Actions

- Actions η have no source-level counterpart

Extra Target Actions

- Actions η have no source-level counterpart
- What can they do?

Extra Target Actions

- Actions η have no source-level counterpart
- What can they do?

- Good η: outgoing action (!), respect safety, call them $\surd$

Extra Target Actions

- Actions η have no source-level counterpart
- What can they do?

- Good η: outgoing action (!), respect safety, call them $\surd$
- Bad η: incoming action (?), can trigger undesider behaviour

Extra Target Actions

- Actions η have no source-level counterpart
- What can they do?

- Good η: outgoing action (!), respect safety, call them $\surd$
- Bad η: incoming action (?), can trigger undesider behaviour
  outgoing action (!), violate safety

Extra Target Actions

- Actions η have no source-level counterpart
- What can they do?

- Good η: outgoing action (!), respect safety,

outgoing action (!), violate safety

We have no control over η?

But if we only use $\checkmark$ for η! we can

preserve safety and hypersafety

Recall

- π is a property: $\{t\}$
- if π is safety, we can express it dually via the set of bad prefixes $\{m\}$ such that $\{m\} :: \pi$
- compact notation: $\{m\}$ = m

Recall

- π is a property: $\{t\}$
- if π is safety, we can express it dually via the set of bad prefixes $\{m\}$ such that $\{m\} :: \pi$
- compact notation: $\{m\}$ = m
- H is a hyperproperty: $\{\pi\}$, $\{\{t\}\}$
- if H is hypersafety we can express it dually via the set of sets of bad prefixes $\{\{m\}\}$ such that $\{\{m\}\} :: H$
- compact notation: $\{\{m\}\}$ = $\mathbb{M}$

Safety Preservation

$\mathsf{m} \overset{\mathrm{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m

Safety Preservation

$\mathsf{m} \overset{\mathsf{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m
- all prefixes of the form $\mathbf{m}\alpha?\alpha!$ where

Safety Preservation

$\mathsf{m} \overset{\mathsf{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m
- all prefixes of the form $\mathbf{m}\alpha?\alpha!$ where any bad input is not responded to with a $\surd$

Safety Preservation

$\mathsf{m} \overset{\mathsf{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m
- all prefixes of the form $\mathbf{m}\alpha?\alpha!$ where any bad input is not responded to with a $\surd$
  - $\mathbf{m}$ may contain $\surd$, but it is otherwise related to a m

Safety Preservation

$\mathsf{m} \overset{\mathsf{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m
- all prefixes of the form $\mathbf{m}\alpha?\alpha!$ where any bad input is not responded to with a $\surd$
    - $\mathbf{m}$ may contain $\surd$, but it is otherwise related to a m
    - $\alpha?$ has no $\alpha?$ counterpart

Safety Preservation

$\mathsf{m} \overset{\mathsf{SP}}{\approx} \mathbf{m}$ if $\mathbf{m}$ includes:

- all prefixes that relate to prefixes in m
- all prefixes of the form $\mathbf{m}\alpha?\alpha!$ where any bad input is not responded to with a $\surd$
  - $\mathbf{m}$ may contain $\surd$, but it is otherwise related to a m
  - $\alpha?$ has no $\alpha?$ counterpart
  - $\alpha!$ is no $\surd$

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets whose single prefix $\mathbf{m}\alpha?\alpha!$
    - $\mathbf{m}$ may contain $\surd$, but is otherwise related to a prefix in $\mathbb{M}$

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets whose single prefix $\mathbf{m}\alpha?\alpha!$
  - $\mathbf{m}$ may contain $\surd$, but is otherwise related to a prefix in $\mathbb{M}$
  - $\alpha?$ has no $\alpha?$ counterpart

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets whose single prefix $\mathbf{m}\alpha?\alpha!$
  - $\mathbf{m}$ may contain $\surd$, but is otherwise related to a prefix in $\mathbb{M}$
  - $\alpha?$ has no $\alpha?$ counterpart
  - $\alpha!$ is no $\surd$

Hypersafety Preservation

$\mathbb{M} \overset{\mathsf{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets whose single prefix $\mathbf{m}\alpha?\alpha!$
  - $\mathbf{m}$ may contain $\surd$, but is otherwise related to a prefix in $\mathbb{M}$
  - $\alpha?$ has no $\alpha?$ counterpart
  - $\alpha!$ is no $\surd$

as before!

Hypersafety Preservation

$\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ if $\mathbb{M}$ includes:

- all sets of prefixes that relate to sets of prefixes in $\mathbb{M}$
- singleton sets whose single prefix $\mathbf{m}\alpha?\alpha!$
  - $\mathbf{m}$ may contain $\surd$, but is otherwise related to a prefix in $\mathbb{M}$
  - $\alpha?$ has no $\alpha?$ counterpart
  - $\alpha!$ is no $\surd$

  as before!

Singletons: minimum addition s.t. any $\mathbf{P}$ not responding even once as $\surd$ are bad

NI stays NI

Q: How do we know that $\overset{\text{SHP}}{\approx}$ is correct?

NI stays NI

Q: How do we know that $\overset{\text{SHP}}{\approx}$ is correct?

We prove something!

NI stays NI

Q: How do we know that $\overset{\text{SHP}}{\approx}$ is correct?

We prove something!

Theorem (Non-interference is preserved)

Let $\mathbb{M} :: \mathbb{NI}$. *Let* $\mathbb{M} \overset{\text{SHP}}{\approx} \mathbb{M}$ *and let* $\mathbb{S}$ *be a hyperproperty such that* $\mathbb{M} :: \mathbb{S}$. *Then,* $\forall \mathbf{t} \in \mathbb{S}, \mathbf{t} \in \mathbb{NI}$.

Preserving the Meaning (Hyper)Safety

- We know how to define when two (hyper)safety properties encode the same property

Preserving the Meaning (Hyper)Safety

- We know how to define when two (hyper)safety properties encode the same property
- Can we devise a compiler that preserves hypersafety across languages?

Preserving the Meaning (Hyper)Safety

- We know how to define when two (hyper)safety properties encode the same property
- Can we devise a compiler that preserves hypersafety across languages?
- How will it differ from RSC / RHS ?

Trace-Preserving Compilation

With these assumptions:

Trace-Preserving Compilation

With these assumptions:

- reactive setting: only I/O traces (implicitly robust)
- traces capture all form of behaviour

Informal Trace-Preserving Compilation

TPC: $\forall \mathsf{C} \in \mathsf{S}.\ \mathsf{TR}(\llbracket \mathsf{C} \rrbracket^{\mathsf{S}}_{\mathbf{T}}) = \mathsf{TR}(\mathsf{C}) \cup \mathbf{B}_{\mathsf{C}}$.

Informal Trace-Preserving Compilation

TPC: $\forall \mathsf{C} \in \mathsf{S}.\ \mathsf{TR}(\llbracket \mathsf{C} \rrbracket^{\mathsf{S}}_{\mathbf{T}}) = \mathsf{TR}(\mathsf{C}) \cup \mathbf{B}_{\mathsf{C}}$.

- all source traces

Informal Trace-Preserving Compilation

TPC: $\forall C \in S.\ TR([\![C]\!]^{S}_{\mathbf{T}}) = TR(C) \cup \mathbf{B}_{C}$.

- all source traces
- plus all bad ones:

Informal Trace-Preserving Compilation

TPC: $\forall \mathsf{C} \in \mathsf{S}.\ \mathsf{TR}(\llbracket \mathsf{C} \rrbracket^{\mathsf{S}}_{\mathbf{T}}) = \mathsf{TR}(\mathsf{C}) \cup \mathbf{B}_{\mathsf{C}}$.

- all source traces
- plus all bad ones:
    - any trace that to an input $\alpha?$ that has no related source counterpart $\alpha?$

Informal Trace-Preserving Compilation

TPC: $\forall \mathsf{C} \in \mathsf{S}.\ \mathsf{TR}(\llbracket \mathsf{C} \rrbracket^{\mathsf{S}}_{\mathbf{T}}) = \mathsf{TR}(\mathsf{C}) \cup \mathbf{B}_{\mathsf{C}}$.

- all source traces
- plus all bad ones:
  - any trace that to an input $\alpha?$ that has no related source counterpart $\alpha?$
  - responds with a $\surd$

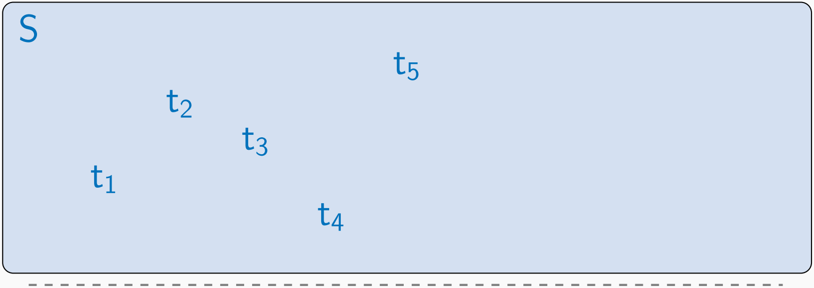
Informal Trace-Preserving Compilation

TPC: $\forall \mathsf{C} \in \mathsf{S}.\ \mathbf{TR}(\llbracket \mathsf{C} \rrbracket^{\mathsf{S}}_{\mathbf{T}}) = \mathsf{TR}(\mathsf{C}) \cup \mathbf{B}_{\mathsf{C}}$.

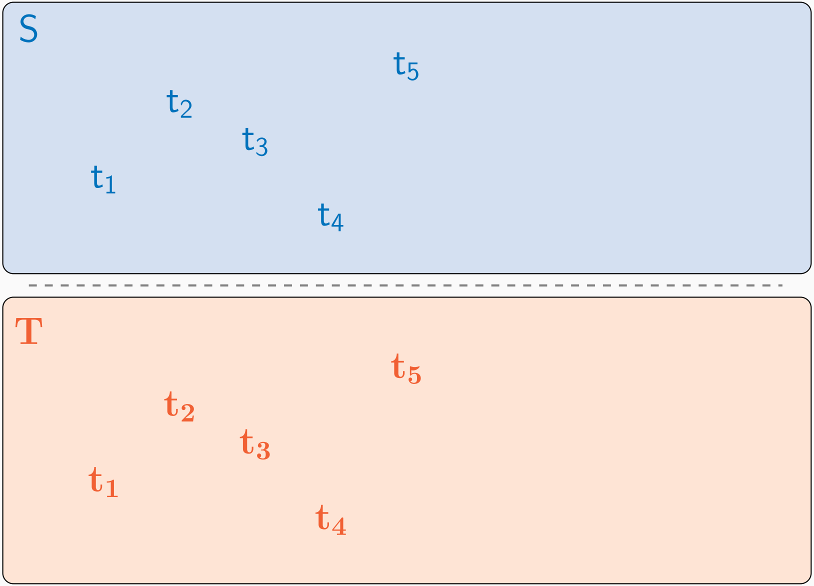
- all source traces (actually: $\mathbf{t} \approx \mathsf{t} \in \mathsf{TR}(\mathsf{C})$)
- plus all bad ones:
  - any trace that to an input $\alpha?$ that has no related source counterpart $\alpha?$
  - responds with a $\surd$

TPC Preserves Safety

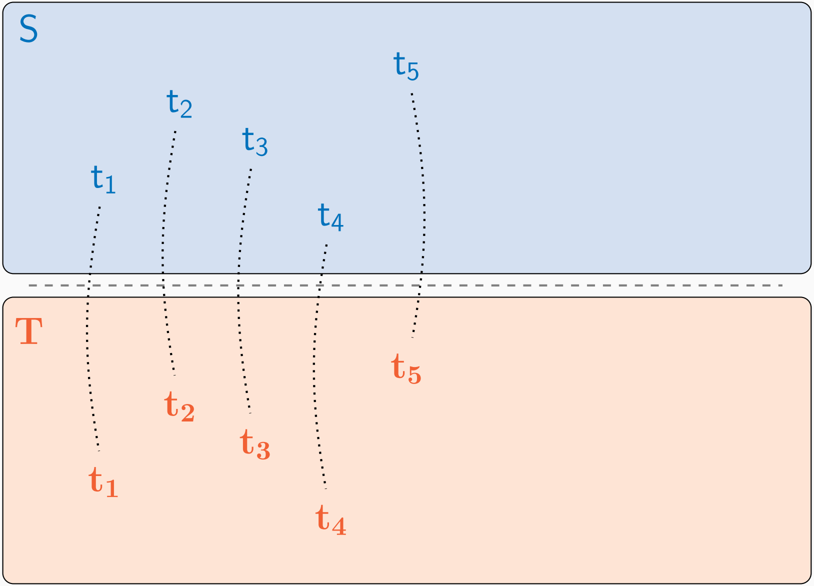
TPC Preserves Safety



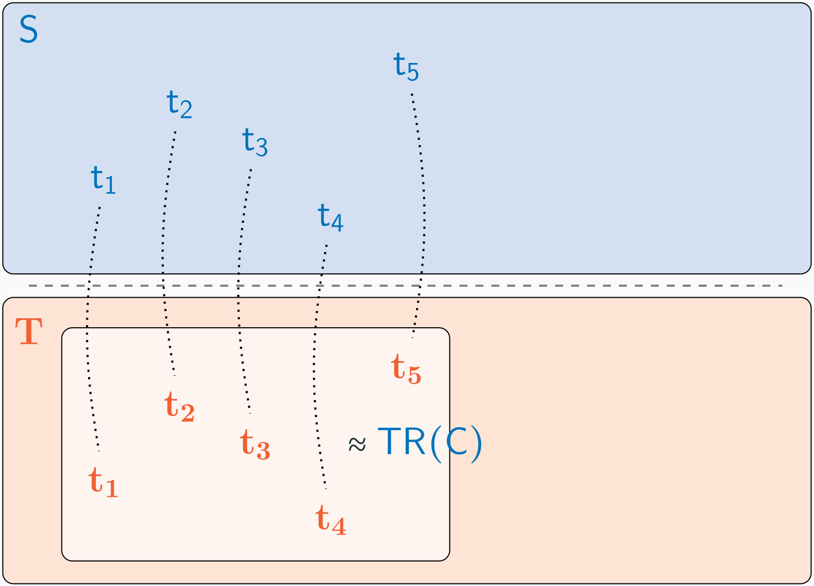
TPC Preserves Safety



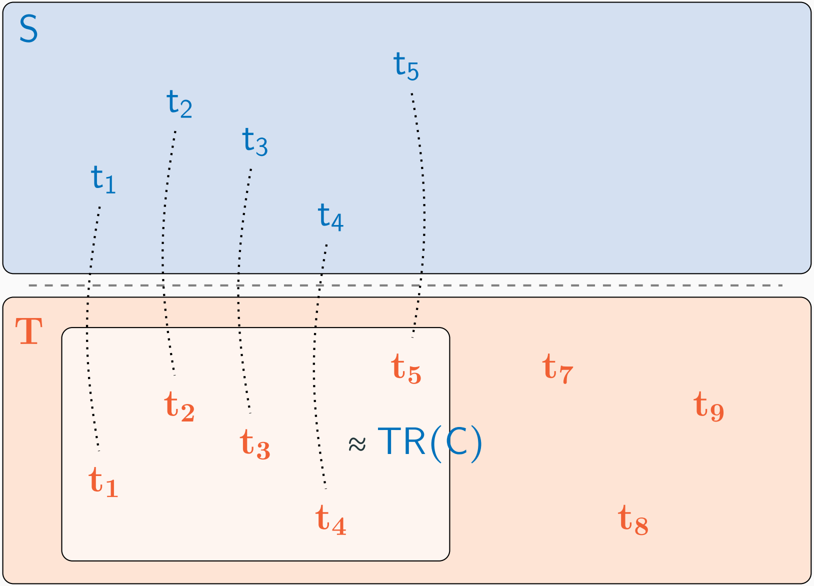
TPC Preserves Safety



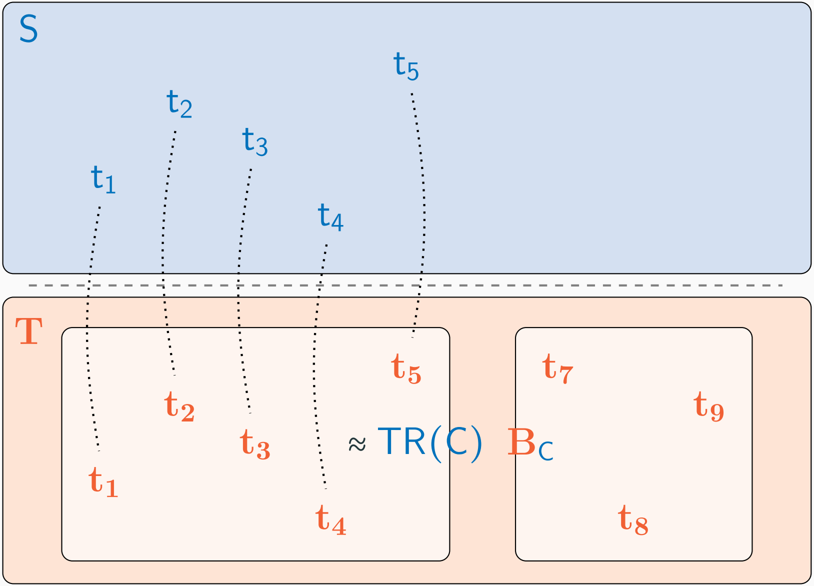
TPC Preserves Safety



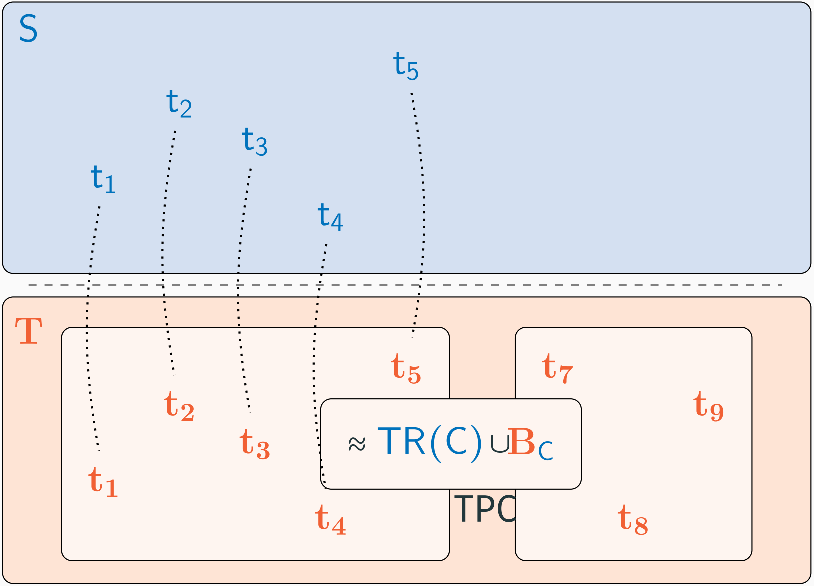
TPC Preserves Safety



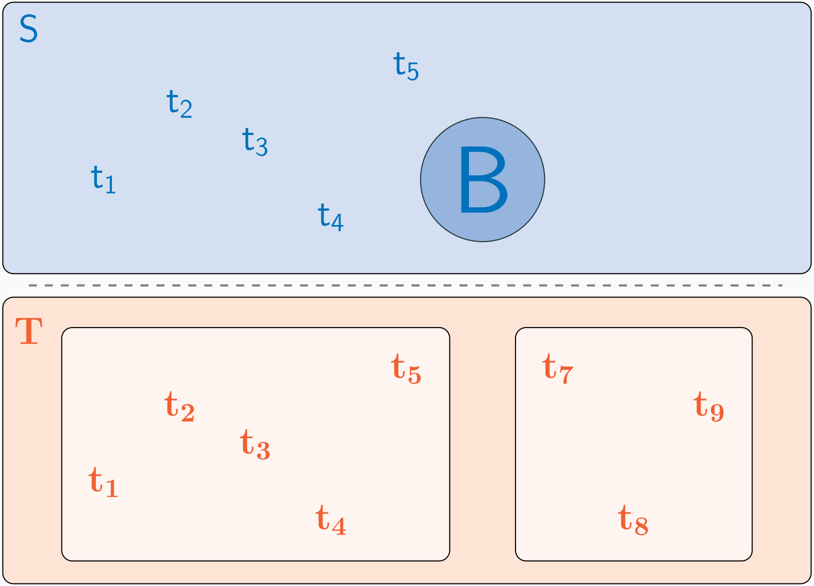
TPC Preserves Safety



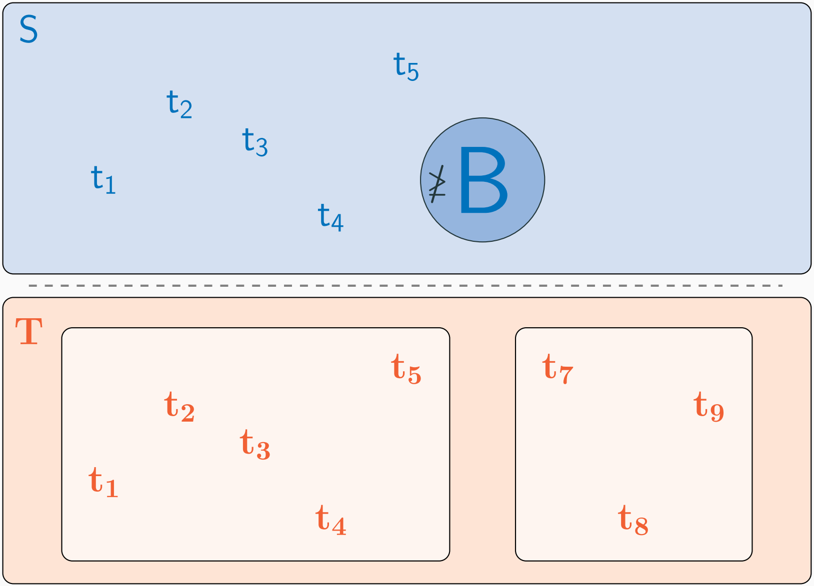
TPC Preserves Safety



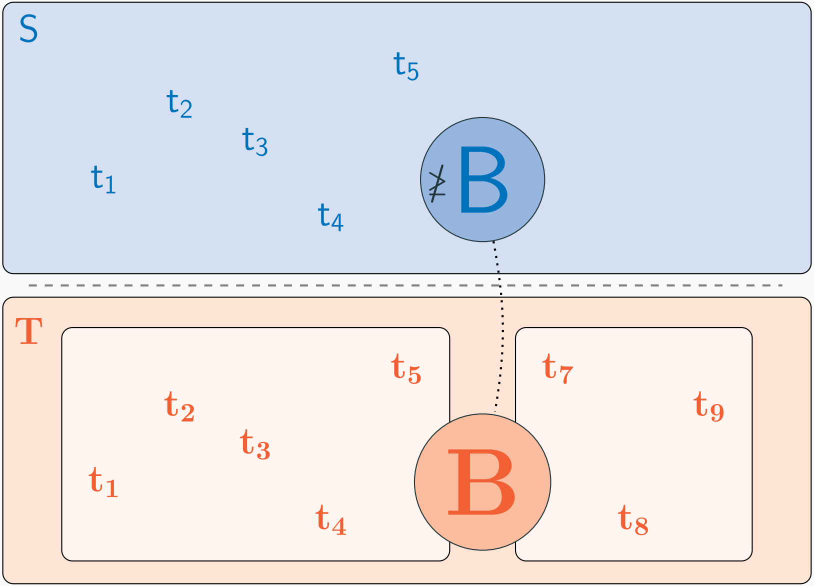
TPC Preserves Safety



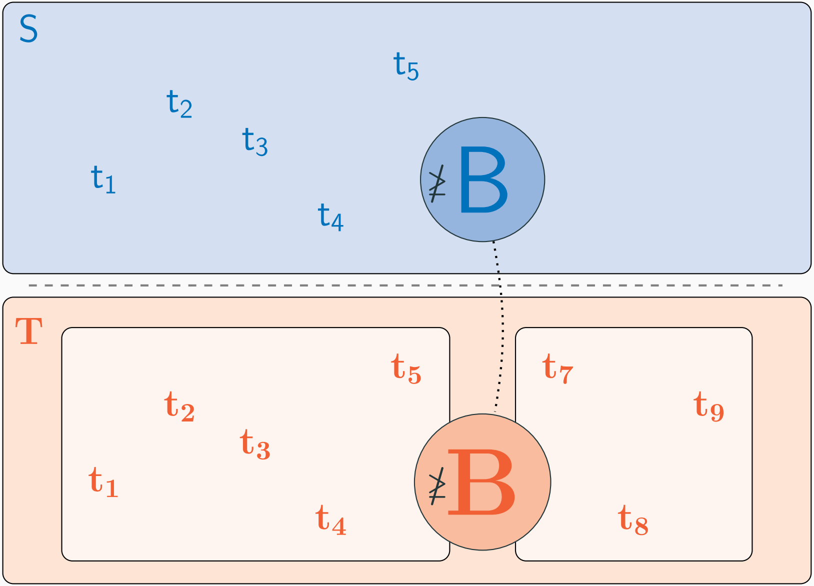
TPC Preserves Safety



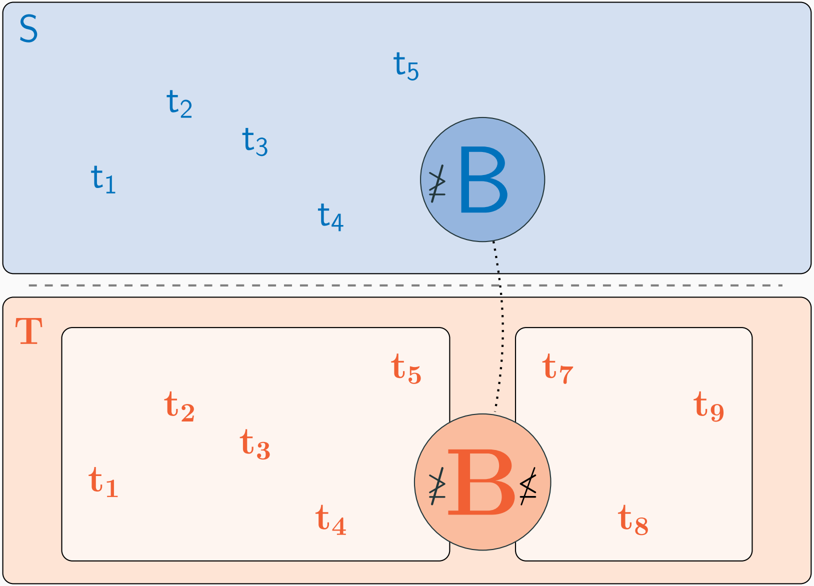
TPC Preserves Safety



TPC Preserves Safety



TPC Preserves Safety



TPC and FAC and RC

- TPC is propositional
- FAC is relational
- RC has both kinds

TPC and FAC and RC

- TPC is propositional
- FAC is relational
- RC has both kinds
- all are robust (TPC implicitly quantifies over all $\mathfrak{C}$)

TPC and FAC and RC

- TPC is propositional
- FAC is relational
- RC has both kinds
- all are robust (TPC implicitly quantifies over all $\mathfrak{C}$)
- RC criteria (except for RSC and RHS) deal with properties
- TPC cannot handle properties