PriSC 2023

Securely Compiling F★ Programs With IO and Then Linking Them Against Weakly-Typed Interfaces

Cezar-Constantin **Andrici**, Cătălin **Hriţcu,** Théo **Winterhalter**

21 January 2023

Statically verified partial IO program is compiled and linked against adversarial unverified context

P^S

statically verified
partial IO program in $F^\star$

P^S interacts with its context via strongly-typed higher-order interface
includes refinement types and pre-post conditions

compile
usually erases specs

P^T interacts via a weakly-typed higher-order interface
without refinement types and pre-post conditions

P^T

$\bowtie$

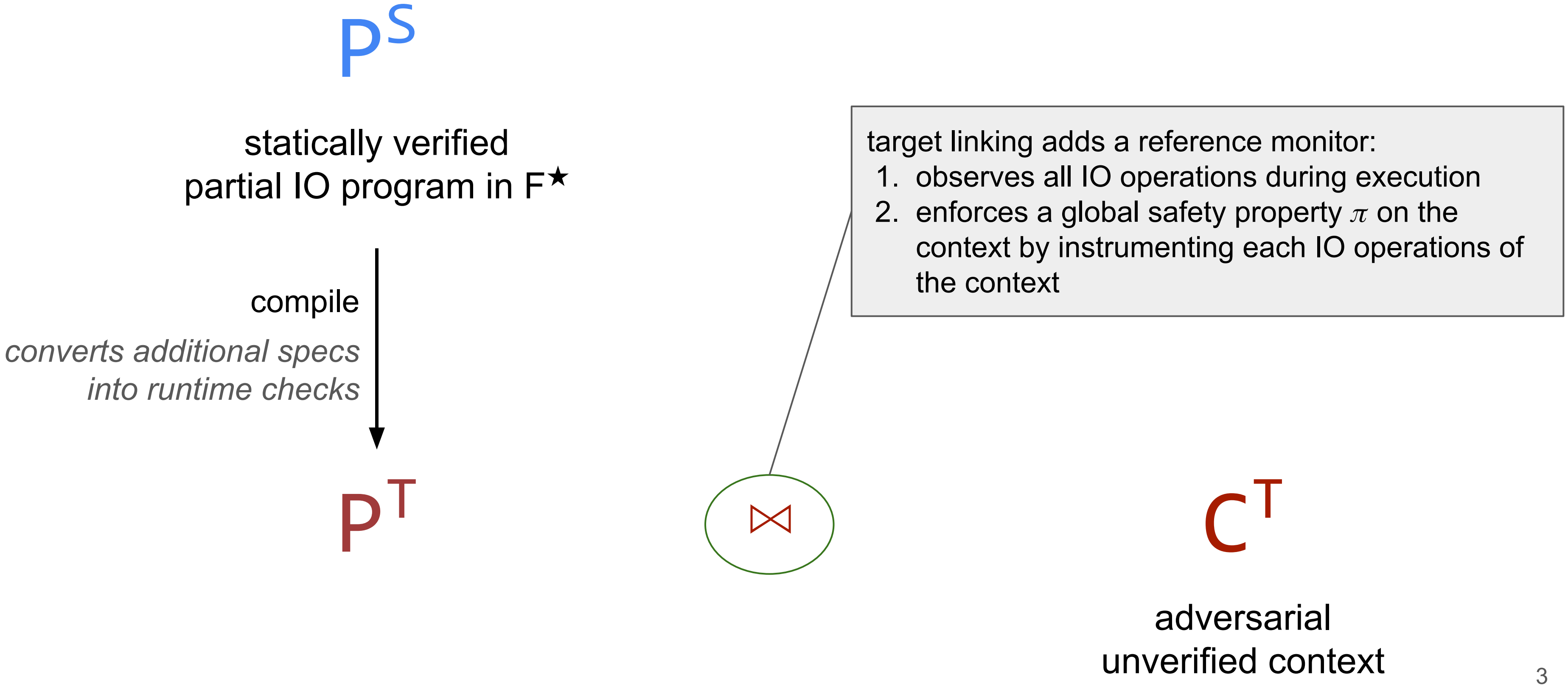
usually done naively and this is unsound

the interface must be strengthened

C^T

adversarial
unverified context

Solution: soundly strengthen the interface by dynamically verifying it



P^S

statically verified
partial IO program in $F^\star$

compile

converts additional specs into runtime checks

P^T

$\bowtie$

target linking adds a reference monitor:

1. observes all IO operations during execution
2. enforces a global safety property π on the context by instrumenting each IO operations of the context

C^T

adversarial
unverified context

Static verification of **partial** IO program (running example)

```
let P^S ctx :
  IO unit (ensures (λ r t → (EOpenfile "/etc/passwd") not_in t)) =
    let fd = openfile "data.csv" in
    let r = ctx fd in
    close fd
```

✓

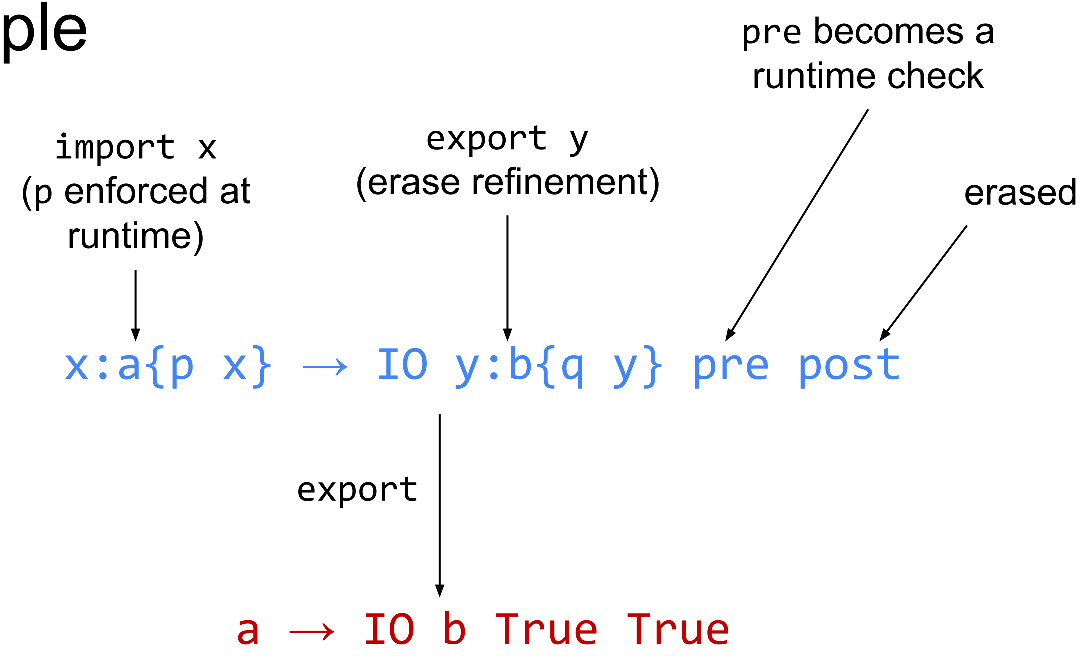
```
val ctx : file_descr →
  IO string (ensures (λ r t →
            length r < 500 ∧
            (EOpenfile "/etc/passwd") not_in t))
```

Enforcing the additional logical assumptions using higher-order contracts

```
val ctx : file_descr →
  IO string (ensures (λ r t →
              length r < 500 ∧
              (EOpenfile "/etc/passwd") not_in t))
```

During compilation, we use a mechanism inspired from higher-order contracts that wraps the context.

`export` example



`pre` becomes a runtime check

`import x`
(p enforced at runtime)

`export y`
(erase refinement)

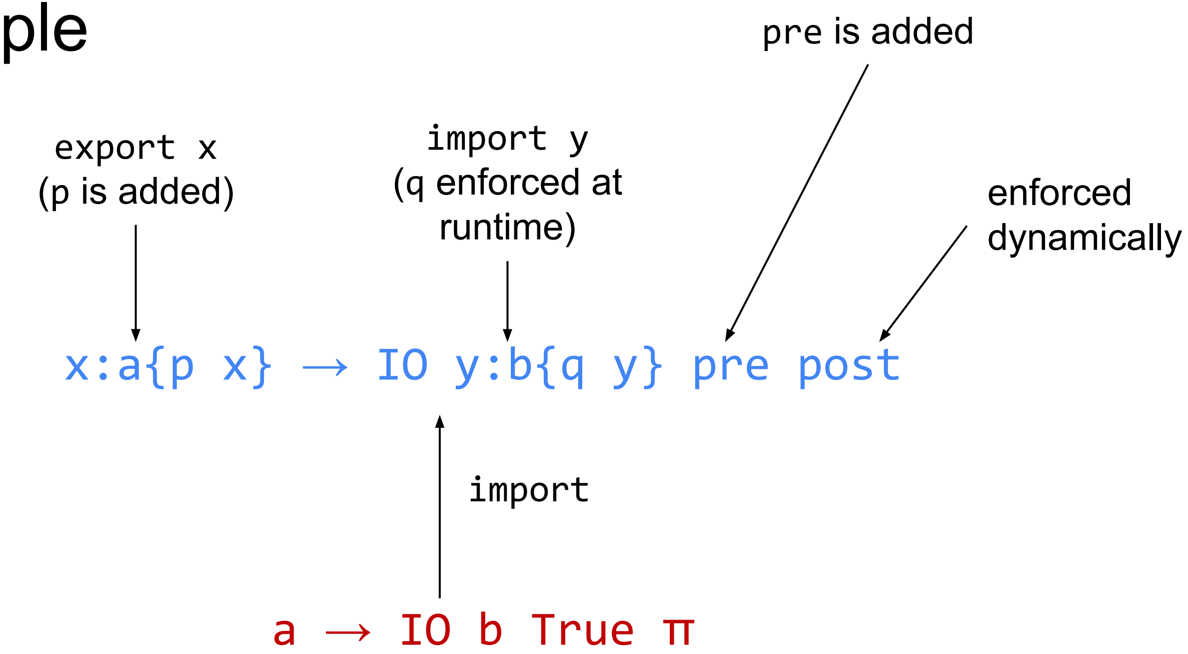
erased

`x:a{p x} → IO y:b{q y} pre post`

`export`

`a → IO b True True`

`import` example



pre is added

`export x`
(p is added)

`import y`
(q enforced at runtime)

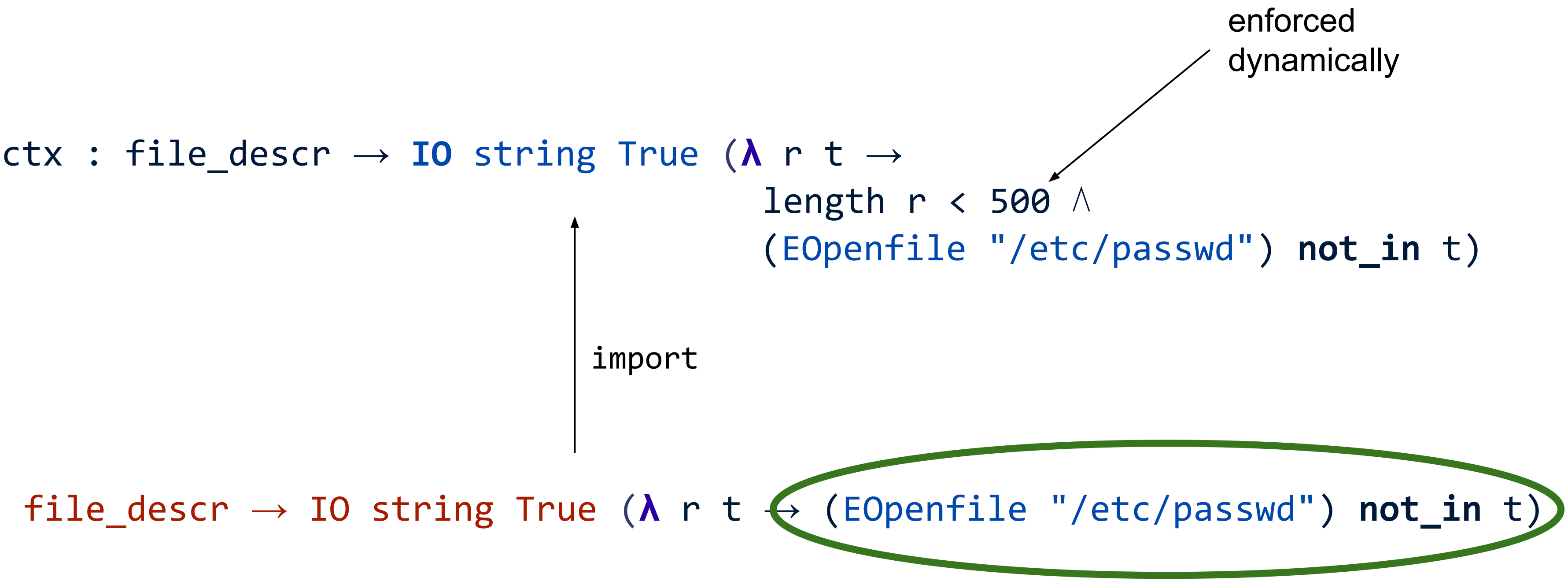
enforced dynamically

`x:a{p x} → IO y:b{q y} pre post`

`import`

`a → IO b True π`

import example



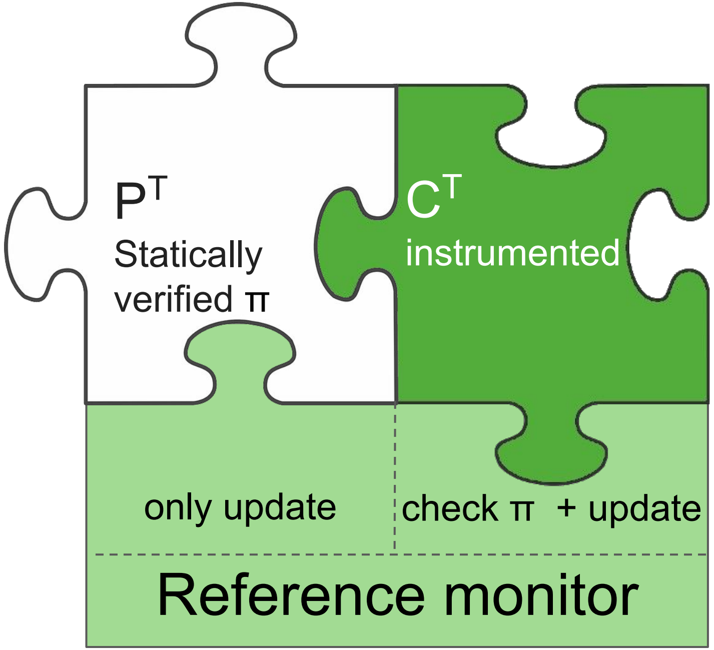
enforced
dynamically

```
ctx : file_descr → IO string True (λ r t →
                               length r < 500 ∧
                               (EOpenfile "/etc/passwd") not_in t)
```

import

```
file_descr → IO string True (λ r t → (EOpenfile "/etc/passwd") not_in t)
```

Enforcing the safety property at the target level



- `P`^{T} and `C`^{T} share the IO operation they can perform, but we give them different implementations during linking.
- The reference monitor instruments `C`^{T} to enforce a global safety property `π.`

In our example:
`π =` “block all openfiles on `/etc/passwd`”

Formalization in the proof-oriented programming language F★

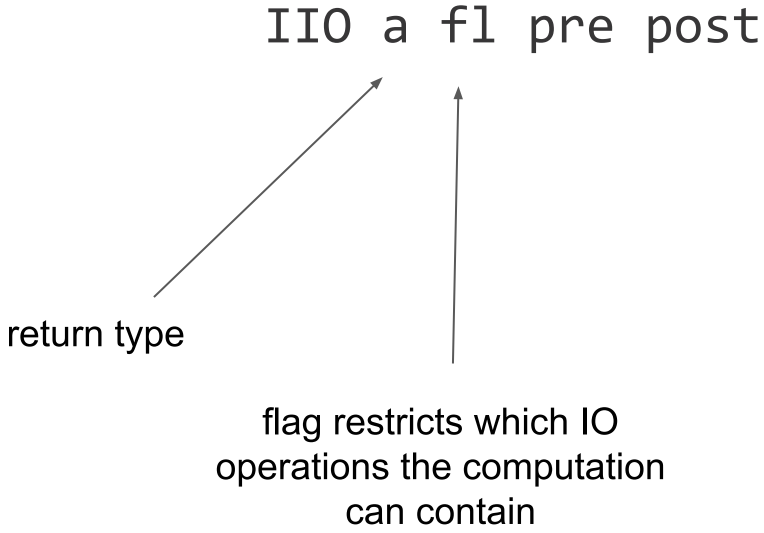
(Swamy *et al.* POPL 2016)

Implement & reason about the compilation chain directly in F★:

- Shallow embeddings of the languages
- Model and reason about the compilation chain:
  - Soundness of instrumentation enforcing safety property
  - *Robust Relational Hyperproperty Preservation,*
    strongest secure compilation criterion of Abate *et al. (CSF'19).*

Model of IO Computations - Indexed Monad

- The partial program P and the context C are interacting IO computations.
- Monad indexed by specs that encode trace properties (Andrici *et al.* HOPE‘22).



`IIO a fl pre post`

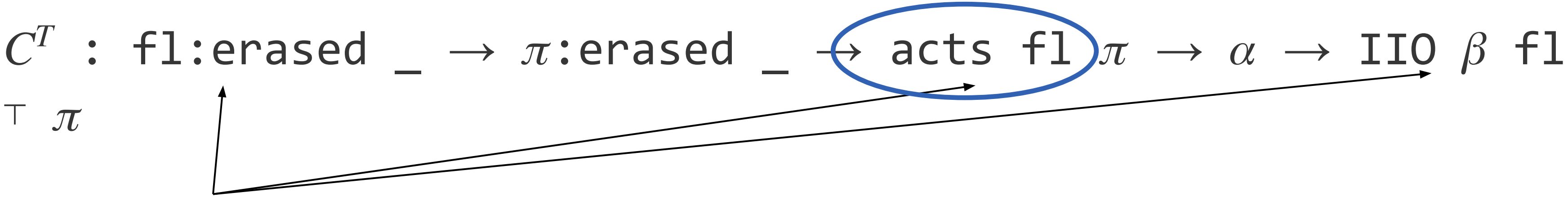
return type

flag restricts which IO operations the computation can contain

Type of target context

(first-order setting, scales to higher-order)

The target context can be linked with any IO implementation of the operations.



$$C^T : \texttt{fl:erased \_} \rightarrow \pi\texttt{:erased \_} \rightarrow \texttt{acts fl}\ \pi \rightarrow \alpha \rightarrow \texttt{IIO}\ \beta\ \texttt{fl}$$

$$\top\ \pi$$

- flag polymorphic
  - label erased makes `fl` unusable in the computation
  - C^T cannot use directly the IO operations

Linking instantiates C^T as follows

$$C^T : \texttt{fl:erased \_} \to \pi\texttt{:erased \_} \to \texttt{acts fl}\ \pi \to \alpha \to \texttt{IIO}\ \beta\ \texttt{fl}\ \top\ \pi$$

Definition of linking: $C^T[P^T] = P^T\ C_\pi$ where

$$C_\pi = C^T\ \texttt{AllActions}\ \pi\ (\texttt{instrument io\_acts}\ \pi)$$

We can implement `instrument` in the IIO monad. Our IIO monad has an extra `GetTrace` operation that returns the trace until now.

Soundness

If $\mathtt{P}^{\mathtt{S}}$ is statically verified to also respect π, then we can prove the following:

$$\forall\ \pi.\ \forall \mathtt{P}^{\mathtt{S}}_{\pi}.\ \forall \mathtt{C}^{\mathtt{T}}.\ \mathtt{Behav}(\mathtt{C}^{\mathtt{T}}[\mathtt{P}^{\mathtt{S}}_{\pi}\downarrow])\ \subseteq\ \pi$$

$$\mathtt{C}^{\mathtt{T}}[\mathtt{P}^{\mathtt{T}}]\ =\ \mathtt{P}^{\mathtt{T}}\ C_{\pi}$$

$$\mathtt{P}^{\mathtt{S}}\downarrow\ =\ \lambda\ C_{\pi} \rightarrow \mathtt{P}^{\mathtt{S}}\ (\mathtt{import}\ C_{\pi})$$

The target linking produces an `IIO True` π computation, thus soundness is ensured by F★ typing.

Robust Relational Hyperproperty Preservation (RrHP)

- Strongest criterion of Abate et al. (CSF’19):

$$\forall C^T.\ \exists C^S.\ \forall P^S.\ \mathtt{Behav}(C^T[P^S\downarrow]) = \mathtt{Behav}(C^S[P^S])$$

- Our languages contain `GetTrace` (a form of reflection):
  - source partial program and context and the target context cannot call `GetTrace` directly because of flag polymorphism.

- To prove such a criterion, one has to define a back-translation of contexts.
- We can prove the following syntactic equality (by unfolding the definitions):

$$\forall C^T.\ \forall P^S.\ C^T[P^S\downarrow] = C^T\uparrow[P^S]$$

Syntactic equality

We can prove the following syntactic equality (by unfolding the definitions):

$$\forall \mathtt{C}^{\mathtt{T}}.\ \forall \mathtt{P}^{\mathtt{S}}.\ \mathtt{C}^{\mathtt{T}}[\mathtt{P}^{\mathtt{S}}\downarrow] = \mathtt{C}^{\mathtt{T}}\uparrow[\mathtt{P}^{\mathtt{S}}]$$

$$\mathtt{C}^{\mathtt{T}}[\mathtt{P}^{\mathtt{T}}] = \mathtt{P}^{\mathtt{T}}\ C_{\pi} \qquad\qquad \mathtt{C}^{\mathtt{S}}[\mathtt{P}^{\mathtt{S}}] = \mathtt{P}^{\mathtt{S}}\ \mathtt{C}^{\mathtt{S}}$$

$$\mathtt{P}^{\mathtt{S}}\downarrow = \lambda\ C_{\pi} \rightarrow \mathtt{P}^{\mathtt{S}}\ (\mathtt{import}\ C_{\pi})$$

$$\mathtt{C}^{\mathtt{T}}\uparrow = \lambda\ \mathtt{fl}\ \pi\ \mathtt{iio\_acts} \rightarrow \mathtt{import}\ (\mathtt{C}^{\mathtt{T}}\ \mathtt{fl}\ \pi\ \mathtt{iio\_acts})$$

Our secure compilation proof is orders of magnitude simpler than most other proofs in this space.

Contributions

- Secure compilation chain for statically verified F* partial programs with IO;
  - Our mechanism scales for Higher-Order interfaces.
- Mechanized proof in F* that our secure compilation chain:
  - soundly enforces a global safety property π;
  - satisfies Robust Relational Hyperproperty Preservation
    - simple proof, follows by construction;

Ongoing & Future work

- Extend the IIO Monad with other effects such as non-termination, exceptions, and state – **our final goal is secure F*-ML interoperability**
- Case study: simple web server that supports third-party plugins;
- Proving (relational) hyperproperties about source partial programs in F*:
  - by exploiting flag polymorphism