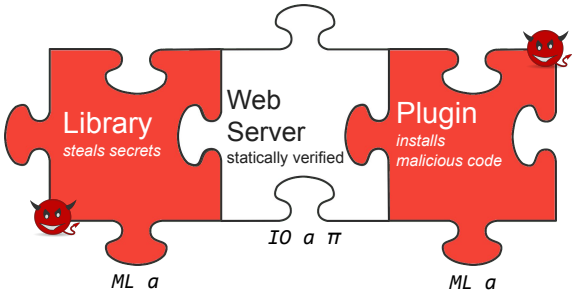
Gradual Enforcement of IO Trace Properties - Cezar Andrici

Advisors: Ştefan Ciobâcă, Cătălin Hriţcu, Guido Martínez, Exequiel Rivas, Éric Tanter

UNIVERSITATEA „ALEXANDRU IOAN CUZA" din IAŞI

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

Motivation and problem. Modern web servers are linked with *unverified* third-party plugins, therefore they can have *unintended_behavior*. How can we offer guarantees about what safety properties hold?




```
(Openfile,"file.md",fd),(Read,fd,"abc"),(Close,fd,())
```

Example of a IO trace of a program that reads from a file

I. Static verification of IO

The IO Effect:
➔ Annotate functions with pre-/postconditions
➔ Trace is allocated and updated at runtime
➔ Enforces trace properties
➔ Parametric in the underlying primitive actions
➔ Primitives can throw exceptions

```
val read : (fd:file_descr) → IO string
 (requires (λ trace → is_open fd trace))
 (ensures (λ msg lt → lt = [(Read,fd,msg)]))
```

Example. Primitive read

Build and verified in F* for programs with input-output behavior

Safety Properties - π

Allows or blocks the current execution:
➔ bool function
➔ **Future work:** extract π from LTL formulas

```
let π trace next_operation : bool =
  match next_operation with
  | (Openfile, fnm) → fnm != "secret.txt"
  | _ → true
```

Example of a trace property

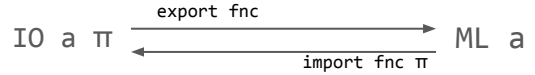
II. Gradual verification of IO

Extended primitives by using wrapping:
➔ Accept an extra precondition - π
➔ variants: static / mixed / dynamic enforcement
➔ Postconditions are enforced statically
➔ **Future work:** a more efficient representation of the trace

```
let example1 () : IO string π =
  let fd = mixed_openfile π "secret.txt" in
  dynamic_close π fd;
  mixed_read π fd
```

Example. this does not statically verify, because *fd* is read after it is closed (for any property *π*)

New transformation functions between F* effects:
➔ Effect ML can not have specification
➔ Effect ML is a safe subset of a Meta Language
➔ *export* - converts static pre-conditions to dynamic checks
➔ *import* - enforces π on each IO operation
➔ **Future work:** extend IO with state and divergence



III. Interoperability verified-unverified

➔ We can prove that the whole program respects π
➔ Only the plugin is monitored

```
val webserver : (int → IO int π) → IO unit π
val plugin : int → ML int
let main () : IO unit π =
  webserver (import plugin π)
```

Results

→ seamless interoperability between static and dynamic checking of IO trace properties
→ seamless interoperability between verified and unverified code
→ ongoing case study on how to extend a verified web server with a ML-plugin mechanism.