Programming language research meets new architectures

Gabriel Scherer

Parsifal, INRIA Saclay
OCaml

March 30, 2021

Programming language theory

Programming languages as mathematical objects
(set of programs, source-level execution function).

- Clear setting to study program properties.
- Can compare to real implementations.
- Can prove properties of implementations.

Proof assistants: automated checking of large-scale human-written proofs about maths or software.

Success stories: seL4, Compcert.

Today: programming language theory to define, design **specifications**.

Simple specifications

Formal proofs guarantee the absence of bugs... within the specification.

We need tools to **specify** properties clearly.

Example: "the compiler is correct". What does that mean, precisely?

Simple specifications

Formal proofs guarantee the absence of bugs... within the specification.

We need tools to **specify** properties clearly.

Example: “the compiler is correct”. What does that mean, precisely?

```
char *decrypt_using_key(char *msg) {
  char key[KEY_SIZE];
  read_secret_key(key);
  char *plaintext = decrypt(msg, key);
  zero_out(key);
  return plaintext;
}
```

Full abstraction

A compiler $\mathrm{comp}(\_) : S \to T$ is **fully abstract** if

$$\forall p_1, p_2 \in S, \qquad p_1 \simeq_S p_2 \iff \mathrm{comp}(p_1) \simeq_T \mathrm{comp}(p_2)$$

($p_1 \simeq_L p_2$: indistinguishable by a reference/idealized interpreter for L)

Very simple statement.

Very strong property!

Full abstraction: example

$$\forall p_1, p_2 \in S, \qquad p_1 \simeq_S p_2 \iff \mathsf{comp}(p_1) \simeq_T \mathsf{comp}(p_2)$$

```
char *decrypt_using_secret_1(char *msg) {
  char key[KEY_SIZE]; read_secret_key(key);
  char *plaintext = decrypt(msg, key);
  zero_out(key);
  return plaintext;
}
char *decrypt_using_secret_2(char *msg) {
  char key[KEY_SIZE]; read_secret_key(key);
  char *plaintext = decrypt(msg, key);
  free(key);
  return plaintext;
}
```

Full abstraction: example

$$\forall p_1, p_2 \in S, \qquad p_1 \simeq_S p_2 \iff \mathrm{comp}(p_1) \simeq_T \mathrm{comp}(p_2)$$

```
char *decrypt_using_secret_1(char *msg) {
  char key[KEY_SIZE]; read_secret_key(key);
  char *plaintext = decrypt(msg, key);
  zero_out(key);
  return plaintext;
}
char *decrypt_using_secret_2(char *msg) {
  char key[KEY_SIZE]; read_secret_key(key);
  char *plaintext = decrypt(msg, key);
  free(key);
  return plaintext;
}
```

Full abstraction $\implies$ enforced privacy.

Full abstraction: consequences

$$\forall p_1, p_2 \in S, \qquad p_1 \simeq_S p_2 \iff \mathrm{comp}(p_1) \simeq_T \mathrm{comp}(p_2)$$

Example of properties preserved by a full-abstraction compiler:

- immutability guarantees
  $\implies$ memory access protection
- privacy/encapsulation: data not reachable from the outside
  $\implies$ enclaves
- preservation of control flow, even when calling user code/callbacks
  $\implies$ control-flow integrity

Most compilers are **not** fully-abstract,
their target lacks runtime protection features.

Full abstraction: summary

Simple, interesting property to think about.

Especially for designers of instruction-set-level features!

Possible in some cases: Javascript
Fully-Abstract Compilation to Javascript,
Fournet, Swamy, Chen, Dagand, Strub, and Livshits [2013].

Zooming back

Programming Language Theory research brings formal tools

relevant to study low-level systems as well.

Specify properties of interest, prove them.

Thanks!

Cédric Fournet, Nikhil Swamy, Juan Chen, Pierre-Evariste Dagand, Pierre-Yves Strub, and Benjamin Livshits. Fully Abstract Compilation to JavaScript. 2013. URL https://hal.inria.fr/hal-00780803.

Aïna Linn Georges, Armaël Guéneau, Thomas Van Strydonck, Amin Timany, Alix Trieu, Sander Huyghebaert, Dominique Devriese, and Lars Birkedal. Efficient and provable local capability revocation using uninitialized capabilities. 2021.

Lau Skorstengaard, Dominique Devriese, and Lars Birkedal. Reasoning about a machine with local capabilities: Provably safe stack and return pointer management. 2018.

Lau Skorstengaard, Dominique Devriese, and Lars Birkedal. Stktokens: Enforcing well-bracketed control flow and stack encapsulation using linear capabilities. 2019.