A Fully-Abstract Translation of Pointers to Capabilities

Akram El-Korashy, Stelios Tsampas, Marco Patrignani, Dominique Devriese, Deepak Garg, Frank Piessens
MPI-SWS, KU-Leuven, CISPA-Saarland & Stanford

Outline

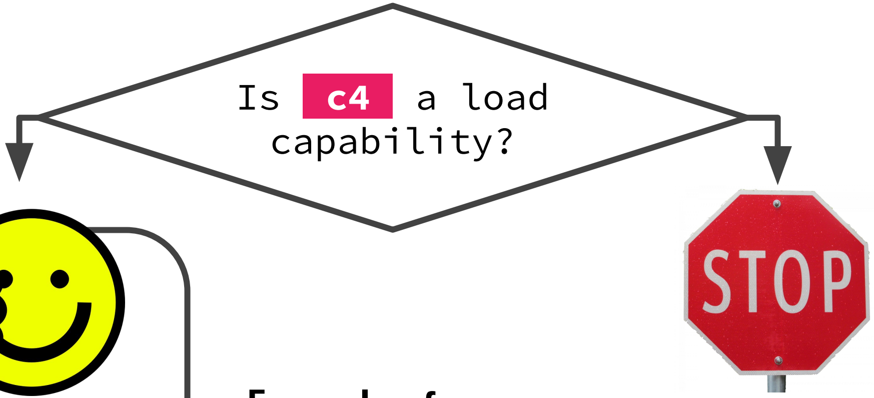
- What is a capability-based computer?
- Our model of a “pointers to capabilities” translation
- What is a fully-abstraction translation?
- Our proof idea: ternary simulation

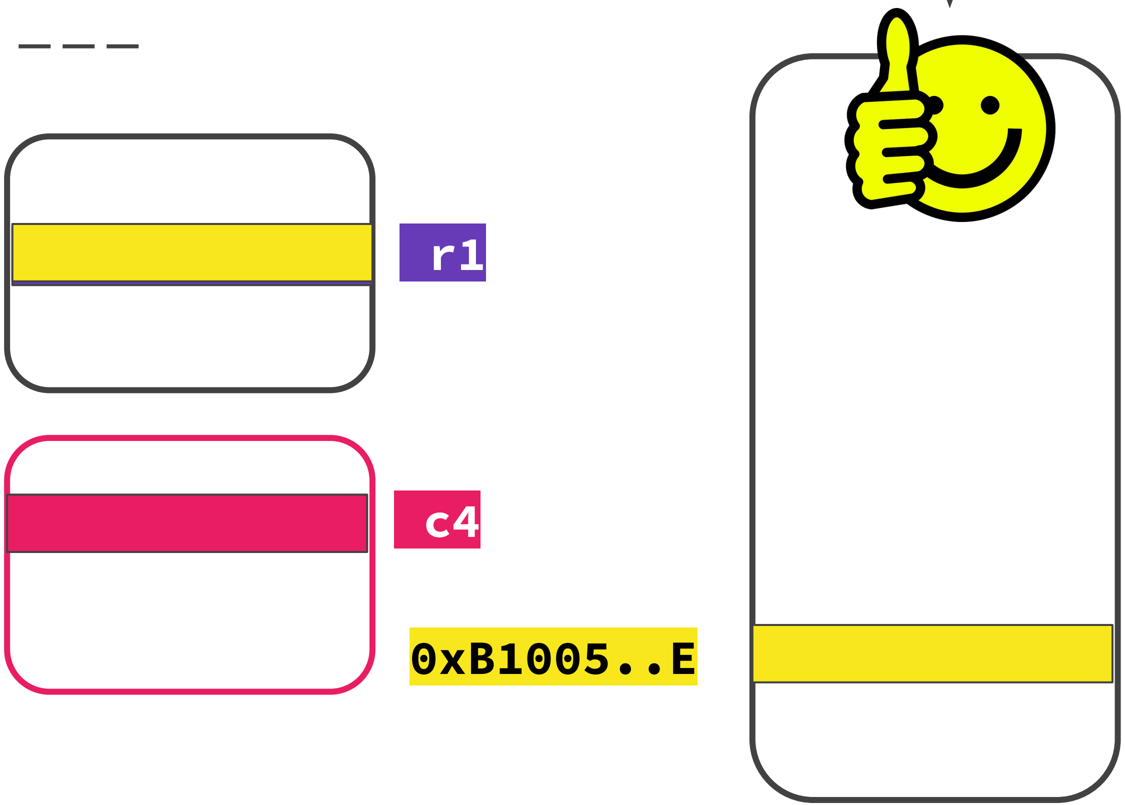
```
A capability is an access token that gives permission to perform some
operation on some resource.
```

```
load r1 , c4
```

What is a capability-based computer?

What is a capability computer?





Example of a memory instruction:

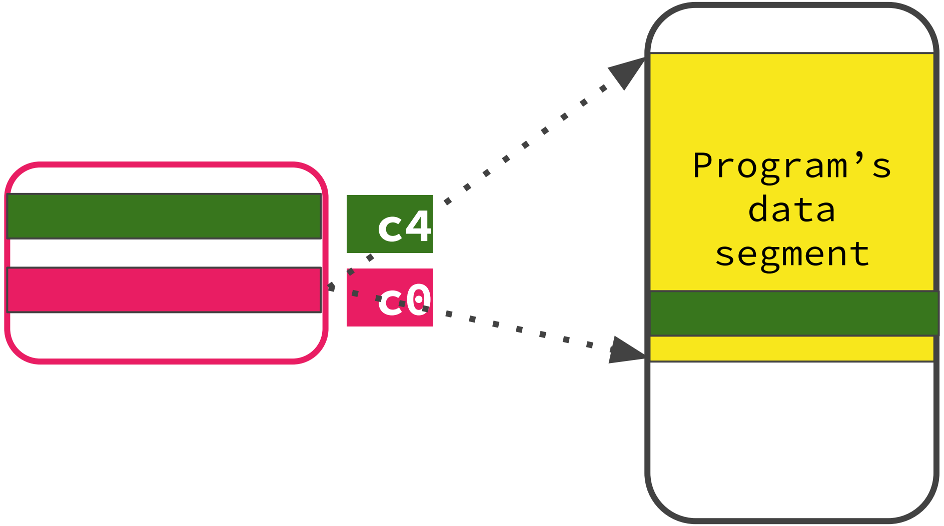
load r1 , c4

For a successful load, **c4** needs to contain a valid "**load capability**" on a region containing **0xB1005..E**

Role of **compiler** and **loader** in a capability computer

- The **loader** places a capability (e.g., in c0) that authorizes memory operations on **all of the program's data segment**.

- The **compiler creates sub-capabilities** as needed and uses them for the corresponding instructions.




```
lim   c4, c0, r3, r2
load  r1 , c4
store ...
```

Our model of a “pointers-to-capabilities” translation

Example program and its translation

```
#include “networking.h”

iobuffer [512];
static secret;
main() {
  iobuffer[42] = 4242;
  send_rcv(&iobuffer);
  handle_secret();
}
handle_secret() { … }
```

```
#include “networking.h”

data_segment_size: 513

main() {
  inc(ddc, 42) = 4242;
  send_rcv(lim(ddc, 0, 512));
  handle_secret();
}
handle_secret() { … }
```

Spatially-safe semantics for pointers

Secure implementation of this semantics

What is a fully-abstract translation?

What is a fully-abstract translation?

The $\Rightarrow$ direction is called preservation of contextual equivalence

$$\forall P_1\ P_2$$

$$\forall C_S.\ C_S[P_1] \approx C_S[P_2]$$
$$\Longleftrightarrow$$
$$\forall C_T.\ C_T[P_1\downarrow] \approx C_T[P_2\downarrow]$$

P_1, P_2: Two source programs

$P_1\downarrow$, $P_2\downarrow$: Translated versions of P_1 and P_2

C_S: Source context

C_T: Target context

How to prove the ⇒ direction? Use trace equivalence (1)

$$\forall P_1\ P_2$$

$$\forall C_S.\ C_S[P_1] \approx C_S[P_2]$$
$$\Rightarrow$$
$$\forall C_T.\ C_T[P_1{\downarrow}] \approx C_T[P_2{\downarrow}]$$

Design traces that record all "observable behavior". Then instead prove:

The traces of a program are all its possible interactions with any context

$$\forall P_1\ P_2$$

$$\mathtt{traces}(P_1) = \mathtt{traces}(P_2)$$
$$\Rightarrow$$
$$\mathtt{traces}(P_1{\downarrow}) = \mathtt{traces}(P_2{\downarrow})$$

How to prove the ⇒ direction? Use trace equivalence (2)

$$\forall P_1\ P_2$$

$$\mathrm{traces}(P_1) = \mathrm{traces}(P_2)$$
$$\Rightarrow$$
$$\mathrm{traces}(P_1{\downarrow}) = \mathrm{traces}(P_2{\downarrow})$$

Show that for any program, its “traces” set does not change after translation

$$\forall P.\ \mathrm{traces}(P) = \mathrm{traces}(P{\downarrow})$$

i.e., need to show

$$\boxed{\forall P\ t.\ t \in \mathrm{traces}(P) \Longleftrightarrow t \in \mathrm{traces}(P{\downarrow})}$$

Ternary simulation to prove the theorem:

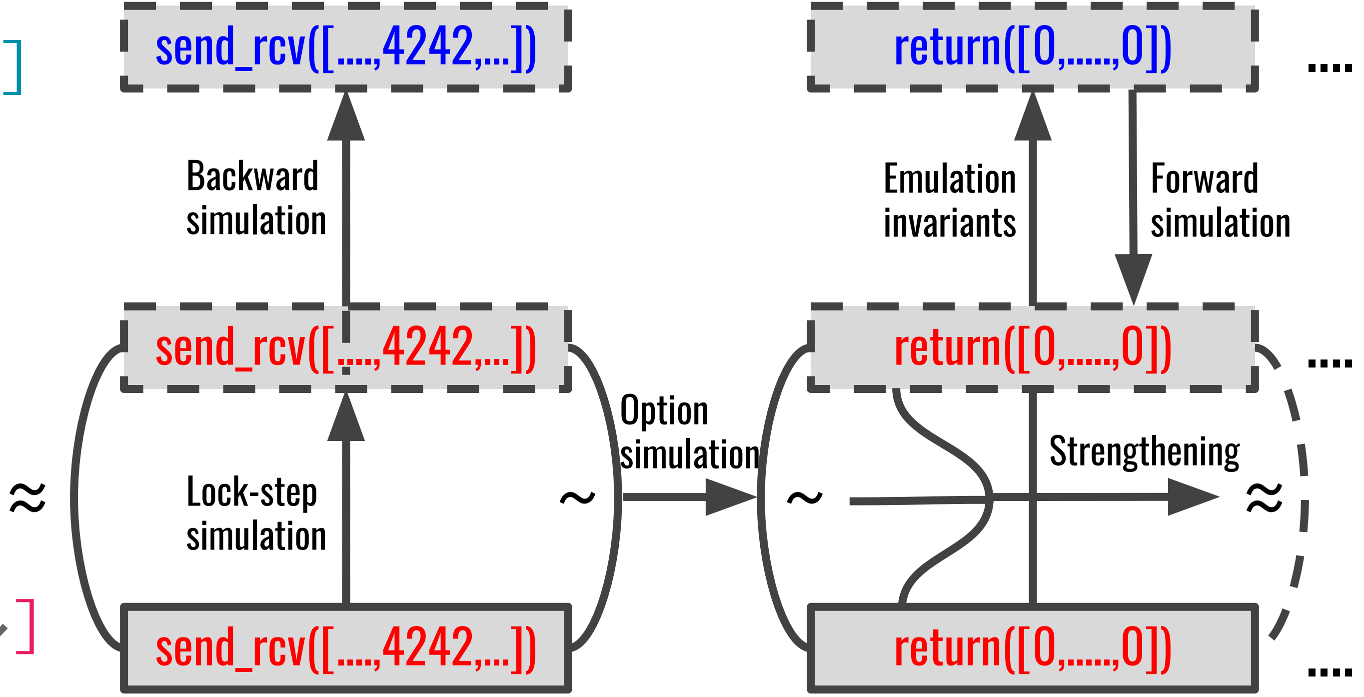
`∀P t. t ∈ traces(P) ⇐ t ∈ traces(P↓)`

Example to explain the use of a ternary cross-language simulation relation

$C_{emulating}[P]$

$C_{emulating}{\downarrow}[P{\downarrow}]$

$C_{obtained}[P{\downarrow}]$



Conclusion:

- Model a pointers-to-capabilities translation.
- Prove that it is fully abstract by using a ternary cross-language simulation for the proof of:

  `∀P t. t ∈ traces(P) ⇇ t ∈ traces(P↓).`

Thank you