Sharing Ghost Variables in a Collection of Abstract Domains

Marc Chevalier Jérôme Feret

DI ENS, INRIA, PSL, Paris, France

VMCAI 2020

Problems

In a Reduced Product

Discussion

Problems

In a Reduced Product

Discussion

A Big Stack

To certify software systems, we have to consider:

- End-user software (more classical)
- Libraries
- Runtime environment
- Operating system (here I work)
- Hypervisor
- Hardware
- Physics

The operating system layer is the place of terrible low level operations.

A Problem

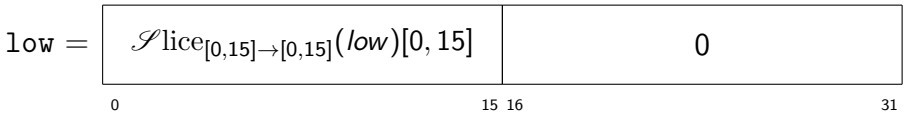
```
1  int something_interesting = ...;
2  int low = something_interesting & 0x0000ffff;
3  int high = something_interesting >> 16;
4  ... // complex computations
5  int rebuilt = low | high << 16;
```

(With enough assumptions:)

$$\texttt{something\_interesting} = \texttt{rebuilt}$$

Ghost Variables to the Rescue

```
1  int something_interesting = ...;
2  int low = something_interesting & 0x0000ffff;
3  ...
```



where

- $\mathscr{S}\mathrm{lice}_{[0,15]\to[0,15]}(low)$ is a ghost variable.
- $\mathscr{S}\mathrm{lice}_{[0,15]\to[0,15]}(low)$:= something_interesting

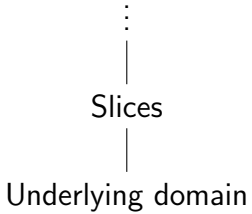
Another Problem

```
1  int something_interesting_2 = ..., noise = ...;
2  int noisy = something_interesting_2 + noise;
3  ...
4  int clean = noisy - noise;
```

(With enough assumptions:)

$$\texttt{something\_interesting\_2} = \texttt{clean}$$

Building a Domain with Ghost Variables



Theoretically:
- Parametric domain.
- Slices_D

Implementation:
- Dependency injection.
- (Objects, templates, functors...)

A Bigger Problem

Slices of linear combinations:

```
1  int* p = ...;
2  int* q = ...;
3  int* r = p + q;
4  int l = r & 0xffff;
5  int h = r << 16;
6  ... // kill r and p
7  int* r2 = l | h >> 16;
8  int* p2 = r2 - q;
```

Linear combinations of slices:

```
1  int* p = ...;
2  int* n = ...;
3  int* n2 = ....;
4  int l = p & 0xffff;
5  int h = p << 16;
6  int a = l + n;
7  int b = h + n2;
8  ... // kill p, l and h
9  int l2 = a - n;
10 int h2 = b - n2;
11 int* p2 = l2 | h2 >> 16;
```

(It happens... really)

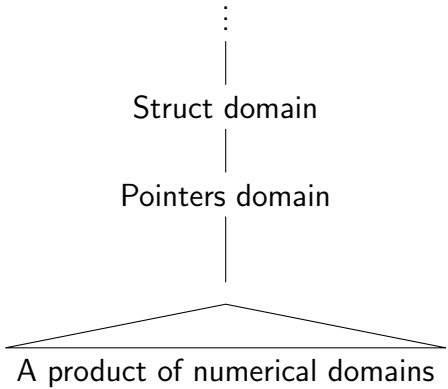
Thus any domain should be aware of everybody helping variables.

Problems

In a Reduced Product

Discussion

Astrée on the Inside



Downsides

- Variables ids are handled by Struct domain: missing ids for ghost variables.
- There is no way to add another Pointer-like domain.

Astrée on the Inside – Reduced Product

Given $(D_1^\sharp, \subseteq_1^\sharp), (D_2^\sharp, \subseteq_2^\sharp)$, abstract domains for the same concrete domain.

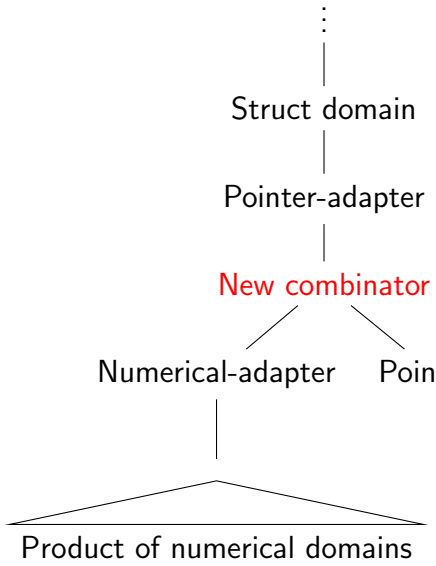
Product: $D_{1\times 2}^\sharp = D_1^\sharp \times D_2^\sharp$ with pointwise operations.
$\gamma_{1\times 2}(a1, a2) = \gamma_1(a_1) \cap \gamma_2(a_2)$

$\rho(a_1, a_2) = (b_1, b_2)$ with

$$\gamma_{1\times 2}(a1, a2) \subseteq \gamma_{1\times 2}(b1, b2) \qquad \text{(sound)}$$

$$\text{Morally:} \quad b_1 \subseteq_1^\sharp a_1 \wedge b_2 \subseteq_2^\sharp a_2 \qquad \text{(better)}$$

Astrée on the Inside



New combinator for pointers domains:

- Id translation by Pointer-adapter.
- Can add domains for pointer slices, linear combinations. . . .
- Cleaner interfaces.
- Each domain can ask everybody to store a ghost variable and do computations on it.

The New Product in Action

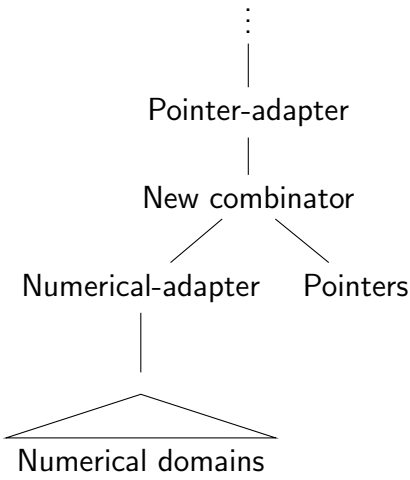
```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

Before line 3:

p points to a, with 0 offset.

Offset stored in ghost variable $o_{\mathtt{p}}$.

- Numerical: $o_{\mathtt{p}} = 0$
- Pointers: $\mathtt{p} = \mathtt{a} + o_{\mathtt{p}}$



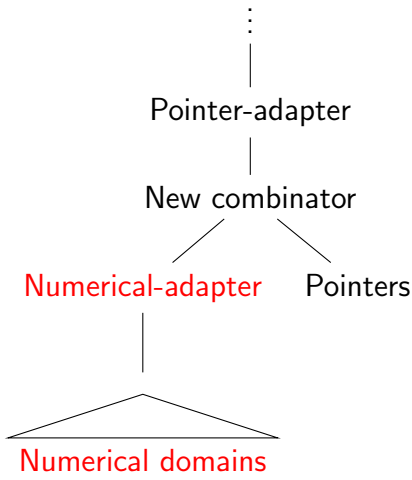
The New Product in Action

```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

$q \leftarrow p + 1$:

In numerical domains

$$p = \top \Downarrow q = \top$$



The New Product in Action

```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

$q \leftarrow p + 1$:

In pointer domain

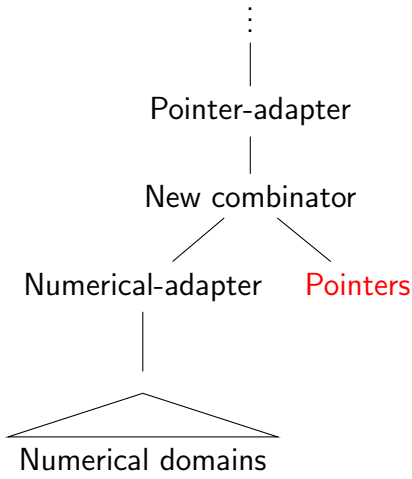
$$p = a + o_p$$
$$\Downarrow$$

Need new offset o_q

and according to the semantics:

$o_q \leftarrow o_p + 1 \times 4$

context



The New Product in Action

```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

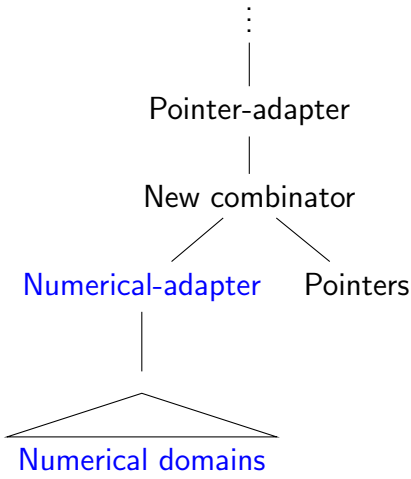
$o_q \leftarrow o_p + 1 \times 4$:

In numerical domains

$o_p = 0$

$\Downarrow$

$o_q = 4$



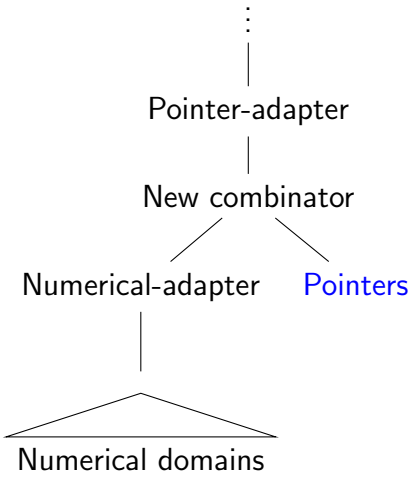
The New Product in Action

```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

$o_q \leftarrow o_p + 1 \times 4$:

In pointer domain

$$o_p \in \mathrm{NUM}$$
$$\Downarrow$$
$$o_q \in \mathrm{NUM}$$



The New Product in Action

```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

$q \leftarrow p + 1$:

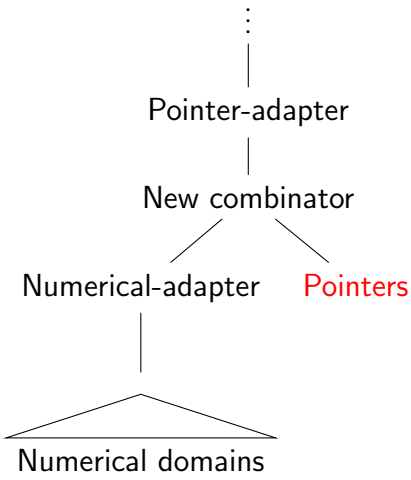
In pointer domain

$p = a + o_p$

context

$\Downarrow$

$q = a + o_q$

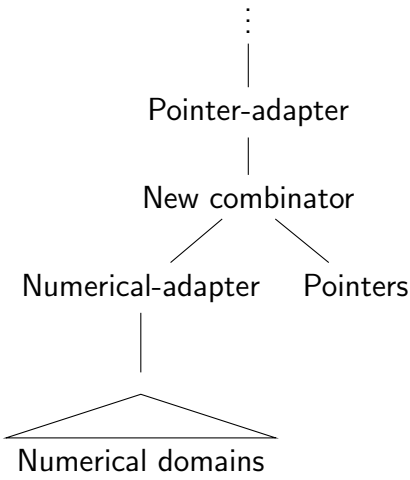


The New Product in Action

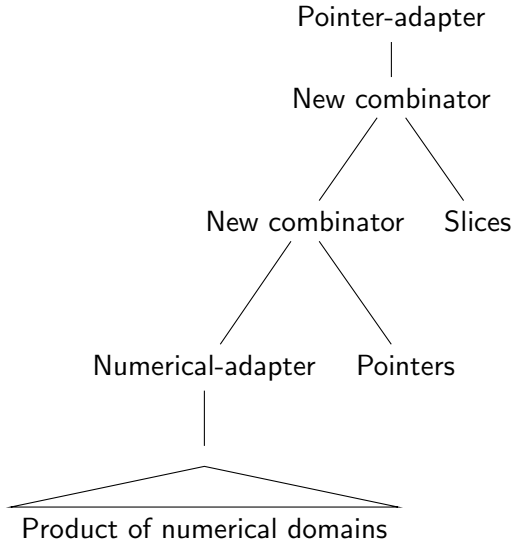
```
1  int a[4], *p, *q;
2  p = &a[0];
3  q = p + 1;
```

After line 3:

- $\mathtt{p} = \mathtt{a} + o_{\mathtt{p}}$
- $\mathtt{q} = \mathtt{a} + o_{\mathtt{q}}$
- $o_{\mathtt{p}} = 0$
- $o_{\mathtt{q}} = 4$



More?


Pointer-adapter
New combinator
New combinator
Slices
Numerical-adapter
Pointers
Product of numerical domains

Some Legitimate Concerns: Soundness

Is it correct to suspend a computation to do something else?

If they are independent enough, computations commute.

$$\Downarrow$$

Generate computations on ghostlier variables.

$$\text{e.g. } \mathrm{q} \leftarrow \mathrm{p} + 1 \qquad \Rightarrow \qquad o_{\mathrm{q}} \leftarrow o_{\mathrm{p}} + 1 \times 4$$

Some Legitimate Concerns: Termination

Are we sure it terminates?

We consume complexity of expressions to create ghost variables

$\Downarrow$

Finite number of new variables

$\Downarrow$

Finite recursion depth.

Problems

In a Reduced Product

Discussion

Current Status

Astrée: 200k OCaml.

Pointer product: Astrée: +30k -14k;
Side effect: 10k OCaml, 10k Python

It works: successfully used on serious software by industrial partner.

Dialectic

+ More general
+ Solve my problem
+ Solve older problems
+ Remove some hacks
+ Cleaner code (more parametric, more abstraction)

- More internal instructions for each real one: slower
- Tricky to implement

And opportunistically: clean and optimize some old code I adapted.

How Slow?!

As of November 2019: 70 times slower.

How Slow?!

As of November 2019: 70 times slower.

As of December 2019: 3 times slower. Phew!

How Slow?!

As of November 2019: 70 times slower.

As of December 2019: 3 times slower. Phew!

Now: still 3 times slower, but we know why and what could be done.

How Slow?!

As of November 2019: 70 times slower.

As of December 2019: 3 times slower. Phew!

Now: still 3 times slower, but we know why and what could be done.

- More precision through more variables $\Rightarrow$ almost linear cost.
- Pointer-numeric adapter spend a lot of time to prepare binary operations. Can be improved, heavy work.
- Time spend in the product within experimental fluctuations.

What Now?

The backbone is there: product's advanced features, developer framework, profiling, debugging...

To do:

- new domains
- better adaptation of old domains
- some may benefit from extra advanced features to relax some invariants.