Relational Abstractions Based on Labeled Union-Find

D. Lesbre, M. Lemerre, H. R. Ait-El-Hara, F. Bobot

PLDI – June 20th, 2025 – Seoul

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```

0 4

Many applications

Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```

Many applications

Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```

Many applications

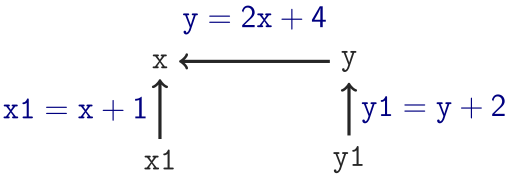
Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```



Many applications

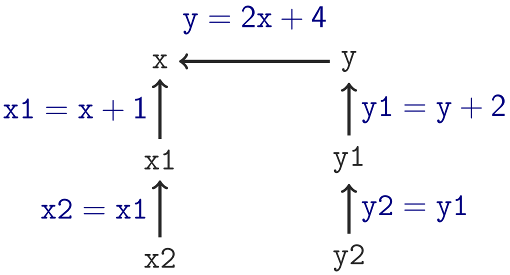
Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```



Many applications

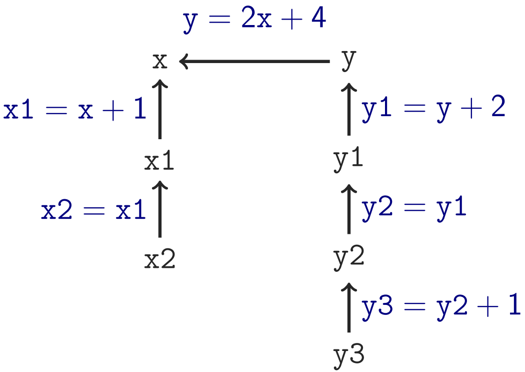
Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```



Many applications

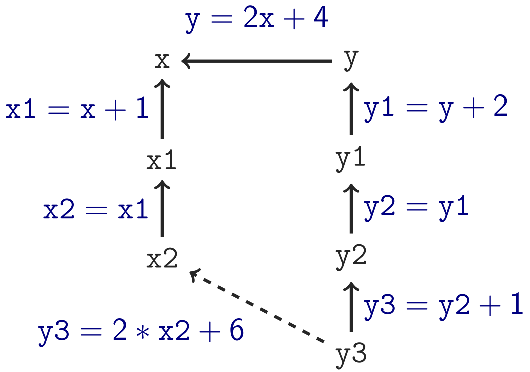
Static analysis, SMT solvers, datalog engines, e-graphs...

What is this talk about

```
init:
  loop(0, 4);

loop(x,y):
  x1 = x + 1;
  y1 = y + 2;
  if(...) loop(x1,y1)
  else    exit(x1,y1);

exit(x2,y2):
  y3 := y2 + 1;
  assert(2 * x2 + 6 == y3);
```

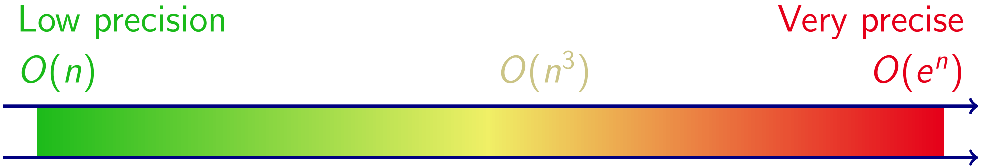


Many applications

Static analysis, SMT solvers, datalog engines, e-graphs...

Context

Program abstractions usually fall on a cost/precision spectrum:



Context

Program abstractions usually fall on a cost/precision spectrum:

Low precision — Very precise

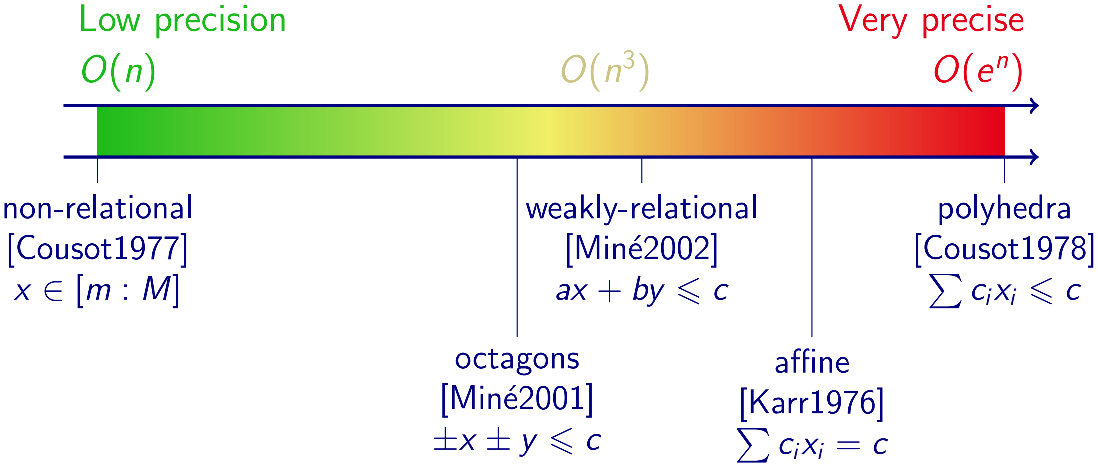
$O(n)$ — $O(n^3)$ — $O(e^n)$

non-relational
[Cousot1977]
$x \in [m : M]$

polyhedra
[Cousot1978]
$\sum c_i x_i \leqslant c$

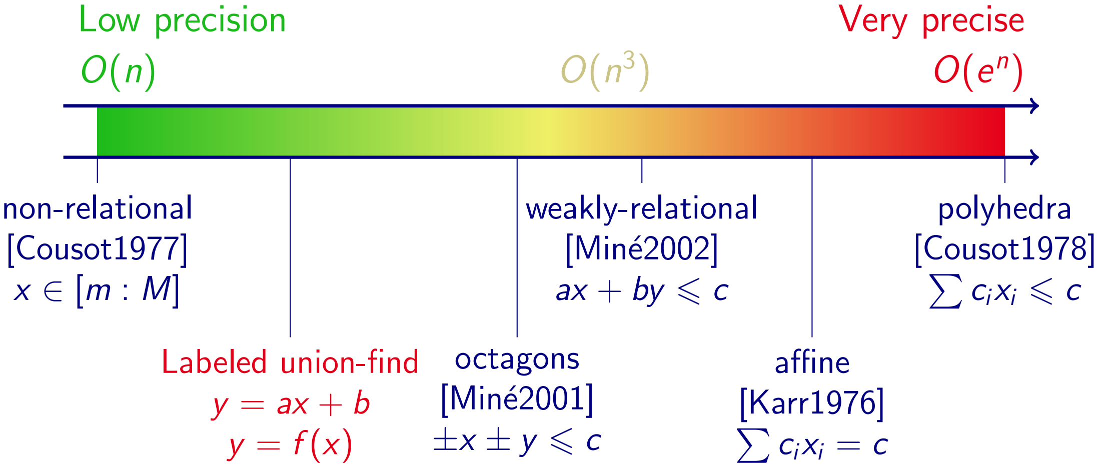
Context

Program abstractions usually fall on a cost/precision spectrum:



Context

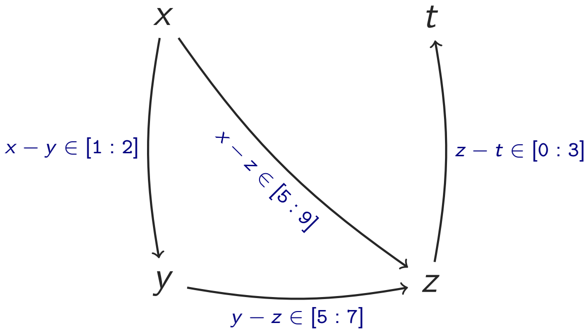
Program abstractions usually fall on a cost/precision spectrum:



1. Why is labeled union-find faster than other domains?

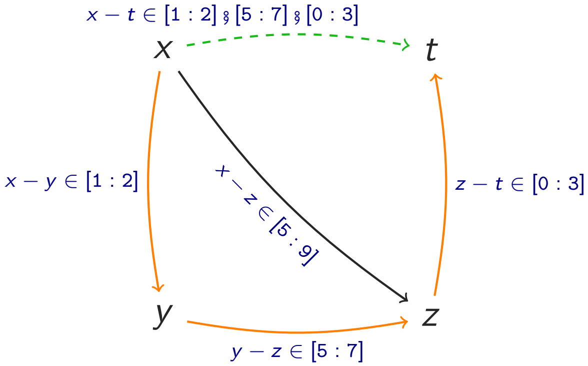
Why are (weakly) relational domains costly?

- Propagate constraints to improve precision
- Transitive closure is costly to compute - $\mathcal{O}(n^3)$



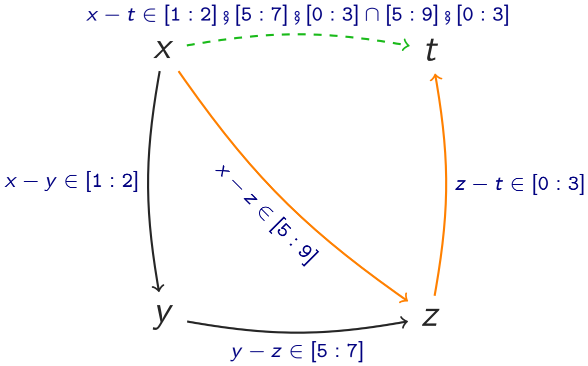
Why are (weakly) relational domains costly?

- Propagate constraints to improve precision
- Transitive closure is costly to compute - $\mathcal{O}(n^3)$



Why are (weakly) relational domains costly?

- Propagate constraints to improve precision
- Transitive closure is costly to compute - $\mathcal{O}(n^3)$



Why are (weakly) relational domains costly?

- Propagate constraints to improve precision
- Transitive closure is costly to compute - $\mathcal{O}(n^3)$

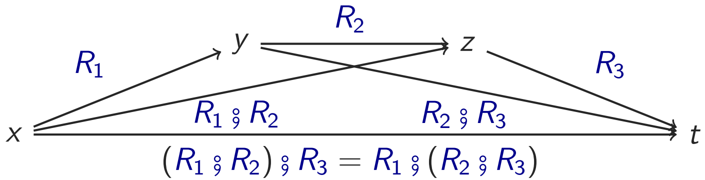
Unique relation assumption

Same relation on every path between two variables

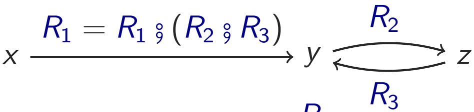
$\Rightarrow$ Transitive closure can be computed on a single path, we only need to maintain a spanning tree

Consequences of the unique relation assumption

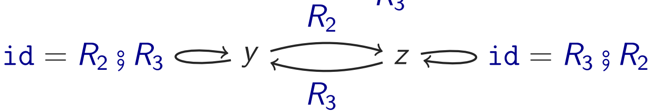
Associativity:



Neutral elements:

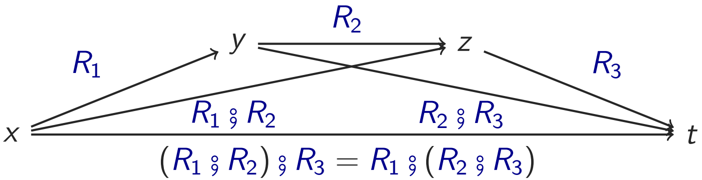


Inverse elements:

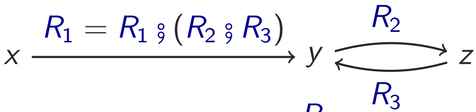


Consequences of the unique relation assumption

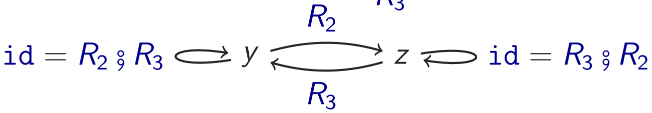
Associativity:



Neutral elements:



Inverse elements:



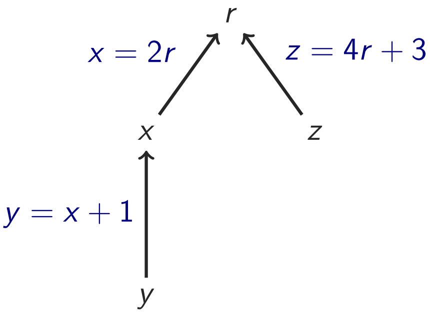
Follow-up assumption

Relations must have a group structure

2. What is labeled union-find?

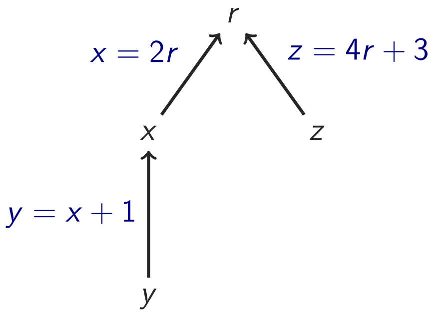
Labeled union-find example

Labeled union-find adds labels to edges of union-find



Labeled union-find example

Labeled union-find adds labels to edges of union-find

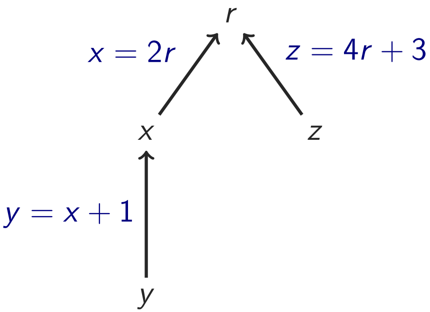


`find` returns representative and relation

- $\texttt{find}(r) = (r, id)$

Labeled union-find example

Labeled union-find adds labels to edges of union-find

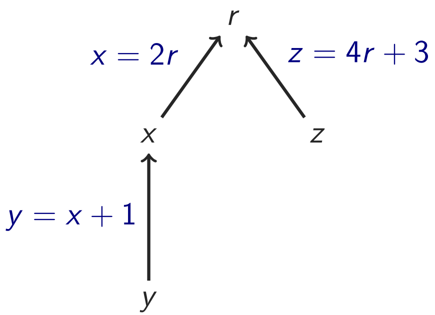


`find` returns representative and relation

- $\texttt{find}(r) = (r, id)$
- $\texttt{find}(x) = (r, x = 2r)$

Labeled union-find example

Labeled union-find adds labels to edges of union-find

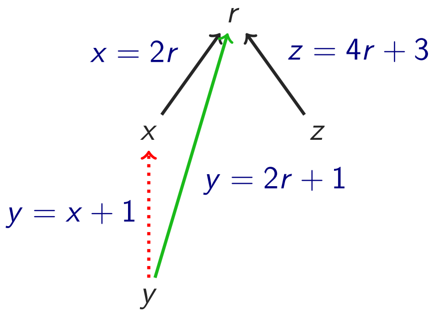


`find` returns representative and relation

- $\texttt{find}(r) = (r, id)$
- $\texttt{find}(x) = (r, x = 2r)$
- $\texttt{find}(y) = (r, y = x + 1 \mathbin{;} x = 2r)$
  $= (r, y = 2r + 1)$

Labeled union-find example

Labeled union-find adds labels to edges of union-find

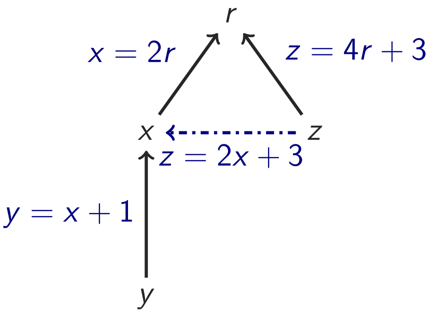


find returns representative and relation

- $\texttt{find}(r) = (r, id)$
- $\texttt{find}(x) = (r, x = 2r)$
- $\texttt{find}(y) = (r, y = x + 1 \fatsemi x = 2r)$
  $= (r, y = 2r + 1)$

Labeled union-find example

Labeled union-find adds labels to edges of union-find

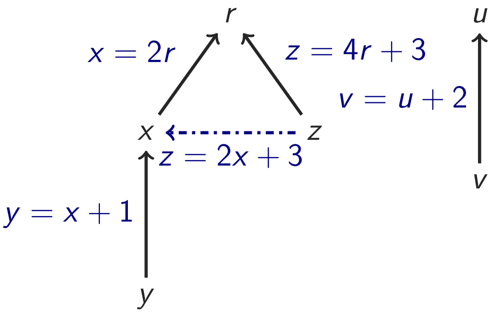


get_relation: finds the relation between two variables

- get_relation(z, x)
  $= z = 4r - 1 \mathbin{;} (x = 2r)^{-1}$
  $= z = 2x - 1$

Labeled union-find example

Labeled union-find adds labels to edges of union-find

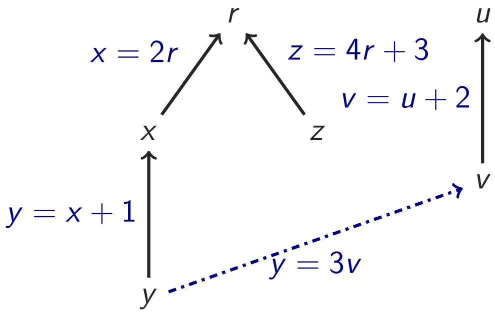


`get_relation`: finds the relation between two variables

- $\texttt{get\_relation}(z, x)$
  $= z = 4r - 1 \mathbin{;} (x = 2r)^{-1}$
  $= z = 2x - 1$
- $\texttt{get\_relation}(z, u) = \texttt{None}$

Labeled union-find example

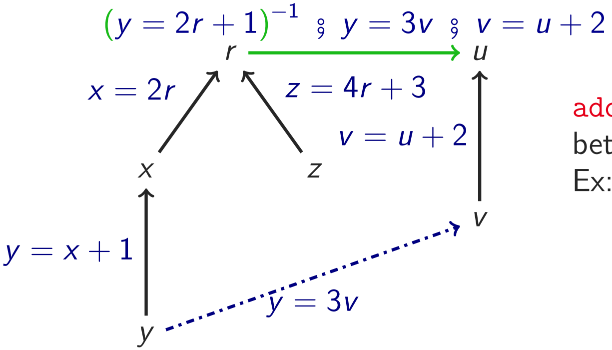
Labeled union-find adds labels to edges of union-find



`add_relation`: adds a new relation between variables
Ex: `add_relation`$(y, v, y = 3v)$

Labeled union-find example

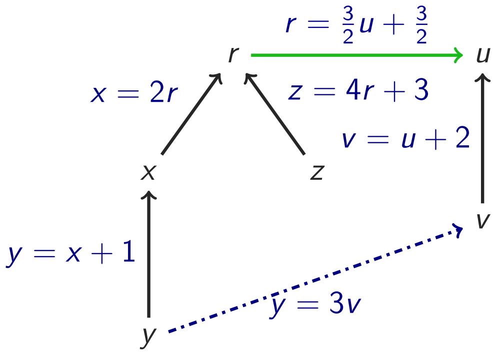
Labeled union-find adds labels to edges of union-find



add_relation: adds a new relation between variables
Ex: add_relation($y, v, y = 3v$)

Labeled union-find example

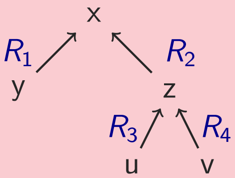
Labeled union-find adds labels to edges of union-find



`add_relation`: adds a new relation between variables
Ex: `add_relation`$(y, v, y = 3v)$

Labeled union-find data structure

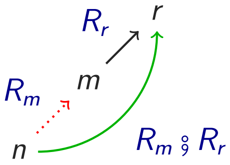
Labeled union-find



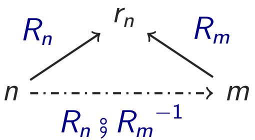
Rooted forest with group operations on labels:

Identity	$\texttt{id} : \mathcal{R}$
Inverse	$\cdot^{-1} : \mathcal{R} \to \mathcal{R}$
Compose	$\cdot \fatsemi \cdot : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$

$\texttt{find}(n)$:

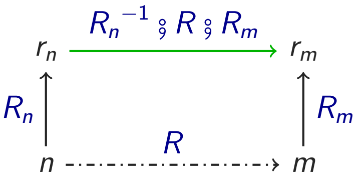


$\texttt{get\_relation}(n, m)$:



or None if $r_n \neq r_m$

$\texttt{add\_relation}(n, m, R)$:



3. What relations can be used with labeled union-find?

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions

~~Interval difference:~~ ~~$y = x + [m : M]$~~

Constant offset: $y = x + b$ for $b \in \mathbb{Z}$

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions

~~Interval difference:~~ ~~$y = x + [m : M]$~~

Constant offset: $y = x + b$ for $b \in \mathbb{Z}$

Two variable linear equality: $y = ax + b$ for $a, b \in \mathbb{Q}$ or $\mathbb{R}$, $a \neq 0$

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions

~~Interval difference:~~ ~~$y = x + [m : M]$~~

Constant offset: $y = x + b$ for $b \in \mathbb{Z}$

Two variable linear equality: $y = ax + b$ for $a, b \in \mathbb{Q}$ or $\mathbb{R}$, $a \neq 0$

Modulo multiplication: $y = ax + b \bmod 2^{64}$ for $a, b \in \mathbb{BV}_{64}$, a odd

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions

~~Interval difference: $y = x + [m : M]$~~

Constant offset: $y = x + b$ for $b \in \mathbb{Z}$

Two variable linear equality: $y = ax + b$ for $a, b \in \mathbb{Q}$ or $\mathbb{R}$, $a \neq 0$

Modulo multiplication: $y = ax + b \texttt{ mod } 2^{64}$ for $a, b \in \mathbb{BV}_{64}$, a odd

XOR-Rotate relation: $y = (x \texttt{ xor } c) \texttt{ rot } n$ for $c \in \mathbb{BV}_{64}$ and $n \in [0 : 63]$

Suitable abstract relations

Abstract relations $\mathcal{R}$ must both:

- Be a sound abstraction of relations $\gamma \in \mathcal{R} \to \mathcal{P}(\mathbb{V} \times \mathbb{V})$
- Have a group structure

Theorem

Suitable relations are injective functions (between equivalence classes)

~~Interval difference:~~ ~~$y = x + [m : M]$~~

Constant offset: $y = x + b$ for $b \in \mathbb{Z}$

Two variable linear equality: $y = ax + b$ for $a, b \in \mathbb{Q}$ or $\mathbb{R}$, $a \neq 0$

Modulo multiplication: $y = ax + b \texttt{ mod } 2^{64}$ for $a, b \in \mathbb{BV}_{64}$, a odd

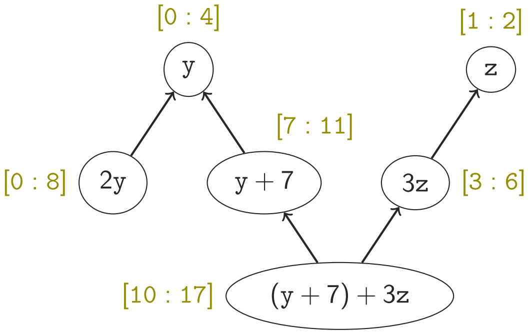
XOR-Rotate relation: $y = (x \texttt{ xor } c) \texttt{ rot } n$ for $c \in \mathbb{BV}_{64}$ and $n \in [0 : 63]$

Parity comparison: $x \texttt{ mod } 2 \bowtie y \texttt{ mod } 2$, for $\bowtie \in \{=, \neq\}$

4. How does labeled union-find combine with other domains?

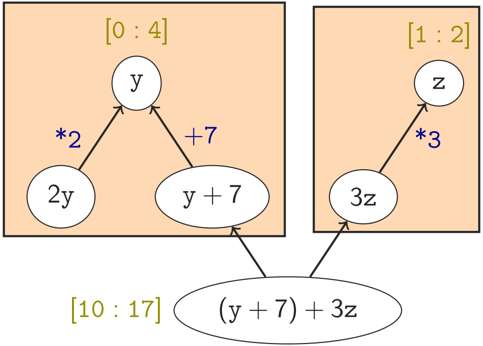
Factorizing numeric information

Redundant numeric information on terms:



Factorizing numeric information

Redundant numeric information on terms: only store one per relational class

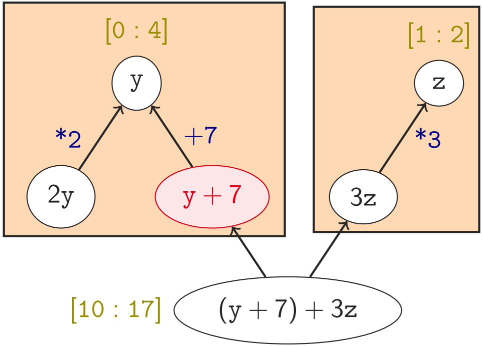


Factorizing numeric information

Redundant numeric information on terms: only store one per relational class

`get_value`($y + 7$):

- find representative
- $[0 : 4]$ via $+7$ is $[7 : 11]$



Factorizing numeric information

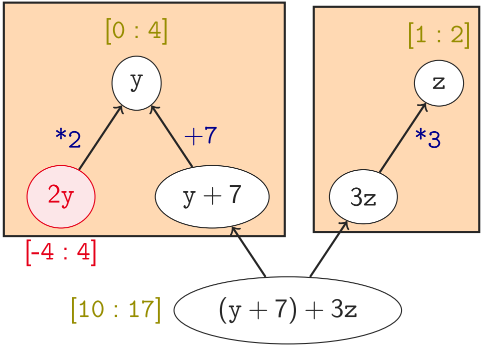
Redundant numeric information on terms: only store one per relational class

`get_value`($y + 7$):

- find representative
- [0 : 4] via +7 is [7 : 11]

`set_value`($2y$, [-4 : 4]) / `add_relation`:

- find representative
- [-4 : 4] via $(*2)^{-1}$ is [-2 : 2]



Factorizing numeric information

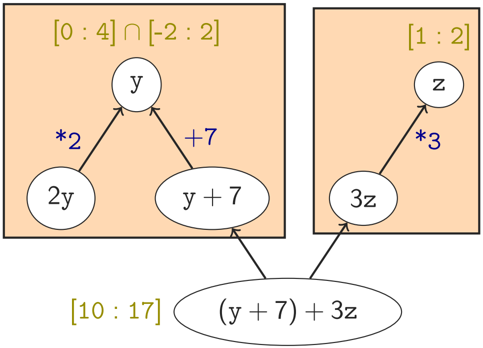
Redundant numeric information on terms: only store one per relational class

`get_value`$(y+7)$:

- find representative
- [0 : 4] via $+7$ is [7 : 11]

`set_value`$(2y,$ [-4 : 4]$)$ / `add_relation`:

- find representative
- [-4 : 4] via $(*2)^{-1}$ is [-2 : 2]
- intersect with old value



Factorizing numeric information

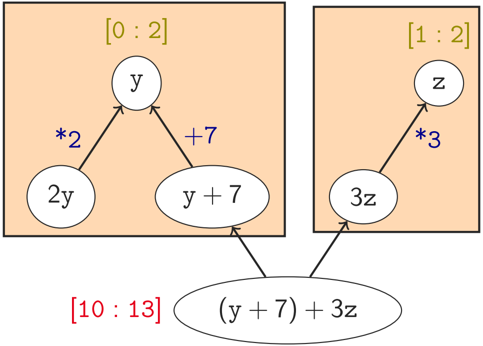
Redundant numeric information on terms: only store one per relational class

`get_value`$(y + 7)$:

- find representative
- $[0 : 4]$ via $+7$ is $[7 : 11]$

`set_value`$(2y, [-4 : 4])$ / `add_relation`:

- find representative
- $[-4 : 4]$ via $(*2)^{-1}$ is $[-2 : 2]$
- intersect with old value
- propagate to sub/superterms [PLDI'24]



Factorizing numeric information

Redundant numeric information on terms: only store one per relational class

Group action to update values via relations:

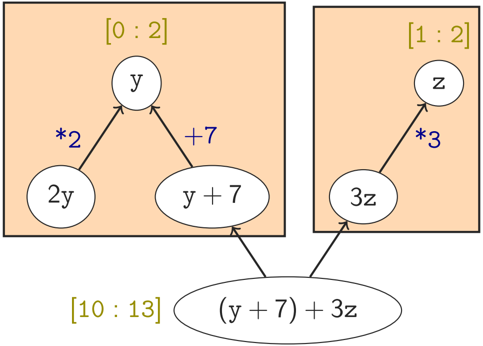
$$\mathcal{A} : \mathcal{R} \times \mathbb{I} \to \mathbb{I}$$

`get_value`$(y + 7)$:

- find representative
- $\mathcal{A}(+7, [0 : 4]) = [7 : 11]$

`set_value`$(2y, [\text{-}4 : 4])$ / `add_relation`:

- find representative
- $\mathcal{A}((*2)^{-1}, [\text{-}4 : 4]) = [\text{-}2 : 2]$
- intersect with old value
- propagate to sub/superterms [PLDI'24]



Exact actions

To avoid losing precision, actions must be exact:

- Constant offset with intervals is exact:

$$\mathcal{A}(y = x + b,\ [m : M]) \triangleq [m + b : M + b]$$

Exact actions

To avoid losing precision, actions must be exact:

- Constant offset with intervals is exact:

$$\mathcal{A}(y = x + b,\ [m : M]) \triangleq [m + b : M + b]$$

- TVPE with intervals is exact (on $\mathbb{R}$ or $\mathbb{Q}$):

$$\mathcal{A}(y = ax + b,\ [m : M]) \triangleq [am + b : aM + b]$$

Exact actions

To avoid losing precision, actions must be exact:

- Constant offset with intervals is exact:

$$\mathcal{A}(y = x + b,\ [m : M]) \triangleq [m + b : M + b]$$

- TVPE with intervals is exact (on $\mathbb{R}$ or $\mathbb{Q}$):

$$\mathcal{A}(y = ax + b,\ [m : M]) \triangleq [am + b : aM + b]$$

- No exact action for constant offset or TVPE with bitwise abstraction: $\mathcal{A}(+\texttt{0b011},\ \texttt{0b00?}) = \texttt{0b???}$.

Exact actions

To avoid losing precision, actions must be exact:

- Constant offset with intervals is exact:

$$\mathcal{A}(y = x + b,\ [m : M]) \triangleq [m + b : M + b]$$

- TVPE with intervals is exact (on $\mathbb{R}$ or $\mathbb{Q}$):

$$\mathcal{A}(y = ax + b,\ [m : M]) \triangleq [am + b : aM + b]$$

- No exact action for constant offset or TVPE with bitwise abstraction: $\mathcal{A}(+\texttt{0b011},\ \texttt{0b00?}) = \texttt{0b???}$.

- XOR-Rotate relation has an exact action with bitwise abstraction:

$$\mathcal{A}(y = (x \ \texttt{xor}\ c)\ \texttt{rot}\ k, b_1 \cdots b_n) \triangleq d_1 \cdots d_n$$
$$\text{with } d_i \triangleq b_{i+k} \text{ xor } c_{i+k}$$

  But inexact with intervals.

Other applications

- Factorize relational domains: only store relations between relational classes:

$$\mathcal{A}(y = x + b, z + x \leqslant c) \triangleq z + y \leqslant c - b$$

Other applications

- Factorize relational domains: only store relations between relational classes:

$$\mathcal{A}(y = x + b, z + x \leqslant c) \triangleq z + y \leqslant c - b$$

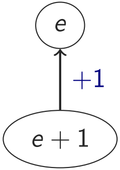
- Discover all equalities, i.e. all (x, y) such that $x \xrightarrow{\text{id}} y$

5. Can we use an imperative structure in an abstract interpreter?

Only add flow insensitive relations

Flow insensitive relations: true no matter where in the program.
Especially common when running the analysis on an SSA form [PLDI'24].

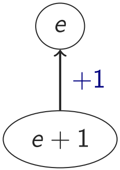
When building terms:



Only add flow insensitive relations

Flow insensitive relations: true no matter where in the program.
Especially common when running the analysis on an SSA form [PLDI'24].

When building terms:



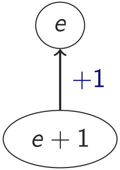
When joining (SSA ϕ-terms):

- If $a \xrightarrow{+1} b$ and $a' \xrightarrow{+1} b'$ then $\phi(a, a') \xrightarrow{+1} \phi(b, b')$

Only add flow insensitive relations

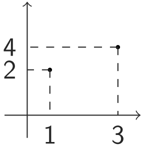
Flow insensitive relations: true no matter where in the program.
Especially common when running the analysis on an SSA form [PLDI'24].

When building terms:



When joining (SSA ϕ-terms):

- If $a \xrightarrow{+1} b$ and $a' \xrightarrow{+1} b'$ then $\phi(a, a') \xrightarrow{+1} \phi(b, b')$
- ϕ of constant pairs in TVPE:

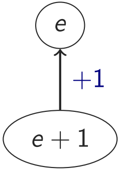


$\phi(2, 4)$ $\phi(1, 3)$

Only add flow insensitive relations

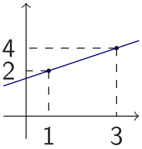
Flow insensitive relations: true no matter where in the program.
Especially common when running the analysis on an SSA form [PLDI'24].

When building terms:



When joining (SSA ϕ-terms):

- If $a \xrightarrow{+1} b$ and $a' \xrightarrow{+1} b'$ then $\phi(a, a') \xrightarrow{+1} \phi(b, b')$
- ϕ of constant pairs in TVPE:



$\phi(2, 4) \xrightarrow{*\frac{1}{3}+\frac{5}{3}} \phi(1, 3)$

Example: relating loop counters

```
int i = 0, j = 4;
while(i < N) {
    i += 1;
    j += 3;
}
```

Example: relating loop counters

```
int i = 0, j = 4;
while(i < N) {
    i += 1;
    j += 3;
}
```

Non-relational: $i \in [N; N]$, $j \in [4; +\infty], j = 1 \mod 3$

Example: relating loop counters

```
int i = 0, j = 4;
while(i < N) {
    i += 1;
    j += 3;
}
```

Non-relational: $i \in [N; N]$, $j \in [4; +\infty], j = 1 \mod 3$

With union-find: $j = 3i + 4$ so $j \in [3N + 4 : 3N + 4]$.

Conclusion

Labeled union-find data structure:

- Extension of union-find with edge labels
- Labels must have a group structure

Labeled union-find domain:

- Fast weakly relational domain (easy transitive closure)
- Only stores injective relations
- Can combine with other domains to store information per class

Implemented as part of the Codex static analysis library (`https://codex.top`) and the Colibri2 solver (https://colibri.frama-c.com).

Postdoc positions: matthieu.lemerre@cea.fr