Ranking Abstractions

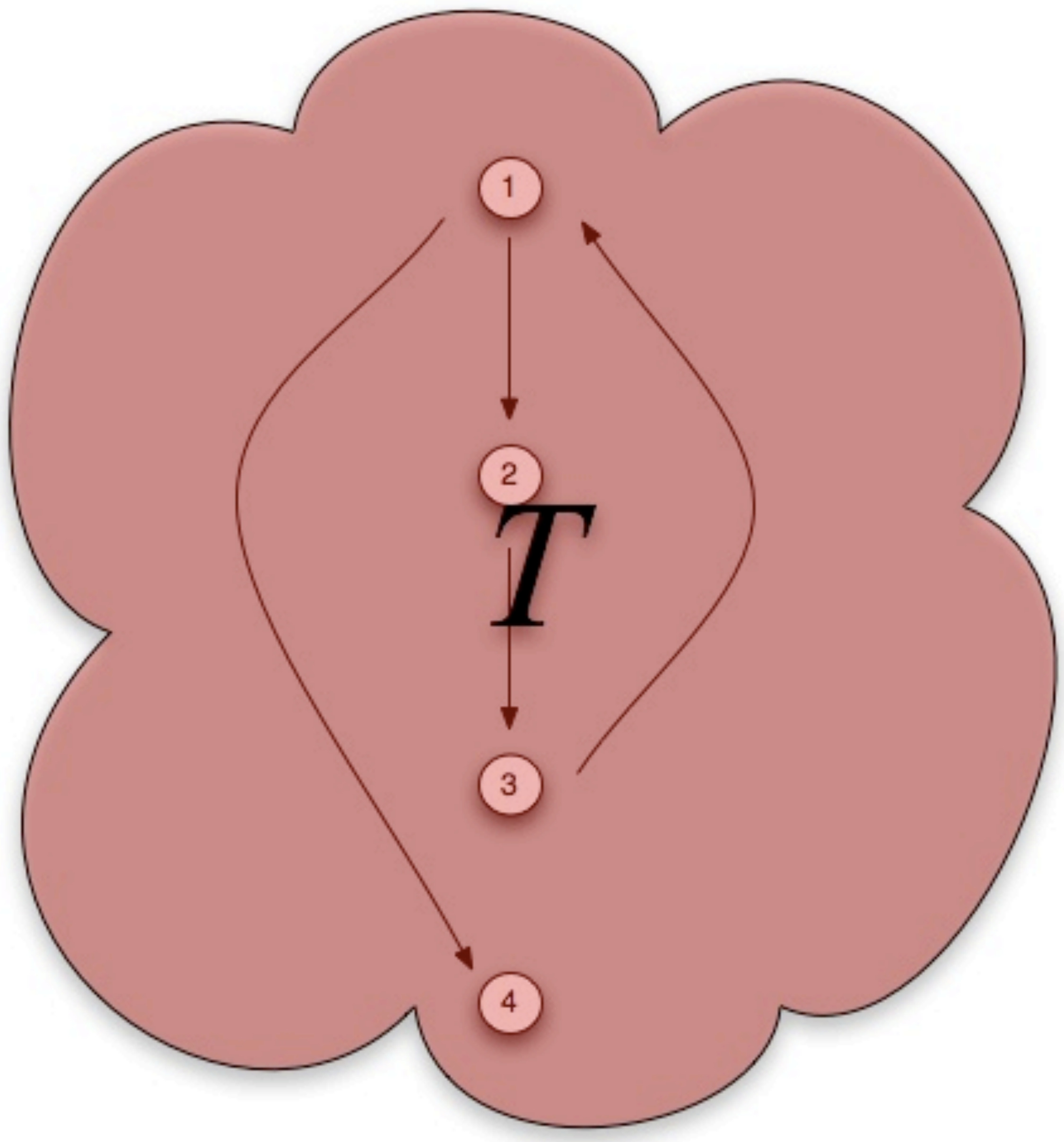
Aziem Chawdhary, Byron Cook, Sumit Gulwani,
Mooly Sagiv and Hongseok Yang

Why prove Termination?

- Verification of Reactive Code
- If I call a function will it always return?
- If I request a resource will I eventually get access to it?
- Liveness properties

Proving Termination

The Traditional Method



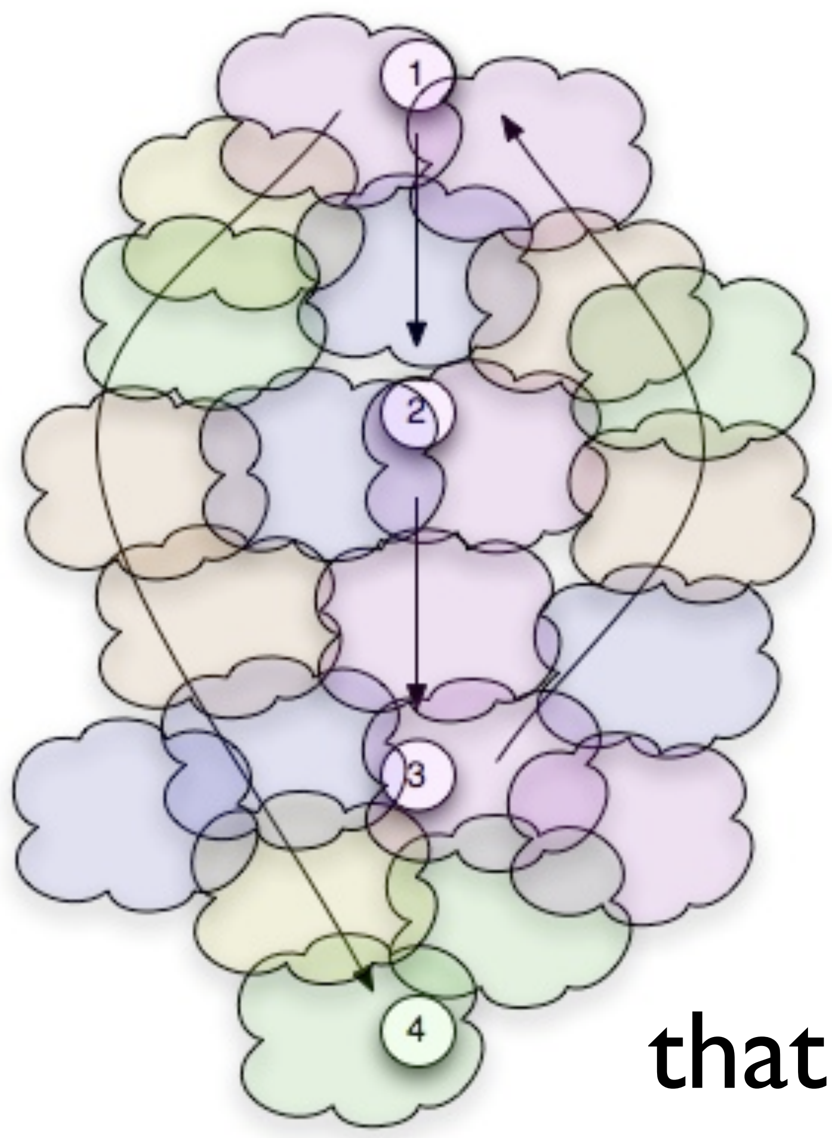
$$\text{Function } T \text{ s.t}$$

$$T(\sigma) > T(\sigma')$$

A function f with a range to a well-ordered set is called a ranking function

Proving Termination

Podelski-Rybalchenko Result



$$R \text{ is well-founded } iff$$

$$R^+ \subseteq T_1 \cup \ldots \cup T_n$$

$$T_i \text{ is well-founded}$$

WF Relation is a relation
that does not allow infinite sequences

Previous Work

$$R^+ \subseteq T_1 \cup ... \cup T_n$$

- Variance Analyses from Invariance Analyses (POPL 07) Josh Berdine, Aziem Chawdhary, Byron Cook, Dino Distefano, Peter O'Hearn
- Take a safety analyser and reuse its results to produce a Termination Analyser
- Use RFS to generate covering relations
- Inclusion check is guaranteed, but WF is not

Essence

- Disjunction of Ranking Relations per program point
- Symbolically Execute Path
  - Leaves Domain of Ranking Relations
  - Call RFS to squish back to Ranking Relations (Cousot, Abstraction)
- RFS guarantees fixpoint convergence and keeps only termination-relevant info

Example

Example

$\texttt{while}\ (x{>}0 \wedge y{>}0)\ \{$

$\quad\quad \texttt{if}\ (*)\ \texttt{then}\ \{\ x{=}x{-}1; y{=}*;\ \}$

$\quad\quad \texttt{else}\ \{\ y{=}y{-}1;\ \}$

$\}$

Example

$$C_1 \stackrel{\text{def}}{=} \text{`}x > 0 \wedge \text{`}y > 0 \wedge x{=}\text{`}x{-}1$$

```
while (
      if (*) then { x=x−1; y=*; }
      else { y=y−1; }
}
```

Example

$$C_1 \stackrel{\text{def}}{=} \text{`}x > 0 \wedge \text{`}y > 0 \wedge x{=}\text{`}x{-}1$$

```
while (
      if (*) then { x=x-1; y=*; }
      else { y=y-1; }
}
```

$$C_2 \stackrel{\text{def}}{=} \text{`}x > 0 \wedge \text{`}y > 0 \wedge x{=}\text{`}x \wedge y{=}\text{`}y{-}1.$$

Example

$$C_1 \stackrel{\text{def}}{=} {}^{\text{`}}x > 0 \wedge {}^{\text{`}}y > 0 \wedge x = {}^{\text{`}}x - 1$$

```
while (x ... y
      ...en { x=x−1; y=*; }
      ...=y−1; }
}
```

Perform RFS

$$C_2 \stackrel{\text{def}}{=} {}^{\text{`}}x > 0 \wedge {}^{\text{`}}y > 0 \wedge x = {}^{\text{`}}x \wedge y = {}^{\text{`}}y - 1.$$

Example

RFS$(C_1) = x$

```
while (...
   ...en { x=x-1; y=*; }
   ...=y-1; }
}
```

Perform RFS

RFS$(C_2) = y$

Example

Initial Abstract State

```
while (x>0 ∧ y>0) {
    if (*) then { x=x−1; y=*; }
    else { y=y−1; }
}
```

$$A_0 \;=\; (`x \geq 0 \wedge `x{-}1 \geq x) \vee (`y \geq 0 \wedge `y{-}1 \geq y \wedge `x{=}x).$$

$$A_0 \;=\; (`x \geq 0 \wedge `x{-}1 \geq x) \vee (`y \geq 0 \wedge `y{-}1 \geq y \wedge `x{=}x)$$

Symb Execute and RFS

$$
\begin{array}{l}
\texttt{while}\,(x{>}0 \wedge y{>}0)\ \{ \\
\quad\quad \texttt{if}\ (*)\ \texttt{then}\ \{\, x{=}x{-}1;\, y{=}*;\ \} \\
\quad\quad \texttt{else}\ \{\, y{=}y{-}1;\ \} \\
\}
\end{array}
$$

$$A_0 \quad = \quad ({}^{\backprime}x \geq 0 \wedge {}^{\backprime}x{-}1 \geq x) \vee ({}^{\backprime}y \geq 0 \wedge {}^{\backprime}y{-}1 \geq y \wedge {}^{\backprime}x{=}x)$$

Symb Execute and RFS

```
while (x>0 ∧ y>0) {
    if (*) then { x=x−1; y=*; }
    else { y=y−1; }
}
```

$$\mathsf{RFS}(({}^{\backprime}x \geq 0 \wedge {}^{\backprime}x+1 \geq x); C_1) \quad = \quad {}^{\backprime}x \geq 0 \wedge {}^{\backprime}x+1 \geq x$$

$$A_0 \;=\; ({}^{`}x \geq 0 \wedge {}^{`}x{-}1 \geq x) \vee ({}^{`}y \geq 0 \wedge {}^{`}y{-}1 \geq y \wedge {}^{`}x{=}x)$$

Symb Execute and RFS

$$
\begin{array}{l}
\texttt{while}\ (x{>}0 \wedge y{>}0)\ \{ \\
\quad\quad \texttt{if}\ (*)\ \texttt{then}\ \{\, x{=}x{-}1;\, y{=}*;\ \} \\
\quad\quad \texttt{else}\ \{\, y{=}y{-}1;\ \} \\
\}
\end{array}
$$

$$\mathsf{RFS}(({}^{`}x \geq 0 \wedge {}^{`}x + 1 \geq x); C_2) \;=\; {}^{`}x \geq 0 \wedge {}^{`}x + 1 \geq x$$

$A_0 \quad = \quad (`x \geq 0 \land `x{-}1 \geq x) \lor (`y \geq 0 \land `y{-}1 \geq y \land `x{=}x)$

Symb Execute and RFS

```
while (x>0 ∧ y>0) {
    if (*) then { x=x−1; y=*; }
    else { y=y−1; }
}
```

RFS$((`y \geq 0 \land `y + 1 \geq y \land `x = x); C_1) \quad = \quad `x \geq 0 \land `x + 1 \geq x$

$A_0 \;=\; ({‘x} \geq 0 \wedge {‘x}-1 \geq x) \vee ({‘y} \geq 0 \wedge {‘y}-1 \geq y \wedge {‘x}=x)$

Symb Execute and RFS

```
while (x>0 ∧ y>0) {
    if (*) then { x=x−1; y=*; }
    else { y=y−1; }
}
```

$\mathsf{RFS}(({‘y} \geq 0 \wedge {‘y}+1 \geq y \wedge {‘x} = x); C_2) \;=\; {‘y} \geq 0 \wedge {‘y}+1 \geq y$

Next Abstract State

ample

$$A_0$$

$$\vee$$

$$(`x \geq 0 \wedge `x{-}1 \geq x)$$

$$\vee$$

$$(`x \geq 0 \wedge `x{-}1 \geq x)$$

$$\vee$$

$$(`x \geq 0 \wedge `x{-}1 \geq x)$$

$$\vee$$

$$(`y \geq 0 \wedge `y{-}1 \geq y)$$

```
w
                        y=*; }

}
```

Next Abstract State

...ample

Reached a fixpoint

A_0

1 wh...

2 ... $y = *;$ }

3

4 }

Next Abstract State

...ample

```
w...
    ... y=*; }
    ...
}
```

Reached a fixpoint

A_0

Example

$$
\begin{array}{l}
\texttt{while}\ (x{>}0 \land y{>}0)\ \{ \\
\quad\quad \texttt{if}\ (*)\ \texttt{then}\ \{\, x{=}x{-}1;\, y{=}*;\ \} \\
\quad\quad \texttt{else}\ \{\, y{=}y{-}1;\ \} \\
\}
\end{array}
$$

$$A_0 \;=\; ({`}x \geq 0 \land {`}x{-}1 \geq x) \lor ({`}y \geq 0 \land {`}y{-}1 \geq y \land {`}x{=}x).$$

Found a disjunctively WF relation overapproximating the program

ple

$$y>0)\ \{$$

$$\texttt{if}\ (*)\ \texttt{then}\ \{\, x{=}x{-}1;\, y{=}*;\, \}$$

$$\texttt{else}\ \{\, y{=}y{-}1;\, \}$$

$$\}$$

$$A_0 \;=\; (`x \geq 0 \wedge `x{-}1 \geq x) \vee (`y \geq 0 \wedge `y{-}1 \geq y \wedge `x{=}x).$$

ple

Found a disjunctively WF relation overapproximating the program

Therefore it Terminates!

$\wedge\ y{>}0)$ {

    `if` $(*)$ `then` $\{\, x{=}x{-}1;\ y{=}*;\ \}$

    `else` $\{\, y{=}y{-}1;\ \}$

}

$$A_0 \;=\; (`x \geq 0 \wedge `x{-}1 \geq x) \vee (`y \geq 0 \wedge `y{-}1 \geq y \wedge `x{=}x).$$

Some Formal Stuff

Programming Language

$$
\begin{array}{rcl}
e & ::= & x \mid r \mid e + e \mid r \times e \\
b & ::= & e{=}e \mid e{\neq}e \mid b \wedge b \mid b \vee b \mid \neg b \\
a & ::= & x{:=}e \mid x{:=}{*} \mid \texttt{assume}(b) \\
c & ::= & a \mid c; c \mid \texttt{while}\, b\; c \mid c \,[]\, c
\end{array}
$$

Abstract Domain

- Analysis is parameterised by a domain representing relations on program states

$$\text{a set } D$$

A Monotone Concretization Function

$$\gamma_r : D \to \mathcal{P}(\mathsf{St} \times \mathsf{St})$$

Abstract Domain

Parameterised Domain

An Abstract Identity Element over-approximating identity relation on states

$$\mathsf{id}_{\mathsf{St}} \quad \subseteq \quad \gamma_r(d_{\mathsf{id}})$$

Abstract Domain

Parameterised Domain

$$\mathrm{RFS}\colon D \to \mathcal{P}_{\mathsf{fin}}(D) \uplus \{\top\}$$

RFS Returns an Over-Approximation

$$\mathsf{RFS}(d) \neq \top \implies \gamma_r(d) \subseteq \bigcup \{\gamma_r(d') \mid d' \in \mathsf{RFS}(d)\}.$$

Abstract Domain

Parameterised Domain

$$\mathrm{RFS}\colon D \to \mathcal{P}_{\mathsf{fin}}(D) \uplus \{\top\}$$

RFS Returns WF relations

$$\mathsf{RFS}(d) \neq \top \implies \bigcup \{\gamma_r(d') \mid d' \in \mathsf{RFS}(d)\} \text{ is well-founded.}$$

Abstract Domain

Parameterised Domain

$$\mathsf{trans}(a) \colon D \to \mathcal{P}_{\mathsf{fin}}(D)$$

Trans overapproximates relational meaning of commands

$$\forall d \in D.\ \ (\gamma_r(d); [\![a]\!]) \ \subseteq\ \bigcup \{\gamma_r(d') \mid d' \in \mathsf{trans}(a)(d)\}$$

Abstract Domain

Parameter Domain

$$\mathsf{comp}\colon D \times D \to D$$

comp over-approximates relational composition

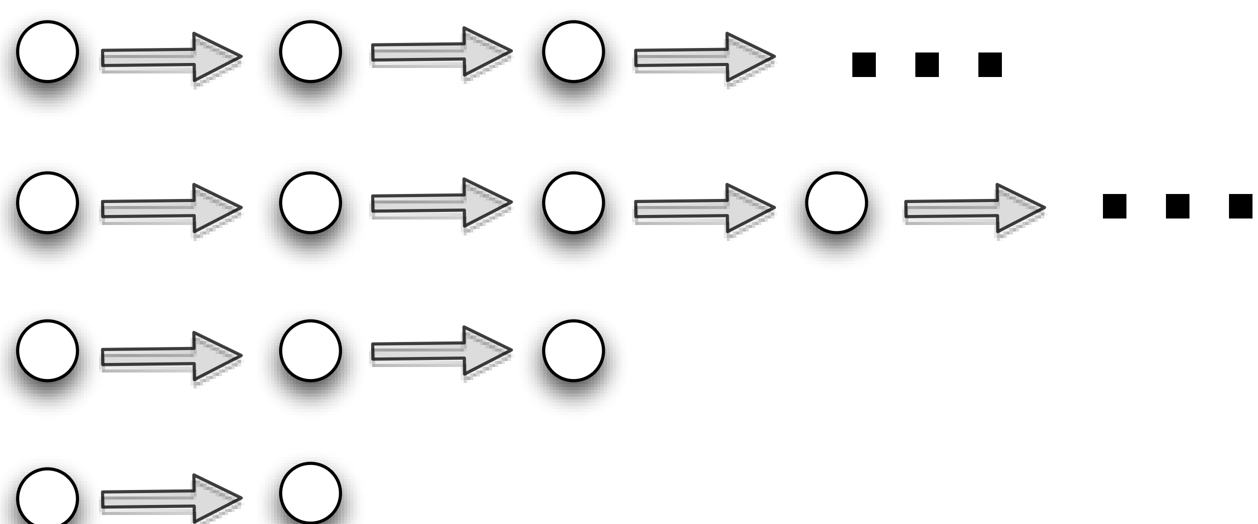
$$\gamma_r(d); \gamma_r(d') \quad \subseteq \quad \gamma_r(\mathsf{comp}(d, d')).$$

Abstract Domain

$$\mathcal{A} \stackrel{\text{def}}{=} (\mathcal{P}_{\mathsf{fin}}(D))^{\top} \quad (\text{i.e., } \mathcal{P}(D) \uplus \{\top\}).$$

$\top$

represents all possible traces incl infinite traces

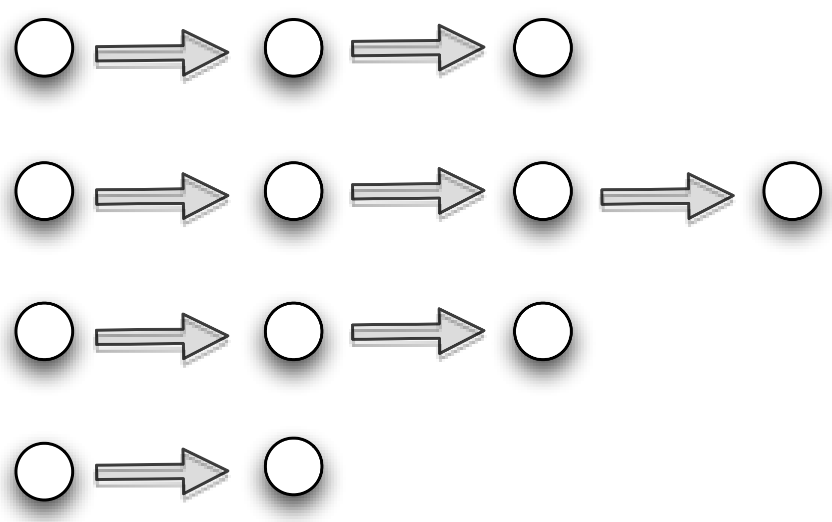


Abstract Domain

$$\mathcal{A} \stackrel{\text{def}}{=} (\mathcal{P}_{\mathsf{fin}}(D))^{\top} \quad (\text{i.e., } \mathcal{P}(D) \uplus \{\top\}).$$

non - $\top$

represents sets of finite non-empty traces



The Analysis

$$
\begin{aligned}
[\![c]\!]^{\#} \;&:\; \mathcal{A} \to \mathcal{A} \\
[\![a]\!]^{\#} A &\stackrel{\mathrm{def}}{=} (\mathsf{trans}(a))^{\dagger} A \\
[\![c_0 ; c_1]\!]^{\#} A &\stackrel{\mathrm{def}}{=} ([\![c_1]\!]^{\#} \circ [\![c_0]\!]^{\#}) A \\
[\![c_0 \,[]\, c_1]\!]^{\#} A &\stackrel{\mathrm{def}}{=} [\![c_0]\!]^{\#} A \sqcup [\![c_1]\!]^{\#} A
\end{aligned}
$$

Fixpoints

$$[\![\texttt{while} * c]\!]^{\#} A \stackrel{\text{def}}{=} \textbf{let } F \stackrel{\text{def}}{=} \lambda A'.[\![c]\!]^{\#}(\{d_{\mathsf{id}}\}) \sqcup [\![c]\!]^{\#}(A') \\ \textbf{in } \left(\mathsf{comp}^{\dagger}(A, \mathsf{fix}\ (\mathsf{RFS}^{\dagger} \circ F))\right)$$

- Overapproximate R^{+}not R^{*}
- Start from id relation rather than input A
- Apply RFS at each iteration

Linear Constraints Domain

- Produced an implementation
- Abstract elements consist of only linear constraints between variables
- Use Rankfinder by Andrey Rybalchenko

Experiments

Experiments

Examples from Octagon Domain Distribution

	1		2		3		4		5		6	
LR	0.01	✓	0.01	✓	0.08	✓	0.09	✓	0.02	✓	0.06	✓
O	0.11	✓	0.08	✓	6.03	✓	1.02	✓	0.16	✓	0.76	✓
P	1.40	✓	1.30	✓	10.90	✓	2.12	✓	1.80	✓	1.89	✓
T	6.31	✓	4.93	✓	T/O	-	T/O	-	33.24	✓	3.98	✓

LR Linear Ranking Domain Implementation

O Octagon Domain Variance Analyses from POPL07

P Polyhedra Domain Variance Analyses from POPL07

T Terminator

Experiments

Examples from Windows Device Drivers

	1		2		3		4		5		6		7		8		9		10	
LR	0.23	✓	0.20	⊘	0.00	⊘	0.04	✓	0.00	✓	0.03	✓	0.07	✓	0.03	✓	0.01	⊘	0.03	✓
O	1.42	✓	1.67	⊘	0.47	⊘	0.18	✓	0.06	✓	0.53	✓	0.50	✓	0.32	✓	0.14	⊘	0.17	✓
P	4.66	✓	6.35	⊘	1.48	⊘	1.10	✓	1.30	✓	1.60	✓	2.65	✓	1.89	✓	2.42	⊘	1.27	✓
T	10.22	✓	31.51	⊘	20.65	⊘	4.05	✓	12.63	✓	67.11	✓	298.45	✓	444.78	✓	T/O	-	55.28	✓

LR Linear Ranking Domain Implementation

O Octagon Domain Variance Analyses from POPL07

P Polyhedra Domain Variance Analyses from POPL07

T Terminator

Experiments

Examples from Polyrank Distribution

	1		2		3		4		6		7		8		9		10		11		12	
LR	0.19	✓	0.02	✓	0.01	†	0.02	†	0.02	†	0.01	†	0.04	†	0.01	†	0.03	†	0.02	†	0.01	†
O	0.30	†	0.05	†	0.11	†	0.50	†	0.10	†	0.17	†	0.16	†	0.12	†	0.35	†	0.86	†	0.12	†
P	1.42	✓	0.82	✓	1.06	†	2.29	†	2.61	†	1.28	†	0.24	†	1.36	✓	1.69	†	1.56	†	1.05	†
T	435.23	✓	61.15	✓	T/O	-	T/O	-	75.33	✓	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-	10.31	†

LR Linear Ranking Domain Implementation

O Octagon Domain Variance Analyses from POPL07

P Polyhedra Domain Variance Analyses from POPL07

T Terminator

Conclusions

- Designing an Abstract Domain with Termination in mind can give big increases in speed without sacrificing precision
- Caveat: Need a cheap safety analyser in a pre-phase