Automatic Inference of Ranking Functions by Abstract Interpretation

Caterina Urban

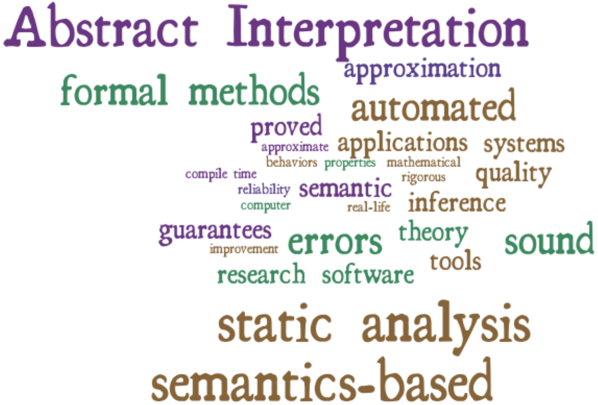
Project-Team ANTIQUE
INRIA Paris-Rocquencourt

Département d'Informatique
École Normale Supérieure

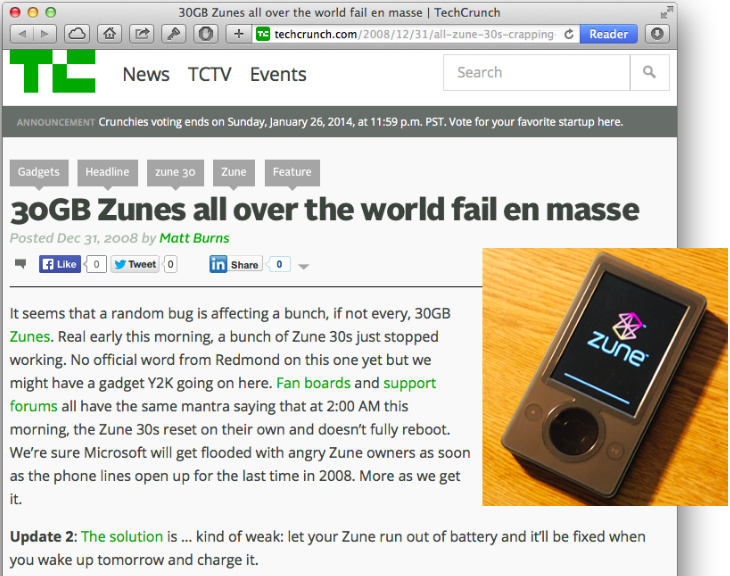
18th March 2014
INRIA Junior Seminar
INRIA Paris-Rocquencourt, France

Project-Team ANTIQUE

ANalyse Sta**TIQUE** par Interprétation Abstraite



Proving Program Termination? Why?



30GB Zunes all over the world fail en masse | TechCrunch

techcrunch.com/2008/12/31/all-zune-30s-crapping

Reader

News TCTV Events

Search

ANNOUNCEMENT Crunchies voting ends on Sunday, January 26, 2014, at 11:59 p.m. PST. Vote for your favorite startup here.

Gadgets Headline zune 30 Zune Feature

30GB Zunes all over the world fail en masse

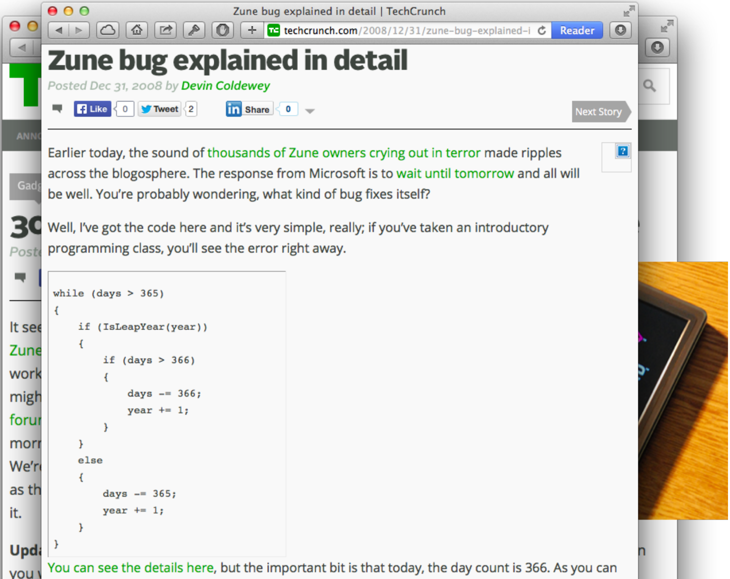
Posted Dec 31, 2008 by Matt Burns

Like 0 Tweet 0 Share 0

It seems that a random bug is affecting a bunch, if not every, 30GB Zunes. Real early this morning, a bunch of Zune 30s just stopped working. No official word from Redmond on this one yet but we might have a gadget Y2K going on here. Fan boards and support forums all have the same mantra saying that at 2:00 AM this morning, the Zune 30s reset on their own and doesn't fully reboot. We're sure Microsoft will get flooded with angry Zune owners as soon as the phone lines open up for the last time in 2008. More as we get it.

Update 2: The solution is ... kind of weak: let your Zune run out of battery and it'll be fixed when you wake up tomorrow and charge it.

Proving Program Termination? Why?


Zune bug explained in detail | TechCrunch

techcrunch.com/2008/12/31/zune-bug-explained-i

Zune bug explained in detail

Posted Dec 31, 2008 by Devin Coldewey

Like 0 Tweet 2 Share 0 Next Story

Earlier today, the sound of thousands of Zune owners crying out in terror made ripples across the blogosphere. The response from Microsoft is to wait until tomorrow and all will be well. You're probably wondering, what kind of bug fixes itself?

Well, I've got the code here and it's very simple, really; if you've taken an introductory programming class, you'll see the error right away.

```
while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

You can see the details here, but the important bit is that today, the day count is 366. As you can

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below
- **idea**: computation of ranking functions by abstract interpretation[2]
- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **affine ranking functions**[3]

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below

- **idea**: computation of ranking functions by abstract interpretation[2]

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **affine ranking functions**[3]

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below

- **idea**: computation of ranking functions by abstract interpretation[2]

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **affine ranking functions**[3]

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below
- **idea**: computation of ranking functions by abstract interpretation[2]

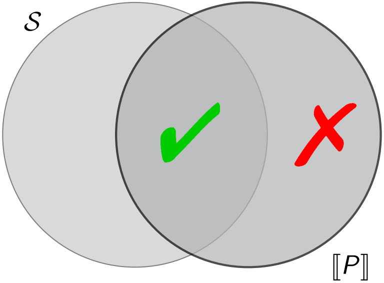
- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **affine ranking functions**[3]

---

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

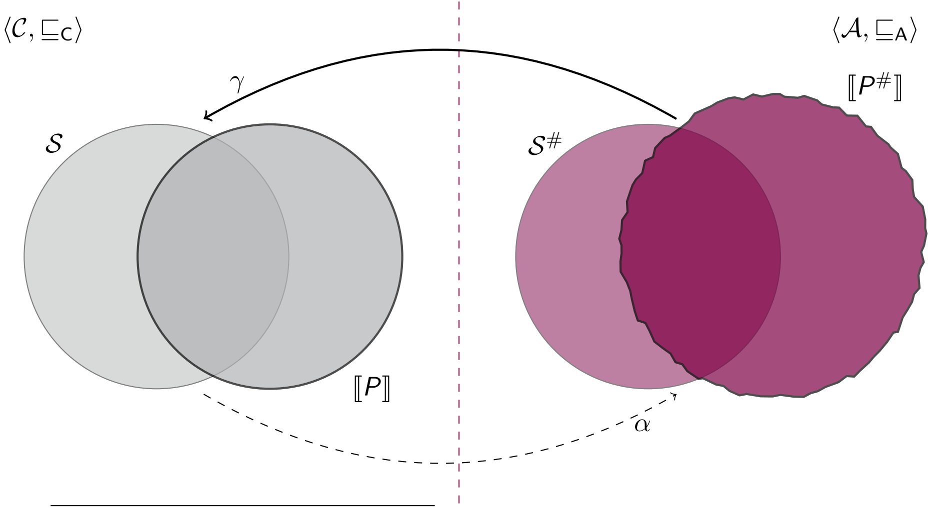
Abstract Interpretation[4]

$\langle \mathcal{C}, \sqsubseteq_\mathsf{C} \rangle$



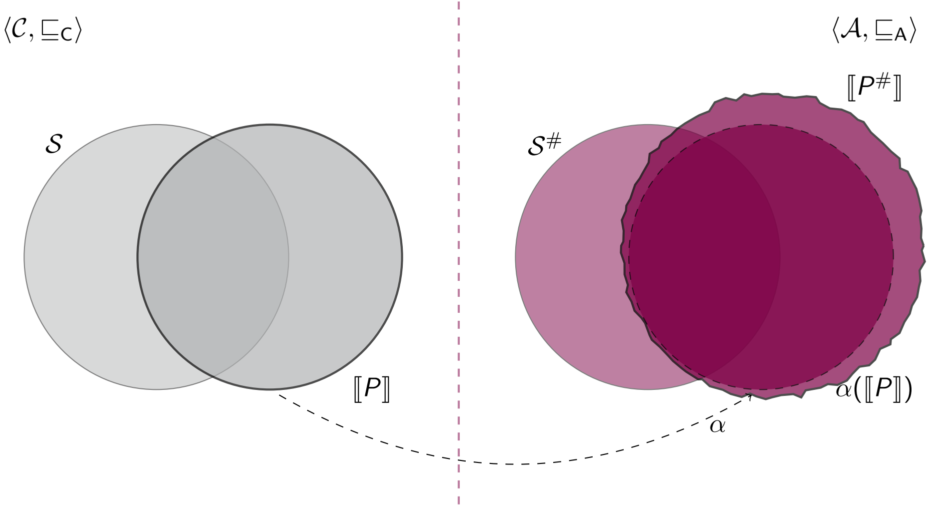
[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

Abstract Interpretation[4]



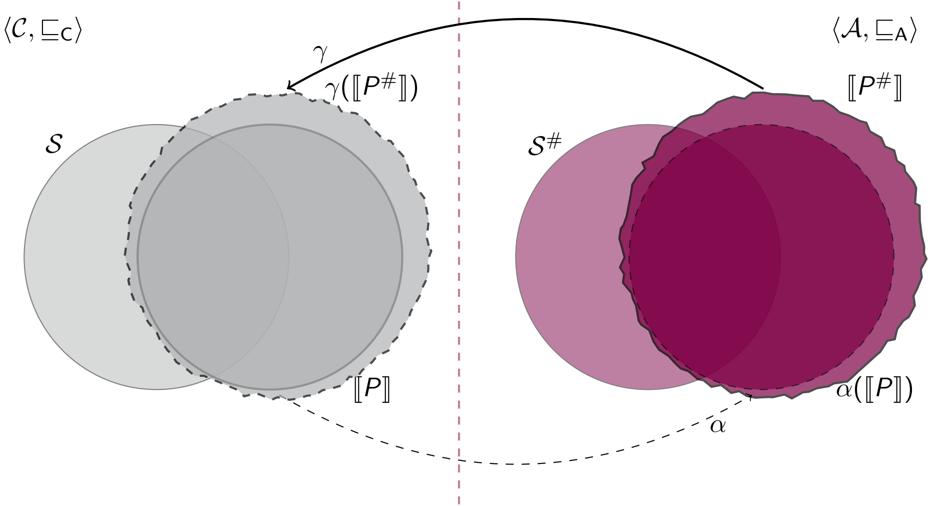
[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

Abstract Interpretation[4]



[4]Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

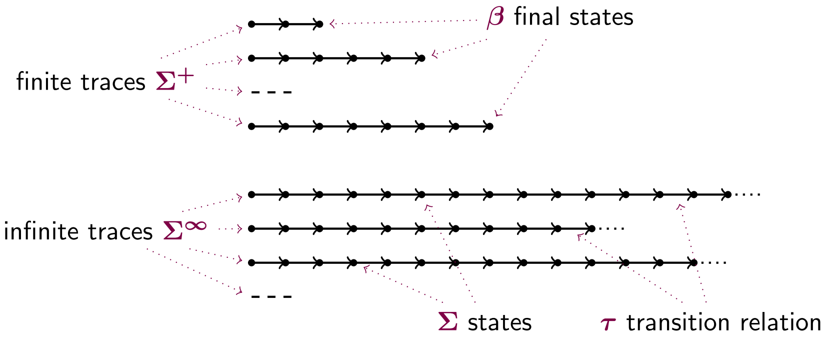
Abstract Interpretation[4]



[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

Concrete Semantics

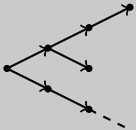
program $\mapsto$ **trace semantics**



program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

idea = define a ranking function
that **counts the number of program steps**
from the end of the program

Example



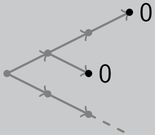
Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete**
to prove the termination of programs

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

Example



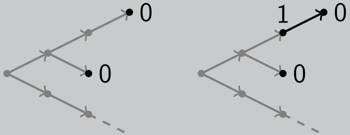
Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete**
to prove the termination of programs

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

Example



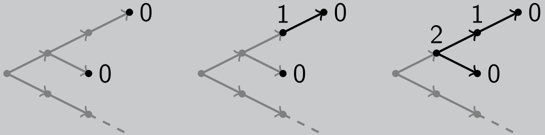
Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

Example



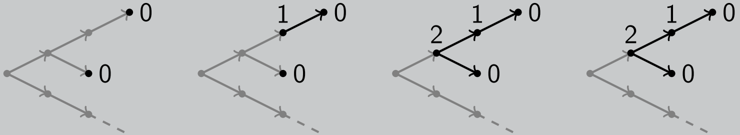
Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

Example



Theorem (Soundness and Completeness)

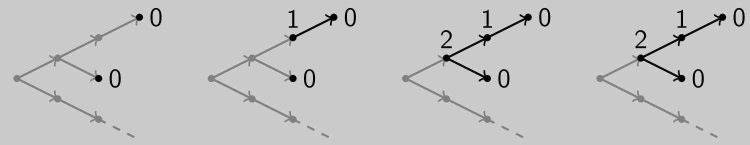
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

the termination semantics **extracts the well-founded part** of the program transition relation

Example



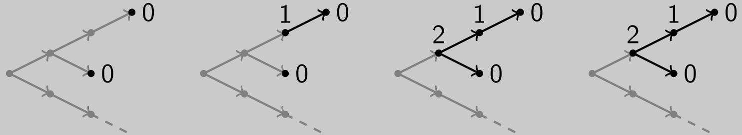
Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ trace semantics $\mapsto$ **termination semantics**

Example



Theorem (Soundness and Completeness)

the termination semantics is **sound** *and* **complete**
to prove the termination of programs

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

Example

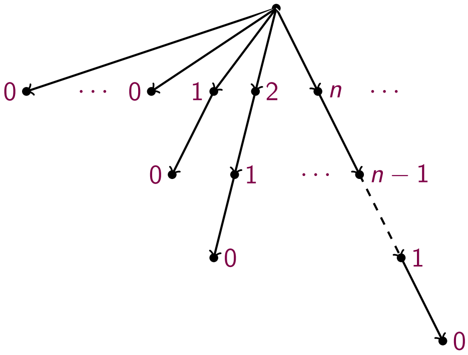
int : x

$x :=\ ?$

while $(x \geq 0)$ do

  $x := x - 1$

od



Example

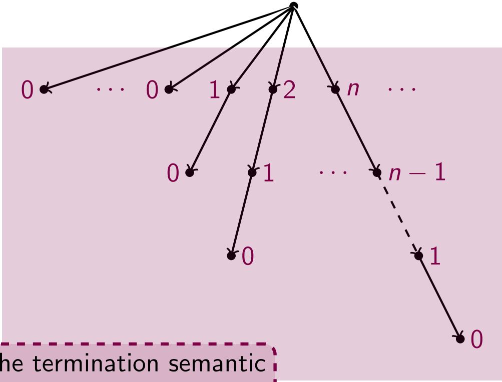
int : x

$x :=\ ?$

while $(x \geq 0)$ do

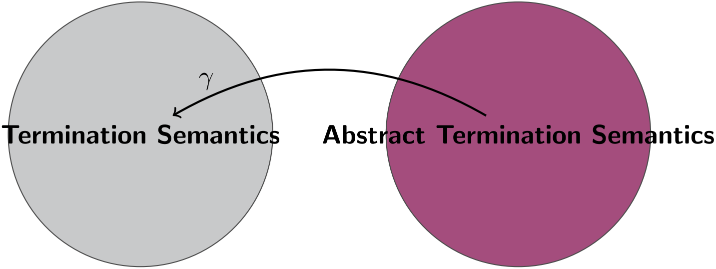
$x := x - 1$

od

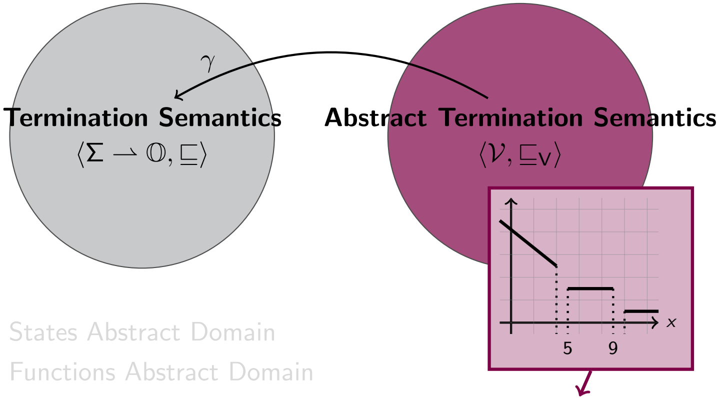


the termination semantic is **not computable**!

Piecewise-Defined Ranking Functions



- States Abstract Domain $\mathcal{S}$
- Functions Abstract Domain $\mathcal{F}$
- Piecewise-Defined Ranking Functions Abstract Domain $\mathcal{V}(\mathcal{S}, \mathcal{F})$



- States Abstract Domain — S
- Functions Abstract Domain — F
- Piecewise-Defined Ranking Functions Abstract Domain — V(S, F)

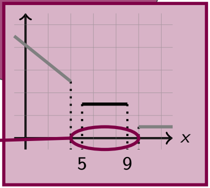
γ

Termination Semantics

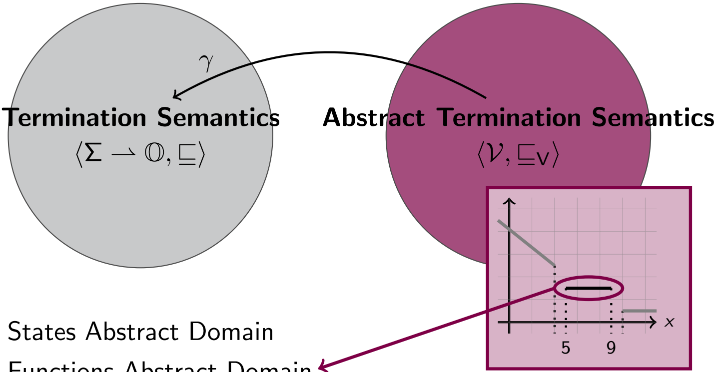
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq_\mathsf{V} \rangle$



- States Abstract Domain — S
- Functions Abstract Domain — F
- Piecewise-Defined Ranking Functions Abstract Domain — V(S, F)



- States Abstract Domain S
- Functions Abstract Domain F
- Piecewise-Defined Ranking Functions Abstract Domain $\mathsf{V}(\mathsf{S}, \mathsf{F})$

Why Piecewise-Defined Ranking Functions?

Example

int : x

while $^{1}(x \neq 0)$ {

  if $^{2}(x < 0)$ { $^{3}x := x + 1;$ } else { $^{4}x := x - 1;$ }

}^{5}

$$f(x) \triangleq \begin{cases} -3x + 1 & x < 0 \\ 1 & x = 0 \\ 3x + 1 & x > 0 \end{cases}$$

Example

int : x

while $^{1}(x \geq 0)$ {

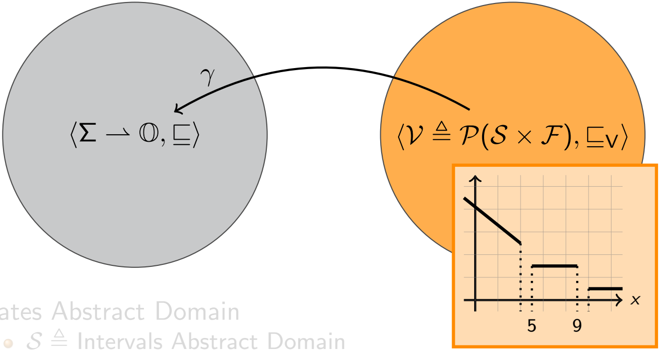
  $^{2}x := -2x + 10;$

}^{3}

$$f(x) \triangleq \begin{cases} 1 & x < 0 \\ 5 & 0 \leq x \leq 2 \\ 9 & x = 3 \\ 7 & 4 \leq x \leq 5 \\ 3 & 5 < x \end{cases}$$

Affine Ranking Functions

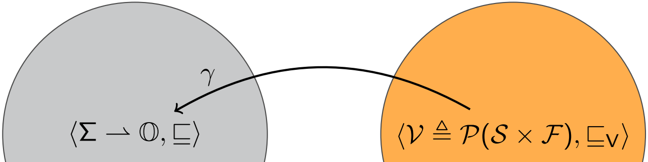
Affine Ranking Functions Domain

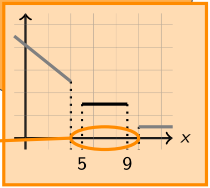


- States Abstract Domain
  - $\mathcal{S} \triangleq$ Intervals Abstract Domain
- Functions Abstract Domain
  - $\mathcal{F} \triangleq \{\bot_{\mathsf{F}}\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top_{\mathsf{F}}\}$
    where $f \equiv f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Cousot&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)

Affine Ranking Functions Domain



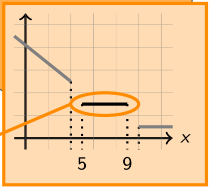


- States Abstract Domain
  - $\mathcal{S} \triangleq$ Intervals Abstract Domain
- Functions Abstract Domain
  - $\mathcal{F} \triangleq \{\bot_{\mathsf{F}}\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top_{\mathsf{F}}\}$
    where $f \equiv f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Cousot&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)

Affine Ranking Functions Domain

$$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle \xleftarrow{\gamma} \langle \mathcal{V} \triangleq \mathcal{P}(\mathcal{S} \times \mathcal{F}), \sqsubseteq_{\mathsf{V}} \rangle$$



- States Abstract Domain
  - $\mathcal{S} \triangleq$ Intervals Abstract Domain
- Functions Abstract Domain
  - $\mathcal{F} \triangleq \{\bot_{\mathsf{F}}\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top_{\mathsf{F}}\}$
    where $f \equiv f(x_1, \dots, x_n) = m_1 x_1 + \dots + m_n x_n + q$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Cousot&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)

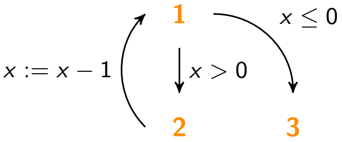
Example

int : x

while $^{1}(x > 0)$ {

$^{2}x := x - 1$

}^{3}



we map each point to a function of x giving an upper bound on the steps before termination

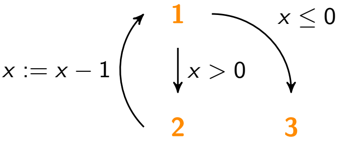
Example

int : x

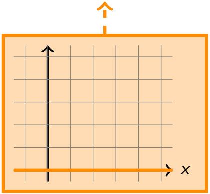
while $^{1}(x > 0)$ {

$^{2}x := x - 1$

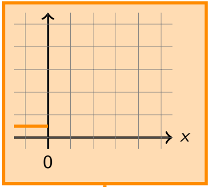
}^{3}



we start at the end with 0 steps before termination



we take into account $x \leq 0$ and we have 1 step to termination



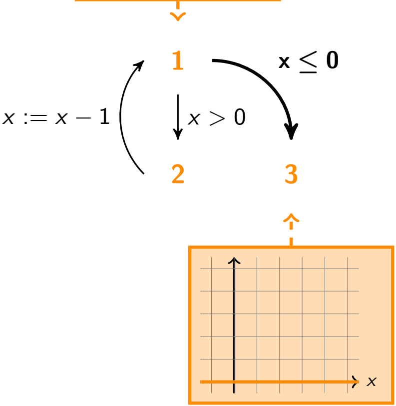
Example

int : x

while $^{1}(x > 0)$ {

$^{2}x := x - 1$

}^{3}



Example

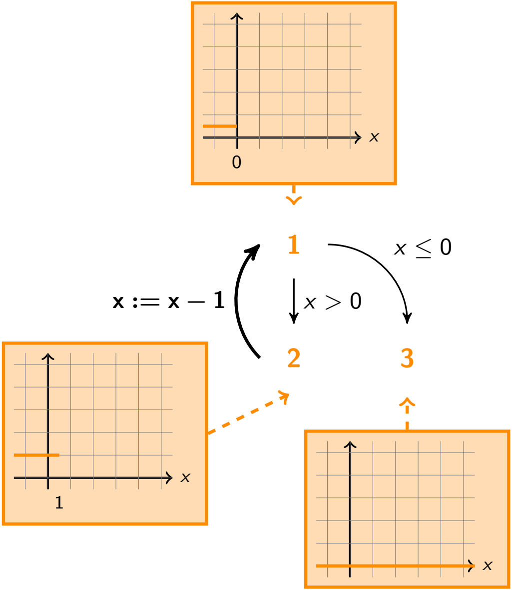
int : x

while [1]$(x > 0)$ {

[2]$x := x - 1$

}[3]

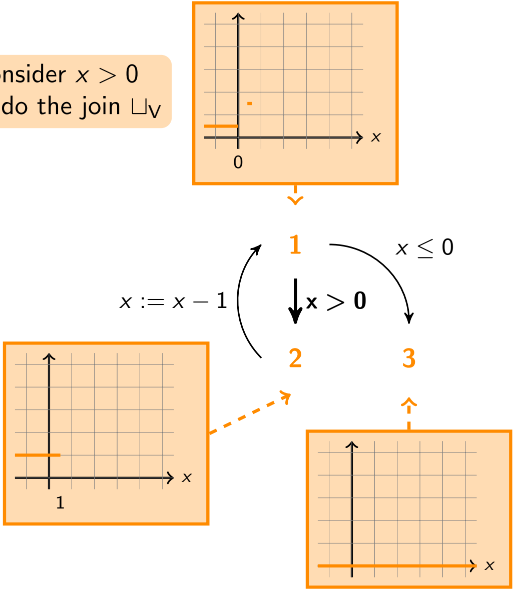
we consider the assignment $x := x - 1$ and we are at 2 steps to termination



we consider $x > 0$
and we do the join $\sqcup_V$

Example

int : x
while [1]$(x > 0)$ {
  [2]$x := x - 1$
}[3]



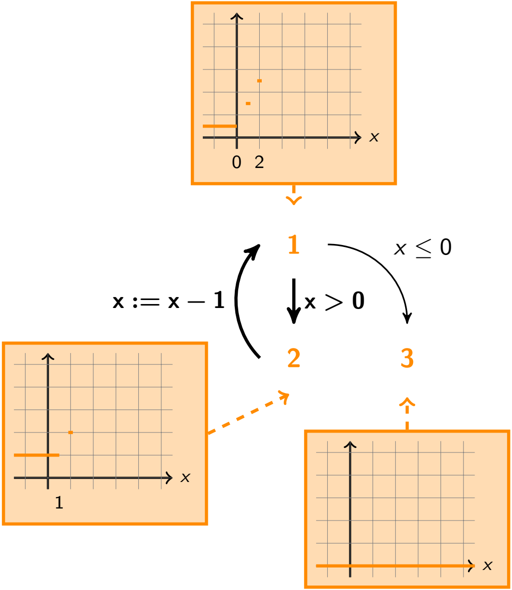
Example

int : x

while $^{1}(x > 0)$ {

$^{2}x := x - 1$

}^{3}



we do the widening ∇_V

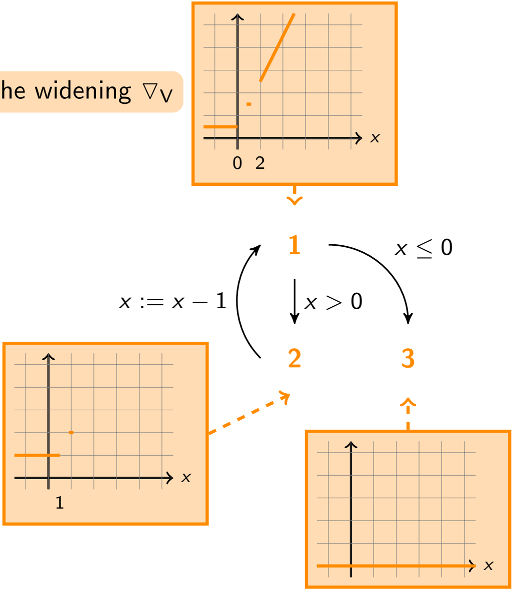
Example

int : x

while [1]$(x > 0)$ {

  [2]$x := x - 1$

}[3]



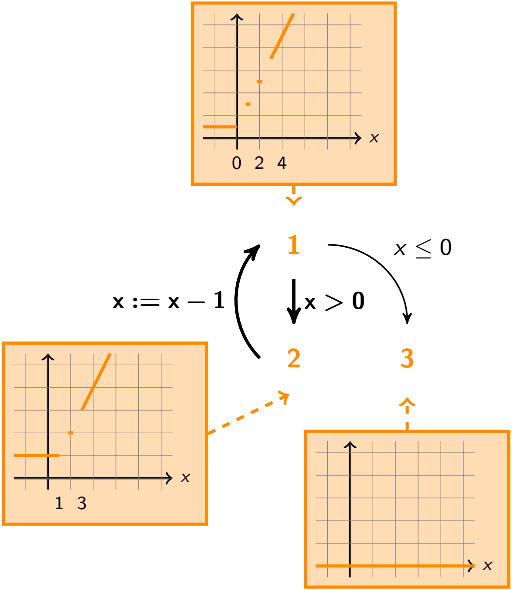
Example

int : x

while [1]$(x > 0)$ {

[2]$x := x - 1$

}[3]



Example

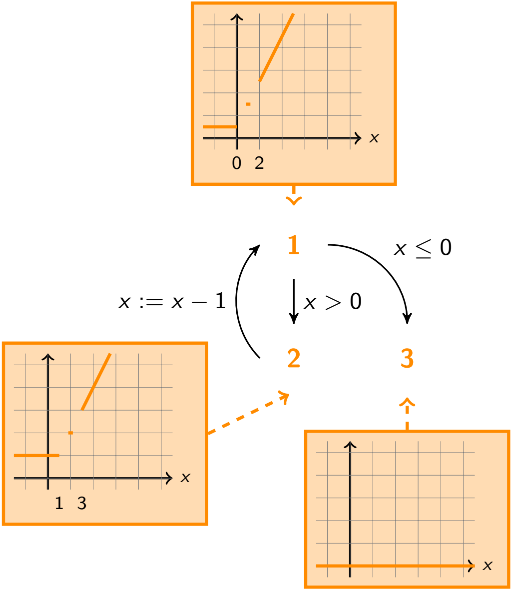
int : x

while [1]$(x > 0)$ {

  [2]$x := x - 1$

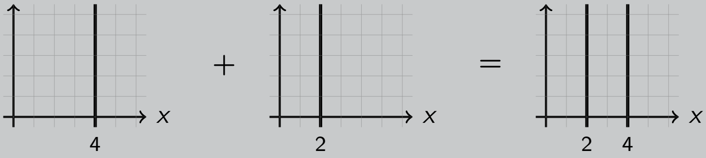
}[3]

the analysis gives true as sufficient precondition for termination



- segmentation unification

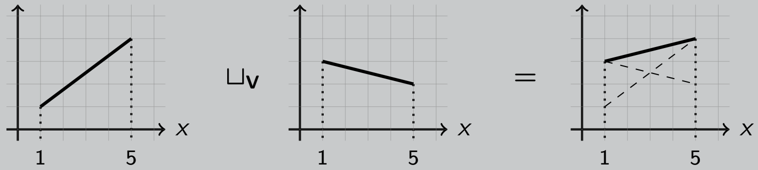
Example



- join: $\sqcup_V$
- widening: $\triangledown_V$
- backward assignments: ASSIGN_V

- segmentation unification
- join: $\sqcup_V$

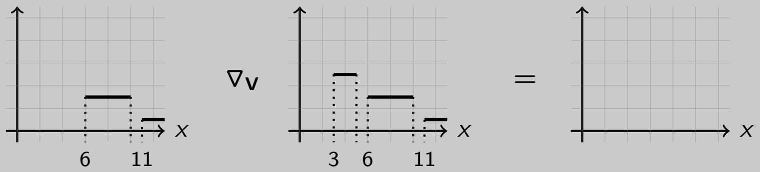
Example



- widening: $\triangledown_V$
- backward assignments: ASSIGN_V

- segmentation unification
- join: $\sqcup_V$
- widening: ∇_V

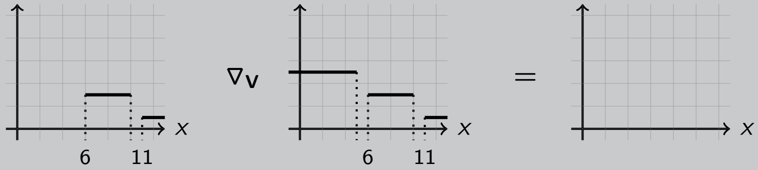
Example



- backward assignments: ASSIGN_V

- segmentation unification
- join: $\sqcup_V$
- widening: ∇_V

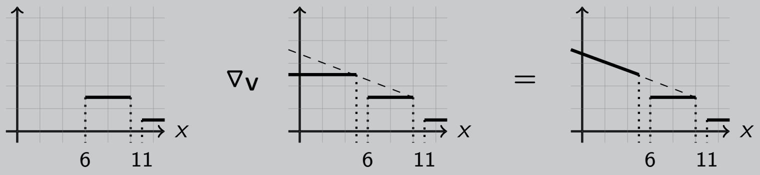
Example



- backward assignments: ASSIGN_V

- segmentation unification
- join: $\sqcup_V$
- widening: ∇_V

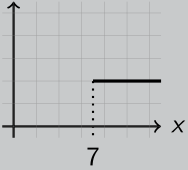
Example



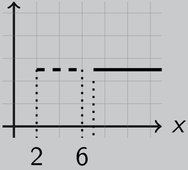
- backward assignments: ASSIGN_V

- segmentation unification
- join: $\sqcup_V$
- widening: $\triangledown_V$
- backward assignments: ASSIGN_V

Example

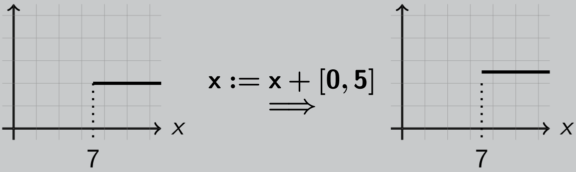


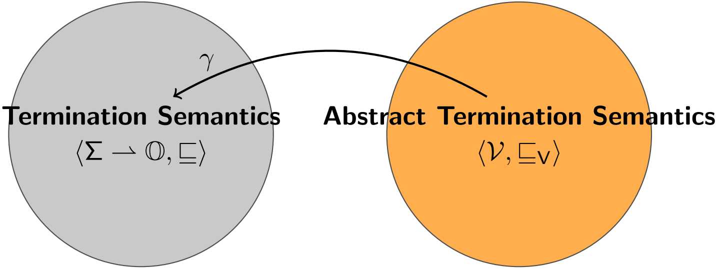
$$\mathbf{x} := \mathbf{x} + [\mathbf{0}, \mathbf{5}] \Longrightarrow$$



- segmentation unification
- join: $\sqcup_V$
- widening: ∇_V
- backward assignments: ASSIGN_V

Example





Theorem (Soundness)

the abstract termination semantics is **sound**
to prove the termination of programs

Example

int : x

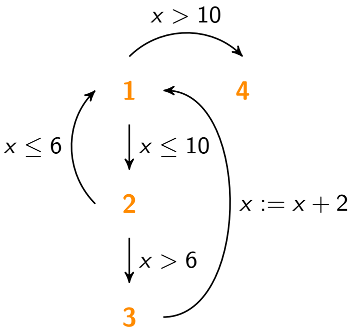
while ${}^{1}(x \leq 10)$ do

  if ${}^{2}(x > 6)$ then

    ${}^{3}x := x + 2$

  fi

od^{4}



we map each point to a function of x giving an upper bound on the steps before termination

Example

int : x

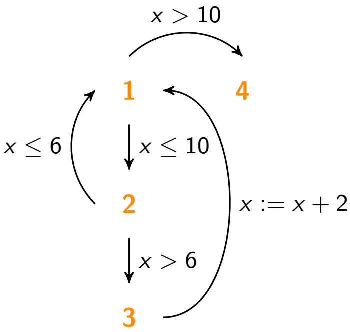
while [1]$(x \leq 10)$ do

  if [2]$(x > 6)$ then

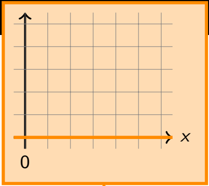
    [3]$x := x + 2$

  fi

od[4]



we start at the end with 0 steps before termination



Example

int : x

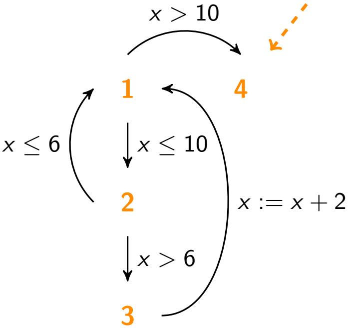
while ${}^{1}(x \leq 10)$ do

    if ${}^{2}(x > 6)$ then

        ${}^{3}x := x + 2$

    fi

od^{4}



we take into account $x > 10$ and we have now 1 step to termination

Example

int : x

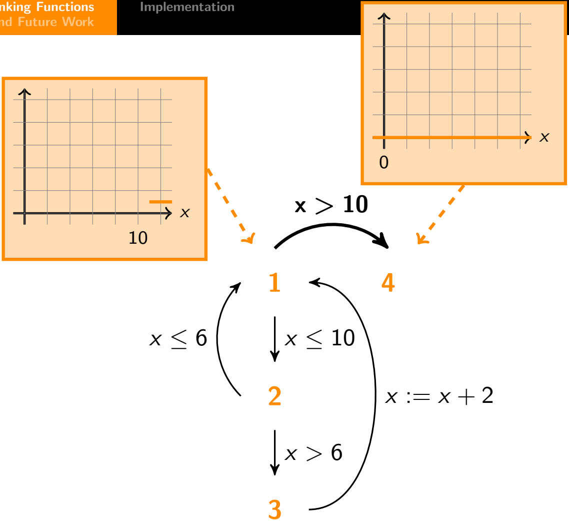
while [1]$(x \leq 10)$ do

  if [2]$(x > 6)$ then

    [3]$x := x + 2$

  fi

od[4]



Example

int : x

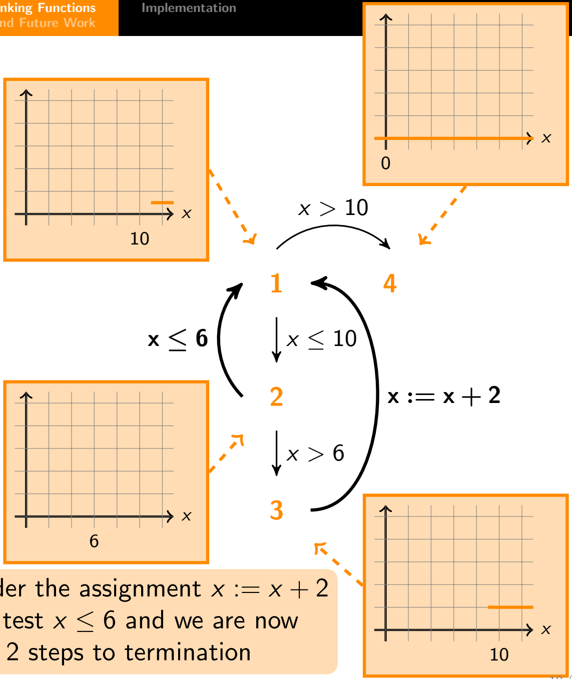
while [1]$(x \leq 10)$ do

  if [2]$(x > 6)$ then

    [3]$x := x + 2$

  fi

od[4]



we consider the assignment $x := x + 2$ or the test $x \leq 6$ and we are now at 2 steps to termination

Example

int : x

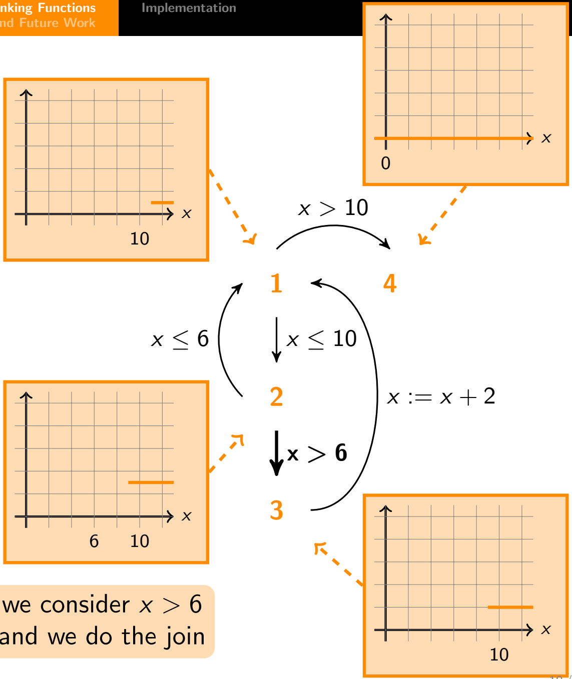
while $^{1}(x \leq 10)$ do

  if $^{2}(x > 6)$ then

    $^{3}x := x + 2$

  fi

od^{4}



we consider $x > 6$
and we do the join

we consider $x \leq 10$ and we do the join

Example

int : x

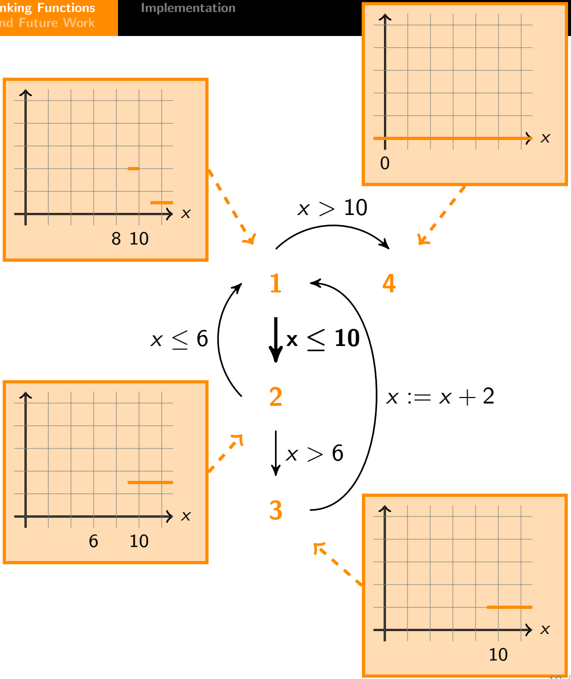
while [1]$(x \leq 10)$ do

if [2]$(x > 6)$ then

[3]$x := x + 2$

fi

od[4]



Example

int : x

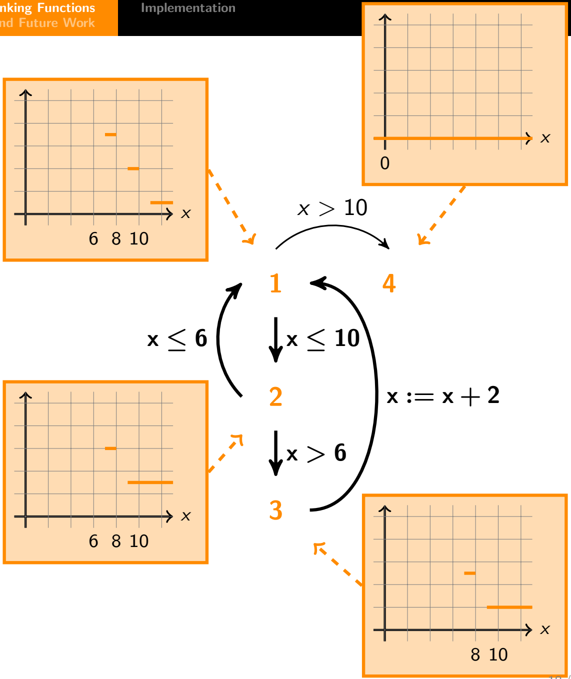
while $^{1}(x \leq 10)$ do

  if $^{2}(x > 6)$ then

    $^{3}x := x + 2$

  fi

od^{4}



we do the widening

Example

int : x

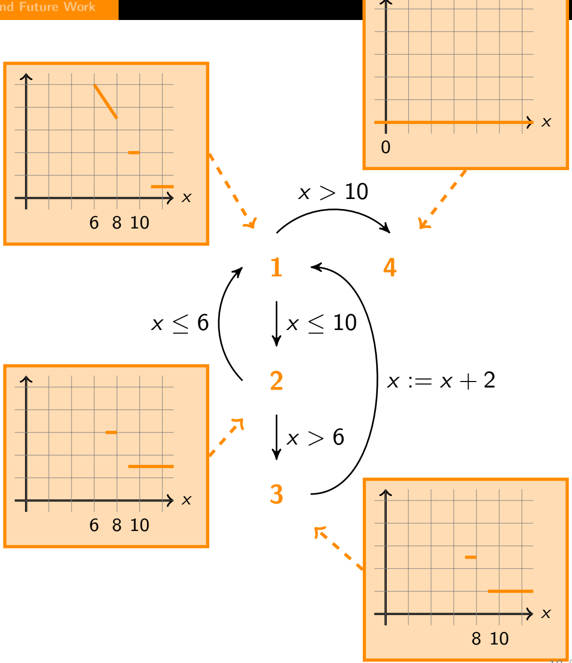
while [1]$(x \leq 10)$ do

  if [2]$(x > 6)$ then

    [3]$x := x + 2$

  fi

od[4]



Example

int : x

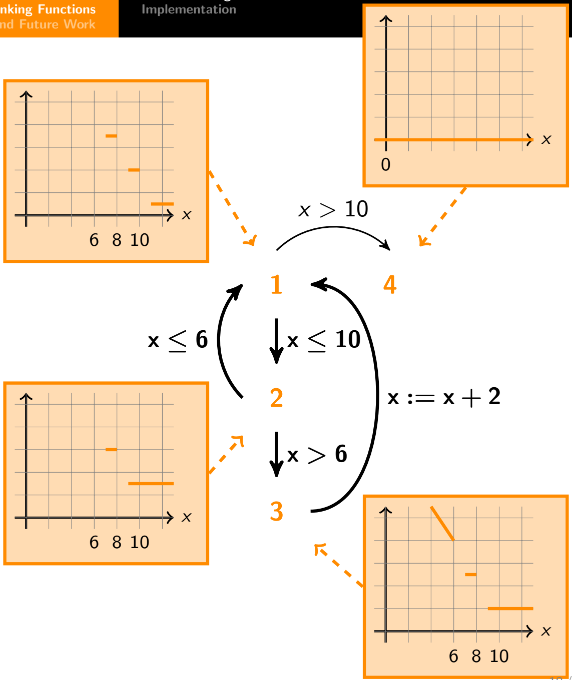
while $^{1}(x \leq 10)$ do

  if $^{2}(x > 6)$ then

    $^{3}x := x + 2$

  fi

od^{4}



the analysis provides $x > 6$ as sufficient precondition for termination

Example

int : x

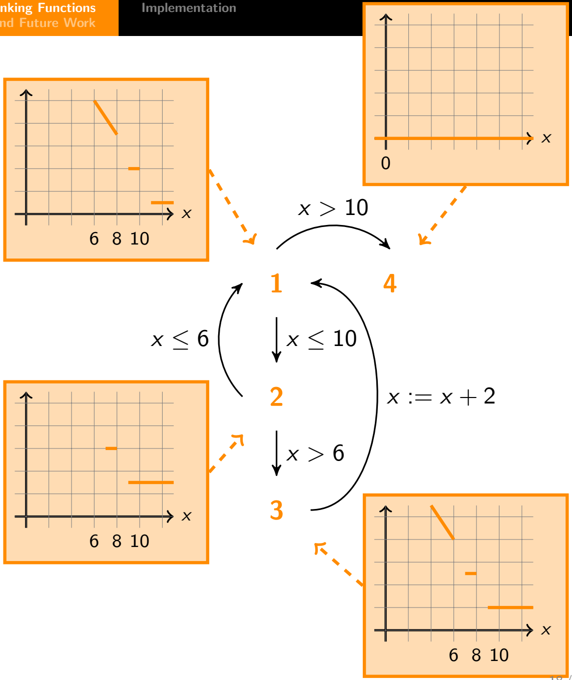
while [1]$(x \leq 10)$ do

  if [2]$(x > 6)$ then

    [3]$x := x + 2$

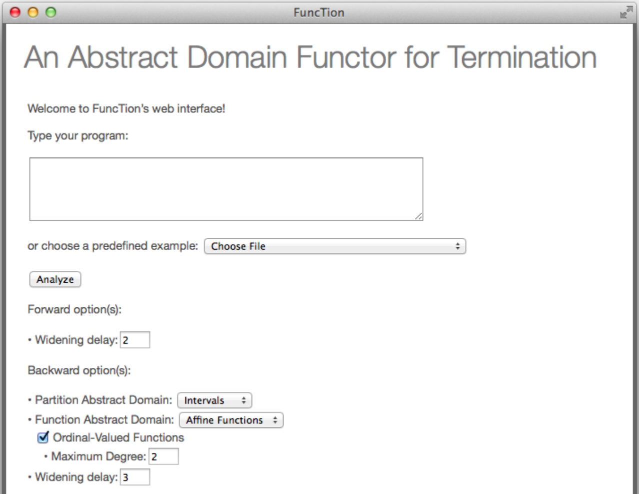
  fi

od[4]



`http://www.di.ens.fr/~urban/FuncTion.html`

- written in OCaml



Experiments

Benchmarks: 87 terminating C programs collected from the literature

Tools:

- AProVE
- T2
- Ultimate Büchi Automizer

Results:

	Tot	FuncTion	AProVE	T2	Ultimate	Time	Timeouts
FuncTion	51	–	8	8	3	6s	5
AProVE	60	17	–	7	2	35s	19
T2	73	30	20	–	3	2s	0
Ultimate	79	31	21	9	–	9s	1

Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **natural-valued functions**
  - affine ranking functions
- instances based on **ordinal-valued functions**
  - ordinals remove the burden of finding lexicographic orders
  - analysis not limited to programs with linear computational complexity

Future Work

- **more abstract domains** (e.g., non-linear ranking functions)
- other liveness properties
- complexity analysis

Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instances based on **natural-valued functions**
  - affine ranking functions
- instances based on **ordinal-valued functions**
  - ordinals remove the burden of finding lexicographic orders
  - analysis not limited to programs with linear computational complexity

Future Work

- **more abstract domains** (e.g., non-linear ranking functions)
- other liveness properties
- complexity analysis

Thank You!

Questions?

"...the purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."
(Edsger Dijkstra)