The Abstract Domain of Segmented Ranking Functions

Caterina Urban

ENS
Département d'Informatique
École Normale Supérieure

SAS 2013
Seattle, USA

Introduction

- **liveness properties** ⇒ "something good eventually happens"
  - **termination**
- **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** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instance based on **intervals** and **affine functions**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

Introduction

- **liveness properties** ⇒ "something good eventually happens"
  - **termination**
- **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** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instance based on **intervals** and **affine functions**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

Introduction

- **liveness properties** ⇒ "something good eventually happens"
  - **termination**
- **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** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instance based on **intervals** and **affine functions**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

Our Contribution

- **liveness properties** $\Rightarrow$ "something good eventually happens"
  - **termination**
- **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** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instance based on **intervals** and **affine functions**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

Our Contribution

- **liveness properties** ⇒ "something good eventually happens"
  - **termination**
- **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** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
  - sufficient conditions for termination
- instance based on **intervals** and **affine functions**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

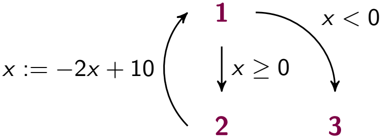
Example

int : x

while [1]$(x \geq 0)$ do

[2]$x := -2x + 10$

od[3]



Example

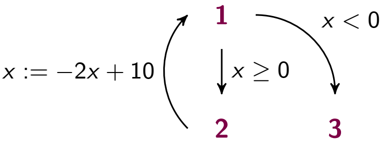
int : x

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

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

od^{3}

the program terminates but there exists no linear ranking function!



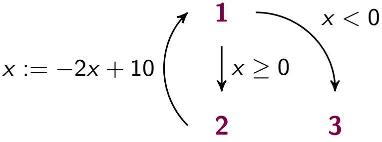
Example

int : x

while [1]$(x \geq 0)$ do

[2]$x := -2x + 10$

od[3]



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

Example

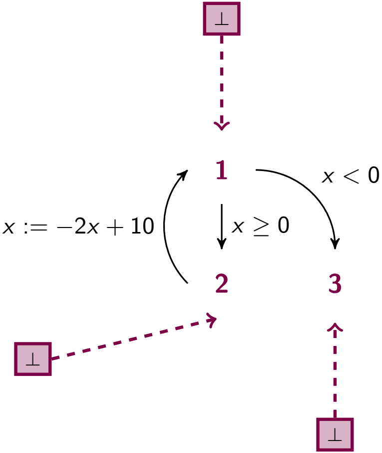
int : x

while [1]$(x \geq 0)$ do

  [2]$x := -2x + 10$

od[3]

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



Example

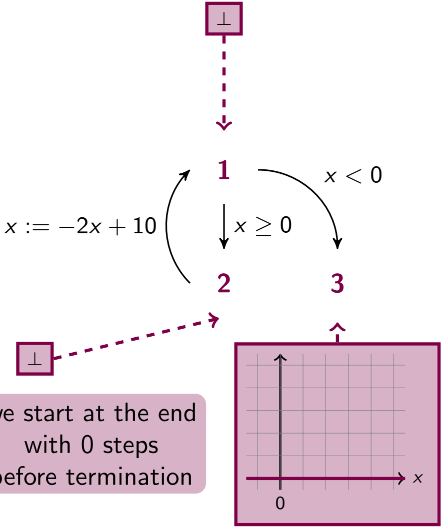
int : x

while [1]$(x \geq 0)$ do

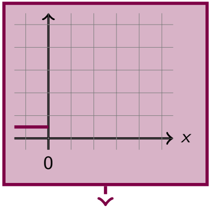
[2]$x := -2x + 10$

od[3]

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



we start at the end with 0 steps before termination we take into account $x < 0$ and we have now 1 step to termination



Example

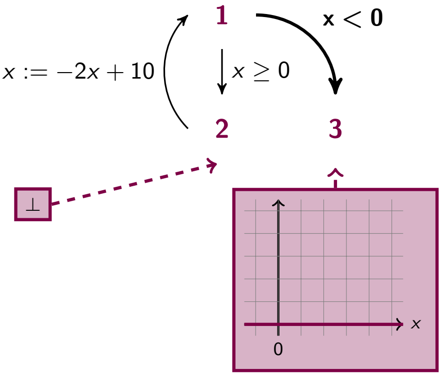
int : x

while [1]$(x \geq 0)$ do

  [2]$x := -2x + 10$

od[3]

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



we consider the assignment and we are now at 2 steps to termination

Example

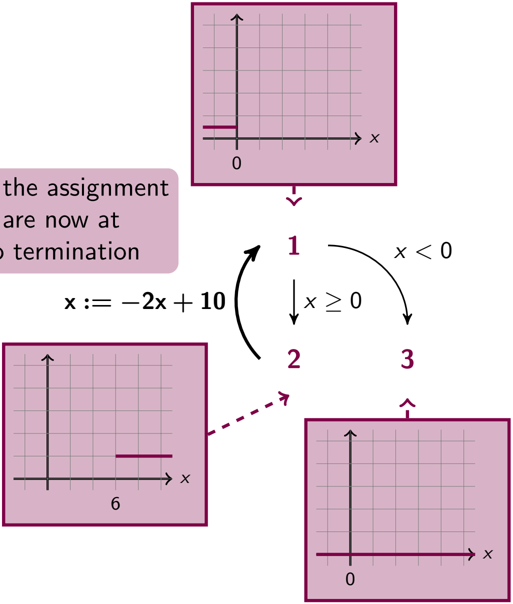
int : x

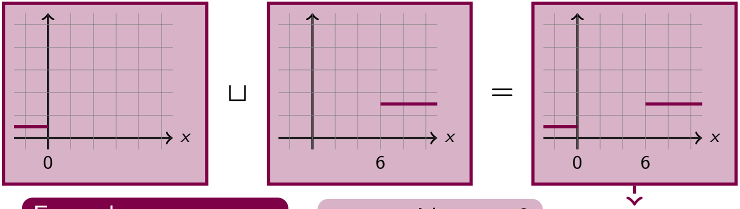
while $^{1}(x \geq 0)$ do

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

od^{3}

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





Example

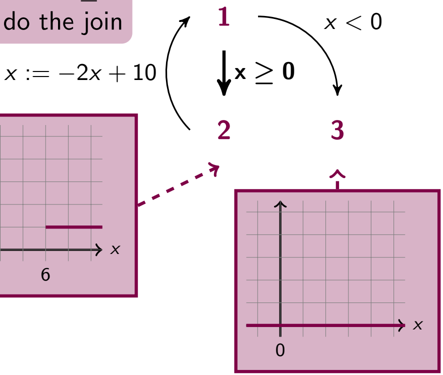
int : x

while [1]$(x \geq 0)$ do

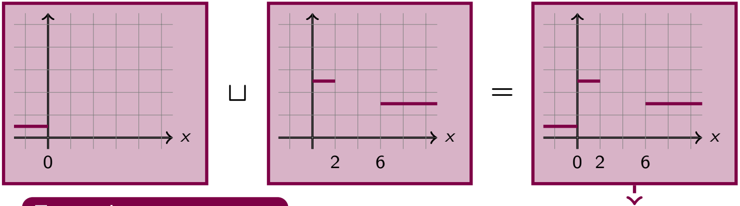
  [2]$x := -2x + 10$

od[3]

we consider $x \geq 0$ and we do the join



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



Example

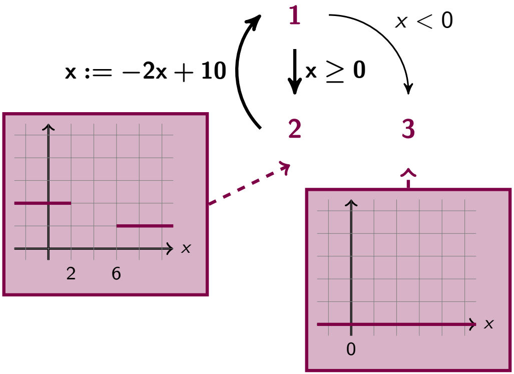
int : x

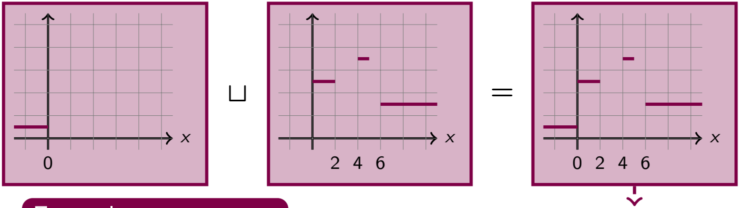
while [1]$(x \geq 0)$ do

[2]$x := -2x + 10$

od[3]

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





Example

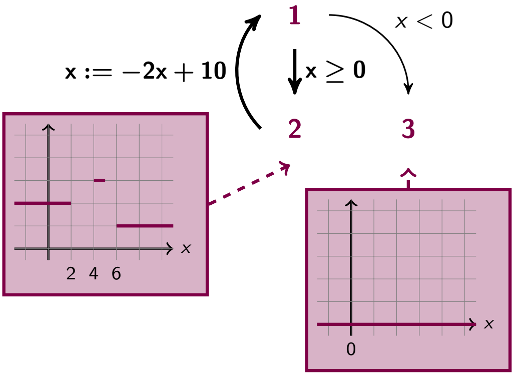
int : x

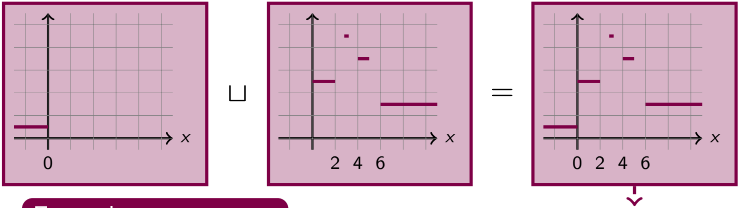
while [1]$(x \geq 0)$ do

[2]$x := -2x + 10$

od[3]

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





Example

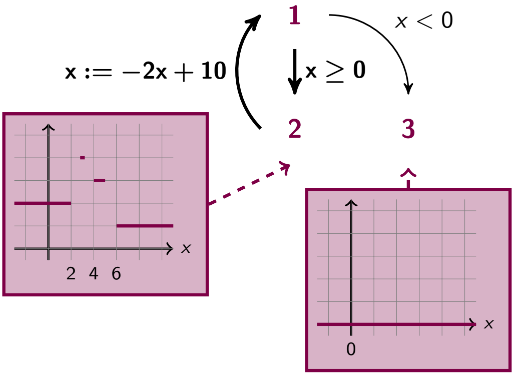
int : x

while [1]$(x \geq 0)$ do

[2]$x := -2x + 10$

od[3]

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



we are able to find a piecewise-defined ranking function for the program!

Example

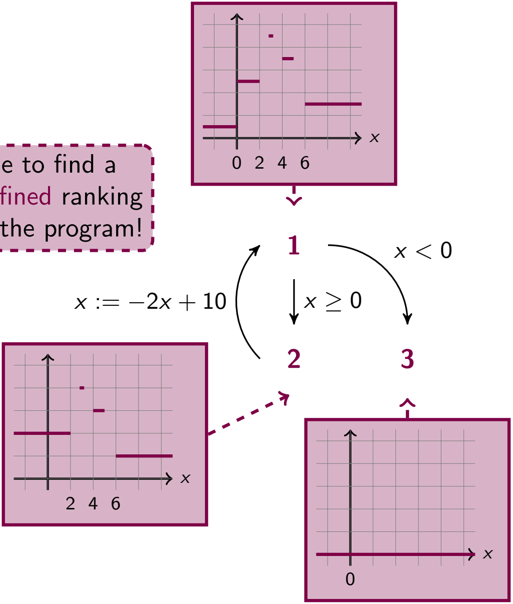
int : x

while $^1(x \geq 0)$ do

$^2x := -2x + 10$

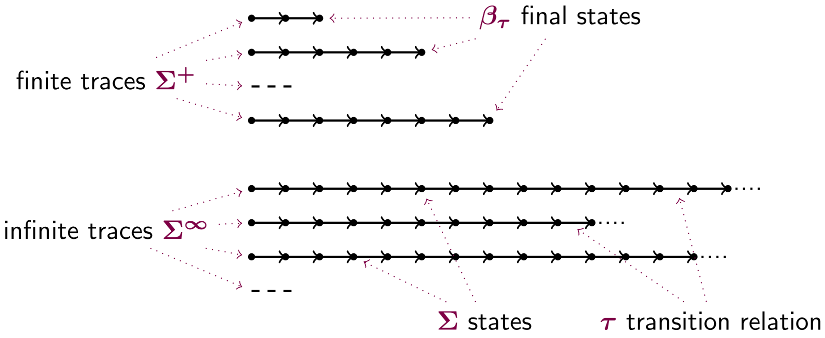
od^3

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



Concrete Semantics

program $P \mapsto$ **trace semantics**

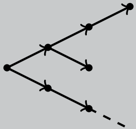


$$v_\tau \in \Sigma \not\rightharpoonup \mathbb{O}$$

$$v_\tau \triangleq \text{lfp } \phi_\tau$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\text{pre}}(\text{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

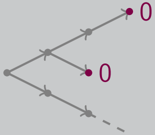
Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

$$v_\tau \in \Sigma \nrightarrow \mathbb{O}$$

$$v_\tau \triangleq \text{lfp } \phi_\tau$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\text{pre}}(\text{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

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

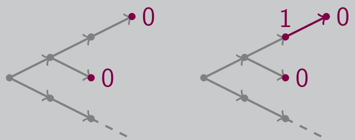
$$v_\tau \in \Sigma \nrightarrow \mathbb{O}$$

$$v_\tau \triangleq \text{lfp } \phi_\tau$$

$$\widetilde{\text{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \in \tau \Rightarrow s' \in X\}$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\text{pre}}(\text{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

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

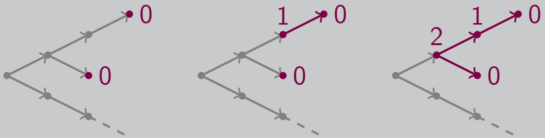
$$v_\tau \in \Sigma \not\rightharpoonup \mathbb{O}$$

$$v_\tau \triangleq \mathrm{lfp}\ \phi_\tau$$

$$\widetilde{\mathrm{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \in \tau \Rightarrow s' \in X\}$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

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

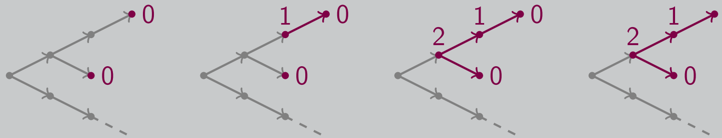
$$v_\tau \in \Sigma \not\rightharpoonup \mathbb{O}$$

$$v_\tau \triangleq \text{lfp } \phi_\tau$$

$$\widetilde{\text{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s, s' \rangle \in \tau \Rightarrow s' \in X\}$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\text{pre}}(\text{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

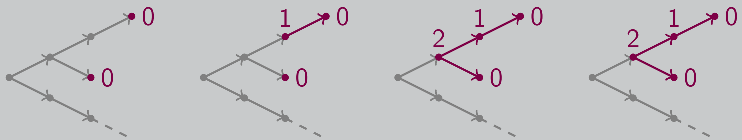
Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

$$v_\tau \in \Sigma \not\rightharpoonup \mathbb{O}$$

$$v_\tau \triangleq \text{lfp } \phi_\tau$$

$$\phi_\tau(v) \triangleq \lambda s. \begin{cases} 0 & \text{if } s \in \beta_\tau \\ \sup\{v(s') + 1 \mid \langle s, s' \rangle \in \tau\} & \text{if } s \in \widetilde{\text{pre}}(\text{dom}(v)) \end{cases}$$

Example



Theorem (Soundness and Completeness)

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

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

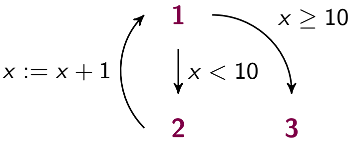
Example

int : x

while $^{1}(x < 10)$ do

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

od^{3}



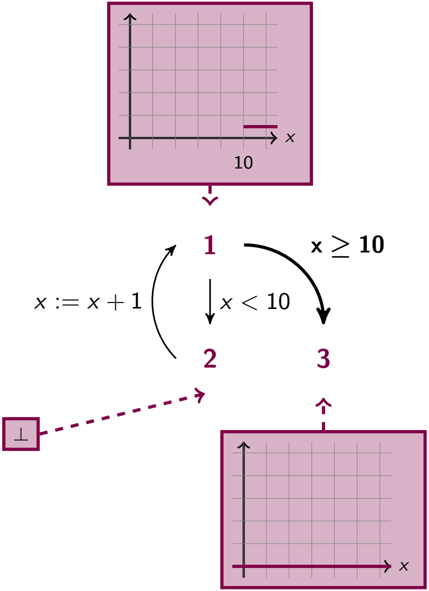
Example

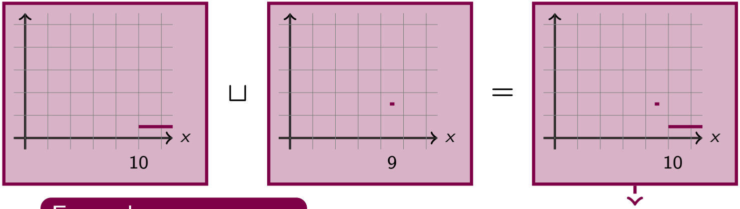
int : x

while [1]$(x < 10)$ do

  [2]$x := x + 1$

od[3]





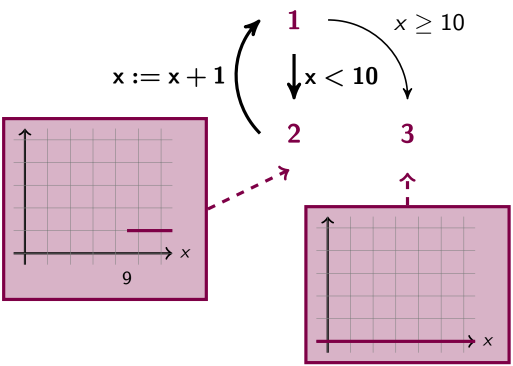
Example

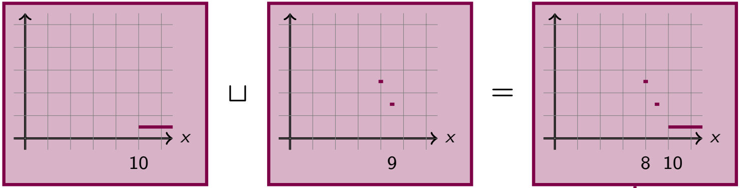
int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]





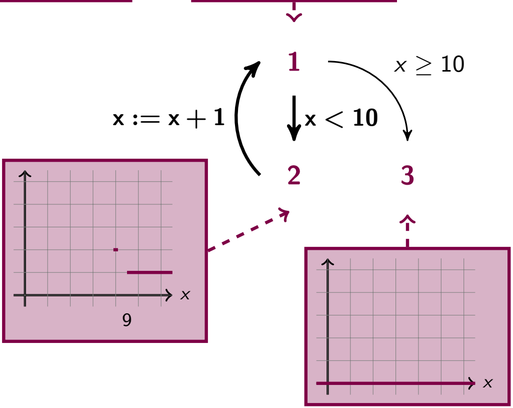
Example

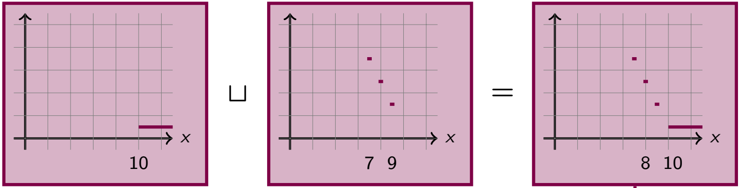
int : x

while [1]$(x < 10)$ do

  [2]$x := x + 1$

od[3]





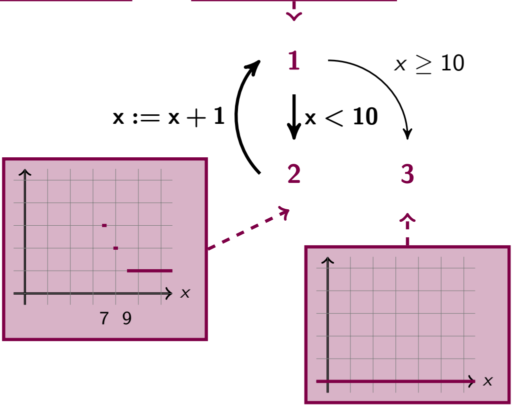
Example

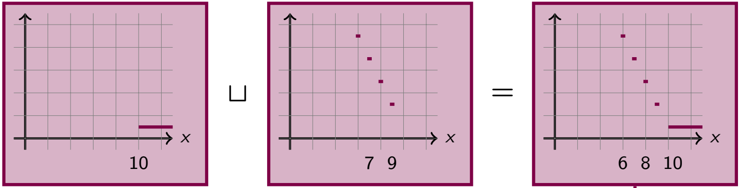
int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]





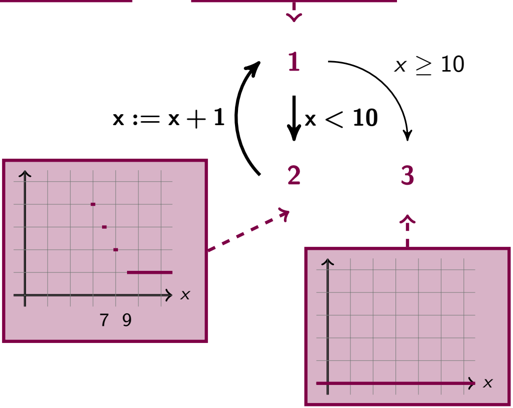
Example

int : x

while $^{1}(x < 10)$ do

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

od^{3}



Example

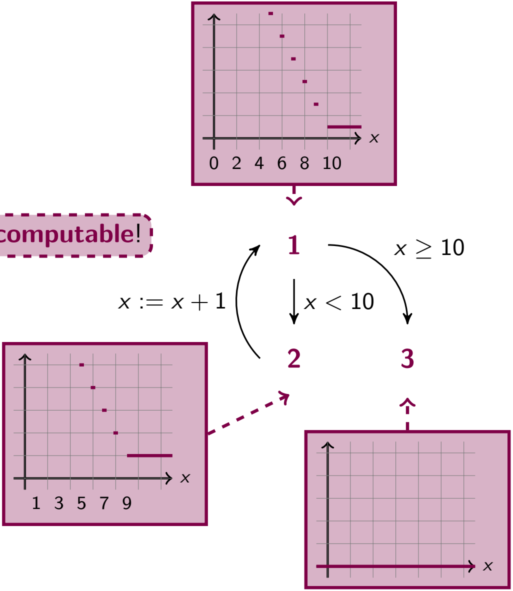
int : x

while [1]$(x < 10)$ do

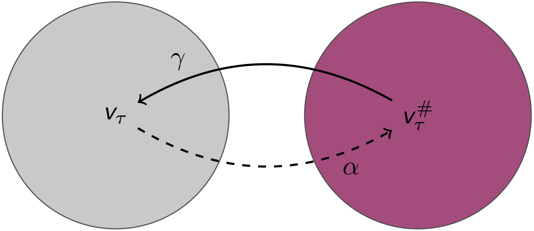
[2]$x := x + 1$

od[3]

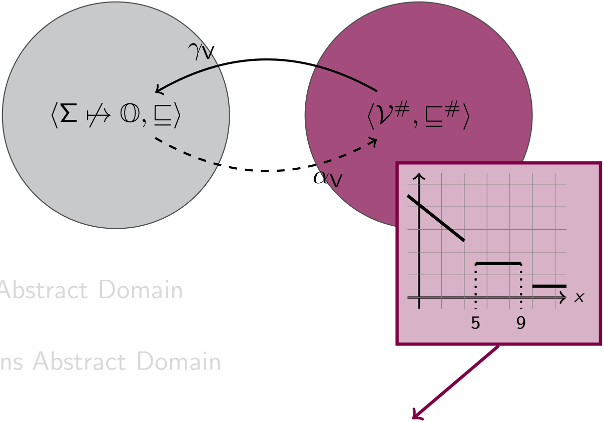
v_τ is **not computable!**



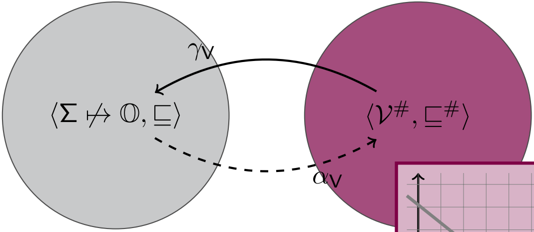
An Abstract Domain for Termination

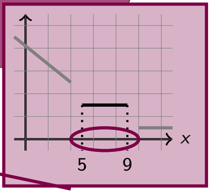


- States Abstract Domain $\mathcal{E}$
- Functions Abstract Domain $\mathcal{P}$
- Segmented Ranking Functions Abstract Domain $\mathcal{V}(\mathcal{E}, \mathcal{P})$

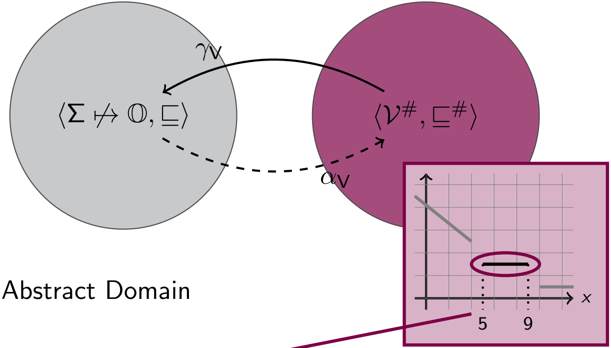


- States Abstract Domain E
- Functions Abstract Domain P
- Segmented Ranking Functions Abstract Domain V(E, P)

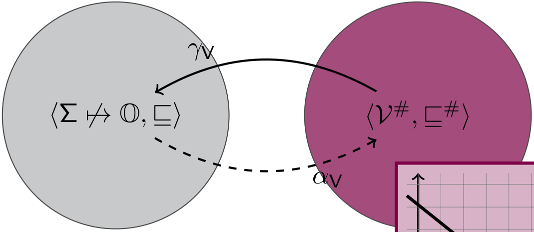


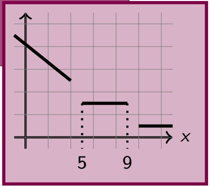


- States Abstract Domain — E
- Functions Abstract Domain — P
- Segmented Ranking Functions Abstract Domain — V(E, P)



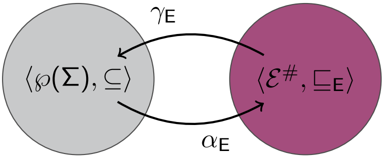
- States Abstract Domain — E
- Functions Abstract Domain — P
- Segmented Ranking Functions Abstract Domain — V(E, P)





- States Abstract Domain — E
  - Intervals Abstract Domain
- Functions Abstract Domain — P
  - Affine Functions Abstract Domain
- Segmented Ranking Functions Abstract Domain — V(E, P)
  - Segmented Affine Ranking Functions Abstract Domain

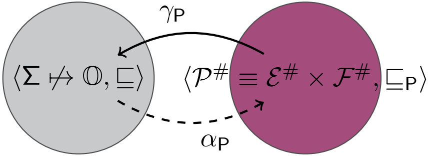
Intervals Abstract Domain[3]



- $\mathcal{E}^\# \triangleq \{\bot_E\} \cup \{[a, b] \mid a \in \mathbb{I} \cup \{-\infty\}, b \in \mathbb{I} \cup \{+\infty\}\} \qquad \mathbb{I} \in \{\mathbb{Z}, \dots\}$
- join: $\sqcup_E$
- meet: $\sqcap_E$
- widening: $\triangledown_E$
- backward assignments: ASSIGN_E
- tests: FILTER_E

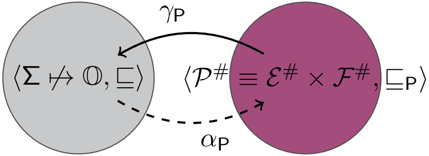
[3] Cousot&Cousot - *Static Determination of Dynamic Properties of Programs* (1976)

Affine Functions Abstract Domain



- $\mathcal{F}^{\#} \triangleq \{\bot_{\mathsf{F}}\} \cup \{f^{\#} \mid f^{\#} \in \mathbb{I}^n \mapsto \mathbb{N}\} \cup \{\top_{\mathsf{F}}\}$
  where $f^{\#} \equiv y = f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- approximation order:
  $\langle \rho_1^{\#}, f_1^{\#} \rangle \sqsubseteq_{\mathsf{P}} \langle \rho_2^{\#}, f_2^{\#} \rangle \triangleq \rho_1^{\#} \sqsupseteq_{\mathsf{E}} \rho_2^{\#} \wedge f_1^{\#} \sqsubseteq_{\mathsf{F}} f_2^{\#}$
  computational order:
  $\langle \rho_1^{\#}, f_1^{\#} \rangle \preccurlyeq_{\mathsf{P}} \langle \rho_2^{\#}, f_2^{\#} \rangle \triangleq \rho_1^{\#} \sqsubseteq_{\mathsf{E}} \rho_2^{\#} \wedge f_1^{\#} \sqsubseteq_{\mathsf{F}} f_2^{\#}$

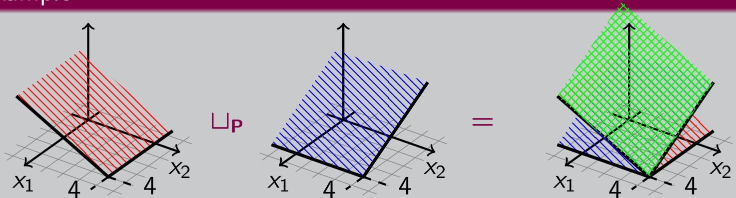
Affine Functions Abstract Domain



- $\mathcal{F}^\# \triangleq \{\bot_F\} \cup \{f^\# \mid f^\# \in \mathbb{I}^n \mapsto \mathbb{N}\} \cup \{\top_F\}$
  where $f^\# \equiv y = f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- approximation order:
  $\langle \rho_1^\#, f_1^\# \rangle \sqsubseteq_P \langle \rho_2^\#, f_2^\# \rangle \triangleq \rho_1^\# \sqsupseteq_E \rho_2^\# \wedge f_1^\# \sqsubseteq_F f_2^\#$
  computational order:
  $\langle \rho_1^\#, f_1^\# \rangle \preccurlyeq_P \langle \rho_2^\#, f_2^\# \rangle \triangleq \rho_1^\# \sqsubseteq_E \rho_2^\# \wedge f_1^\# \sqsubseteq_F f_2^\#$

- join: $\sqcup_P$

Example



$f_1(x_1, x_2) = -\frac{1}{2}x_2 + 2$ $\quad f_2(x_1, x_2) = -\frac{1}{2}x_1 + 2$ $\quad f(x_1, x_2) = -\frac{1}{2}x_1 - \frac{1}{2}x_2 + 4$

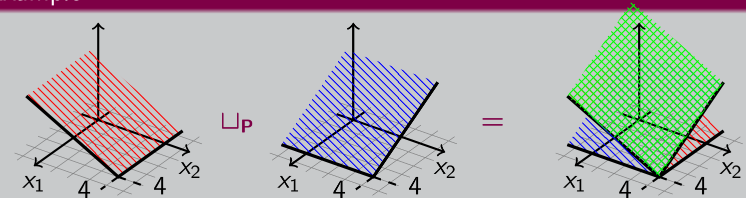
- backward assignments: ASSIGN_P

Example

$f(x) = x - 2$ $\quad f(x) = \mathbf{x + 1} - 2 + \mathbf{1} = x$

- join: $\sqcup_P$

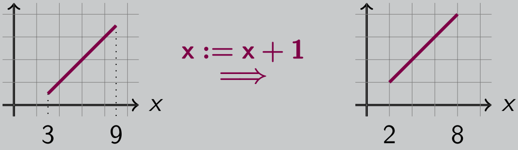
Example



$f_1(x_1, x_2) = -\frac{1}{2}x_2 + 2$ $\quad$ $f_2(x_1, x_2) = -\frac{1}{2}x_1 + 2$ $\quad$ $f(x_1, x_2) = -\frac{1}{2}x_1 - \frac{1}{2}x_2 + 4$

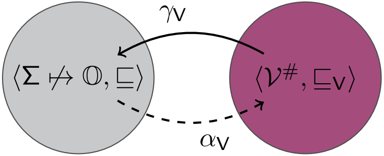
- backward assignments: ASSIGN_P

Example



$f(x) = x - 2$ $\quad$ $f(x) = \mathbf{x + 1} - 2 \mathbf{+ 1} = x$

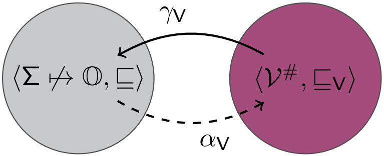
Segmented Affine Ranking Functions Domain



- $\mathcal{V}^\# \triangleq \{(\mathcal{E}^\# \times \mathcal{F}^\#)^k \mid k \geq 0\}$
- segmentation unification

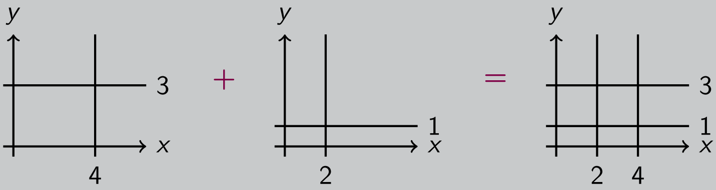
Example

Segmented Affine Ranking Functions Domain



- $\mathcal{V}^\# \triangleq \{(\mathcal{E}^\# \times \mathcal{F}^\#)^k \mid k \geq 0\}$
- segmentation unification

Example

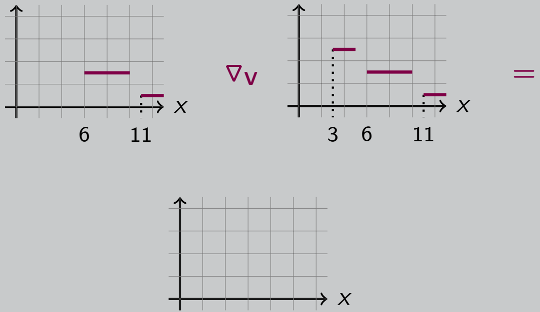


- approximation order: $\sqsubseteq_V$
  computational order: $\preccurlyeq_V$
- join: $\sqcup_V$
- widening: ∇_V

Example

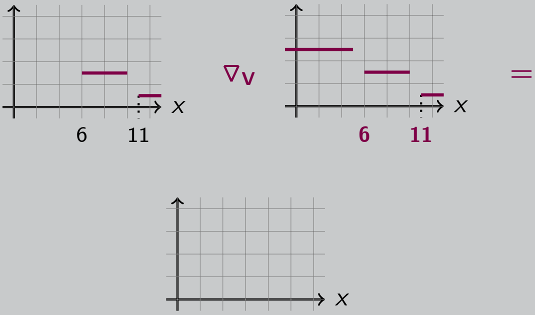
- approximation order: $\sqsubseteq_V$
  computational order: $\preccurlyeq_V$
- join: $\sqcup_V$
- widening: $\triangledown_V$

Example



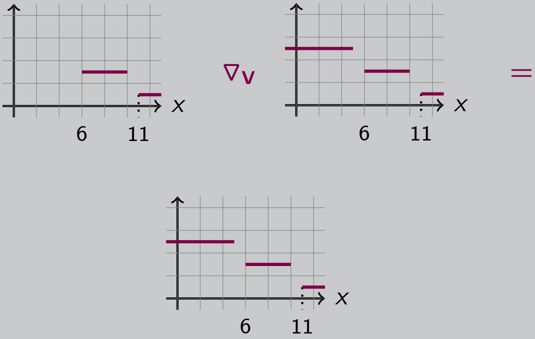
- approximation order: $\sqsubseteq_V$
  computational order: $\preccurlyeq_V$
- join: $\sqcup_V$
- widening: ∇_V

Example



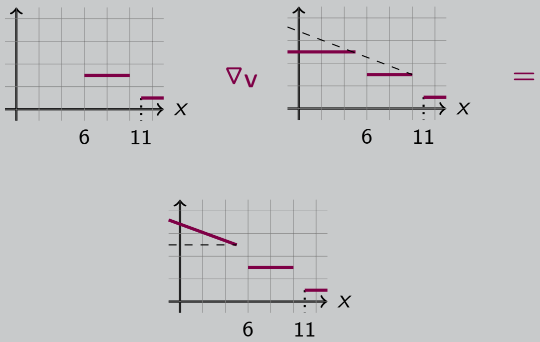
- approximation order: $\sqsubseteq_V$
  computational order: $\preccurlyeq_V$
- join: $\sqcup_V$
- widening: $\triangledown_V$

Example



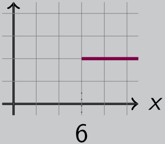
- approximation order: $\sqsubseteq_V$
  computational order: $\preccurlyeq_V$
- join: $\sqcup_V$
- widening: ∇_V

Example



- backward assignments: $\mathrm{ASSIGN}_\mathsf{V}$

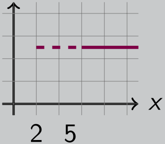
Example



$\langle x \mapsto (-\infty, 5], \bot_\mathsf{F} \rangle$

$\langle x \mapsto [6, +\infty), y = 4 \rangle$

$\mathbf{x := x + [0, 4]}$ $\Longrightarrow$



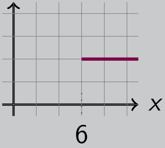
$\langle x \mapsto (-\infty, \mathbf{5}], \bot_\mathsf{F} \rangle$

$\langle x \mapsto [\mathbf{2}, +\infty), y = 4 + \mathbf{1} \rangle$

- tests: $\mathrm{FILTER}_\mathsf{V}$

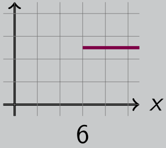
- backward assignments: $\mathrm{ASSIGN}_{\mathsf{V}}$

Example



$\langle x \mapsto (-\infty, 5], \bot_F \rangle$
$\langle x \mapsto [6, +\infty), y = 4 \rangle$

$\mathbf{x := x + [0, 4]}$ $\Longrightarrow$

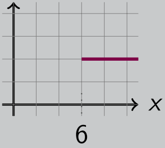


$\langle x \mapsto (-\infty, 1], \bot_F \rangle$
$\langle x \mapsto [2, 5], \bot_F \rangle$
$\langle x \mapsto [6, +\infty), y = 5 \rangle$

- tests: $\mathrm{FILTER}_{\mathsf{V}}$

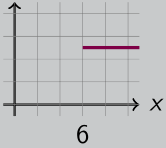
- backward assignments: $\mathrm{ASSIGN}_{\mathsf{V}}$

Example



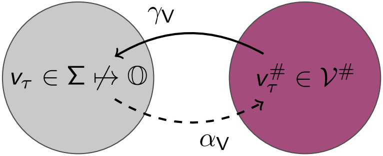
$\langle x \mapsto (-\infty, 5], \bot_{\mathsf{F}} \rangle$
$\langle x \mapsto [6, +\infty), y = 4 \rangle$

$$\mathbf{x} := \mathbf{x} + [\mathbf{0}, \mathbf{4}] \Longrightarrow$$



$\langle x \mapsto (-\infty, 1], \bot_{\mathsf{F}} \rangle$
$\langle x \mapsto [2, 5], \bot_{\mathsf{F}} \rangle$
$\langle x \mapsto [6, +\infty), y = 5 \rangle$

- tests: $\mathrm{FILTER}_{\mathsf{V}}$



$$\boldsymbol{\mathcal{S}}^\#[\![\textbf{statement}]\!] \in \mathcal{V}^\#_{\textbf{POST}} \mapsto \mathcal{V}^\#_{\textbf{PRE}}$$

$$\boldsymbol{\mathcal{S}}^\#[\![\mathbf{x} := A]\!]v \triangleq \mathrm{ASSIGN}_V(\mathbf{x} := A, v)$$

$$\boldsymbol{\mathcal{S}}^\#[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}]\!]v \triangleq$$

$$\mathrm{FILTER}_V(B, \boldsymbol{\mathcal{S}}^\#[\![S_1]\!]v) \curlyvee_V \mathrm{FILTER}_V(\neg B, \boldsymbol{\mathcal{S}}^\#[\![S_2]\!]v)$$

$$\boldsymbol{\mathcal{S}}^\#[\![\textbf{while } B \textbf{ do } S \textbf{ od}]\!]v \triangleq \mathrm{lfp}^{\#\preccurlyeq_V}_{\bot_V}\ \phi^\#$$

$$\text{where } \phi^\# \triangleq \lambda x.\ \mathrm{FILTER}_V(\neg B, v) \curlyvee_V \mathrm{FILTER}_V(B, \boldsymbol{\mathcal{S}}^\#[\![S]\!]x)$$

$$\boldsymbol{\mathcal{S}}^\#[\![S_1\ ;\ S_2]\!]v \triangleq \boldsymbol{\mathcal{S}}^\#[\![S_1]\!](\boldsymbol{\mathcal{S}}^\#[\![S_2]\!]v)$$

Theorem (Soundness)

$\mathbf{v}_\tau^\#$ *is* **sound** *to prove the termination of programs*

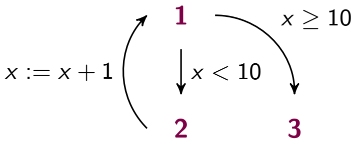
Example

int : x

while [1]$(x < 10)$ do

  [2]$x := x + 1$

od[3]



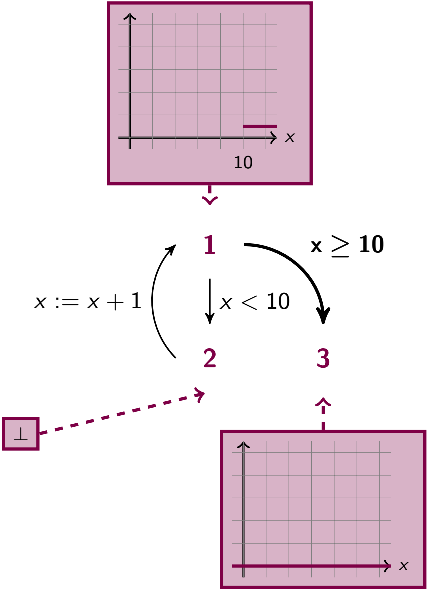
Example

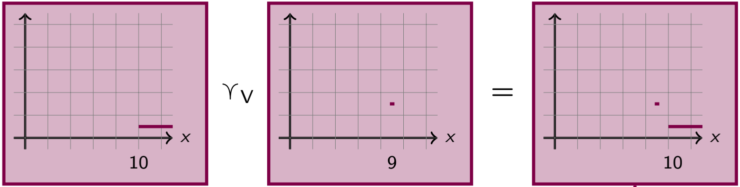
int : x

while [1]($x < 10$) do

  [2]$x := x + 1$

od[3]





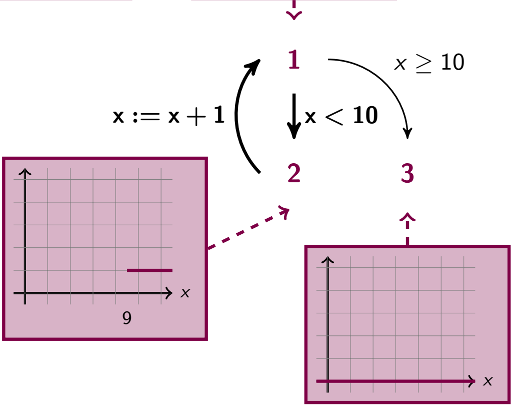
Example

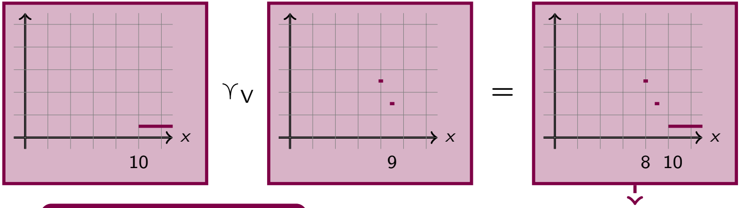
int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]





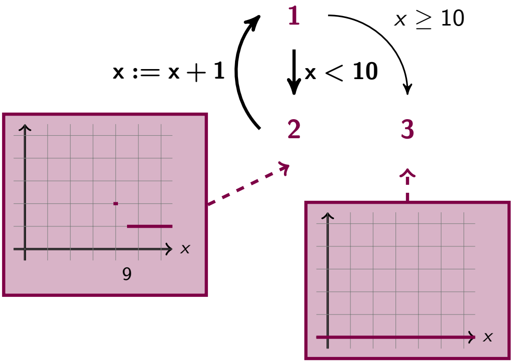
Example

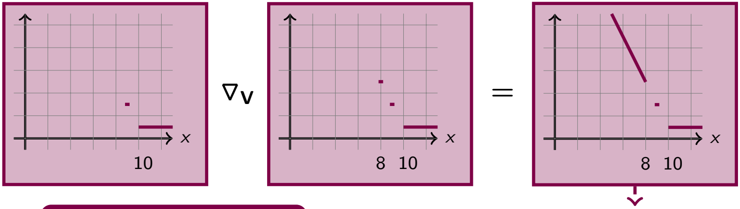
int : x

while [1]($x < 10$) do

[2]$x := x + 1$

od[3]





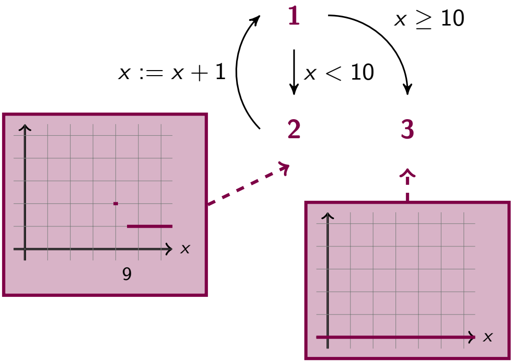
Example

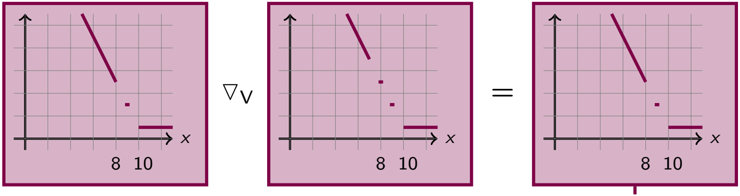
int : x

while [1]$(x < 10)$ do

  [2]$x := x + 1$

od[3]





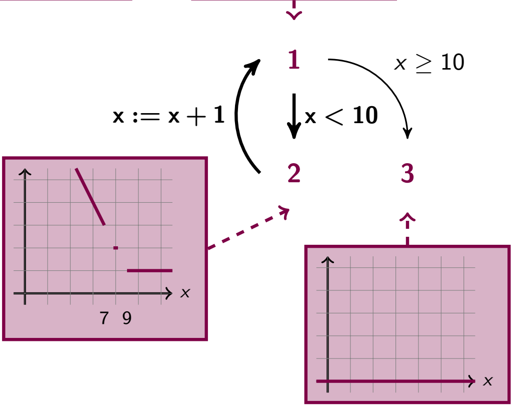
Example

int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]



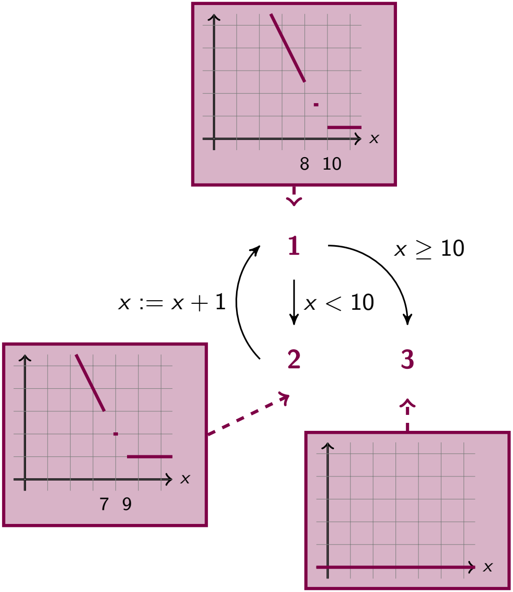
Example

int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]



Alias&Darte&Feautrier&Gonnord - *Multi-Dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs* (SAS 2010)

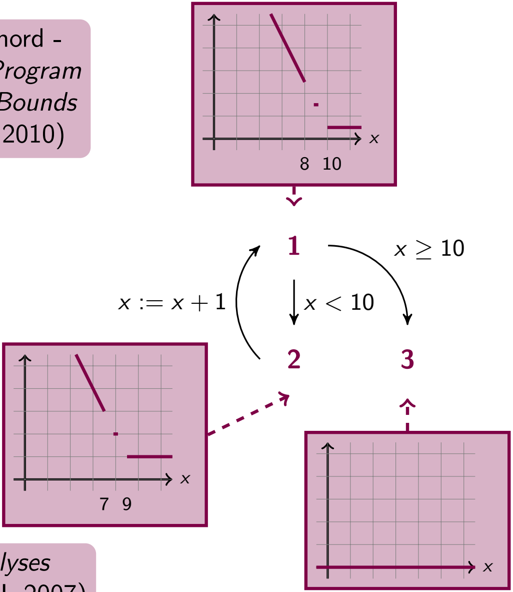
Example

int : x

while [1]$(x < 10)$ do

[2]$x := x + 1$

od[3]



Berdine&al. - *Variance Analyses from Invariance Analyses* (POPL 2007)

Simple Loops

Example

int : x_1, x_2

while ${}^{1}(x_1 \geq 0 \wedge x_2 \geq 0)$ do

  if ${}^{2}(?)$ then

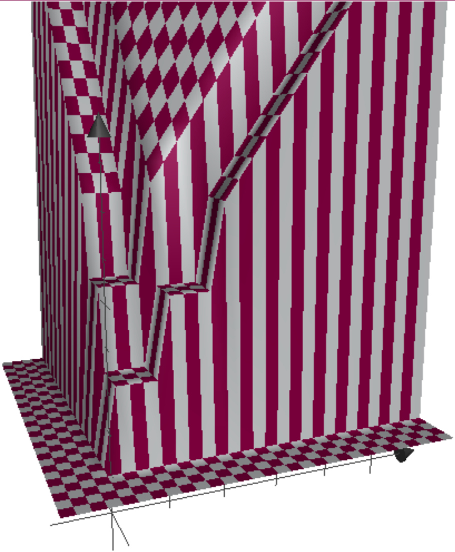
    ${}^{3}x_1 := x_1 - 1$

  else

    ${}^{4}x_2 := x_2 - 1$

  fi

od^{5}



~~Simple Loops~~

Example

int : x_1, x_2

while $^{1}(x_1 \geq 0 \wedge x_2 \geq 0)$ do

if $^{2}(?)$ then

$^{3}x_1 := x_1 - 1$

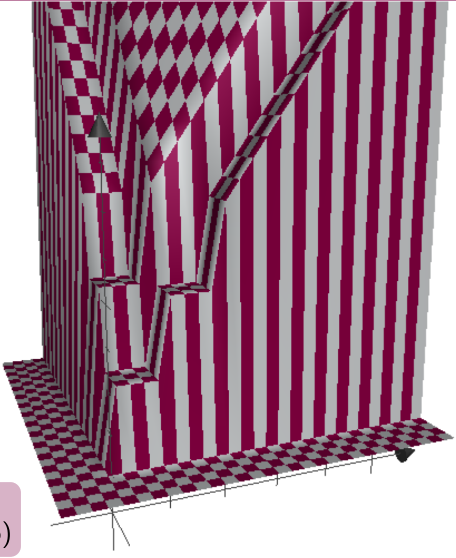
else

$^{4}x_2 := x_2 - 1$

fi

od^{5}

Cook&Podelski&Rybalchenko - *Terminator: Beyond Safety* (CAV 2006)



Sufficient Preconditions for Termination

Example

int : x

while [1]$(x < 10)$ do

  [2]$x := 2 * x$

od[3]

$$f(x) = \begin{cases} 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

$$f(x) = \begin{cases} 9 & x = 1 \\ 7 & x = 2 \\ 5 & 3 \leq x \leq 4 \\ 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

Sufficient Preconditions for Termination

Example

int : x

while [1]$(x < 10)$ do

[2]$x := 2 * x$

od[3]

$$f(x) = \begin{cases} 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

$$f(x) = \begin{cases} 9 & x = 1 \\ 7 & x = 2 \\ 5 & 3 \leq x \leq 4 \\ 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

Sufficient Preconditions for Termination

Example

int : x

while [1]$(x < 10)$ do

[2]$x := 2 * x$

od[3]

$$f(x) = \begin{cases} 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

$$f(x) = \begin{cases} 9 & x = 1 \\ 7 & x = 2 \\ 5 & 3 \leq x \leq 4 \\ 3 & 5 \leq x \leq 9 \\ 1 & 10 \leq x \end{cases}$$

Cook&al - *Proving Conditional Termination* (CAV 2008) http://www.di.ens.fr/~urban/FuncTion.html

- written in OCaml
- implemented on top of Apron[4]

- forward reachability analysis to improve precision

Example

int : x_1, x_2
[1]$x_2 := 1$
while [2]$(x_1 < 10)$ do
  [3]$x_1 := x_1 + x_2$
od[4]

[4] http://apron.cri.ensmp.fr/library/

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

- written in OCaml
- implemented on top of Apron[4]

- forward reachability analysis to improve precision

Example

int : x_1, x_2

${}^{1}x_2 := 1$

while ${}^{2}(x_1 < 10)$ do

  ${}^{3}x_1 := x_1 + x_2$

od^{4}

[4] http://apron.cri.ensmp.fr/library/

Conclusions

- **family of** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
- instance based on **intervals** and **affine functions**
  - segmentation overcomes non-existence of linear ranking functions
  - analysis not limited to simple loops
  - sufficient conditions for termination

Future Work

- **more abstract domains** (e.g. non-linear functions)
- other liveness properties
- cost analysis
- non-termination

Conclusions

- **family of** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
- instance based on **intervals** and **affine functions**
  - segmentation overcomes non-existence of linear ranking functions
  - analysis not limited to simple loops
  - sufficient conditions for termination

Future Work

- **more abstract domains** (e.g. non-linear functions)
- other liveness properties
- cost analysis
- non-termination

Conclusions

- **family of** parameterized **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward invariance analysis
- instance based on **intervals** and **affine functions**
  - segmentation overcomes non-existence of linear ranking functions
  - analysis not limited to simple loops
  - sufficient conditions for termination

Future Work

- **more abstract domains** (e.g. non-linear functions)
- other liveness properties
- cost analysis
- non-termination

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)