Conflict-Driven Conditional Termination

Vijay D'Silva — Google

Caterina Urban — ENS, Carnegie Mellon University

July 24th, 2015
CAV 2015
San Francisco, USA

Conditional Termination

- **termination**: does the program terminate for all initial states?

- **conditional termination**: for which initial states does the program terminate?

termination $\equiv$ conditional termination for all initial states

Example

int : x

while $^{1}(x < -2 \vee x > 2)$ {

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

$\}^{3}$

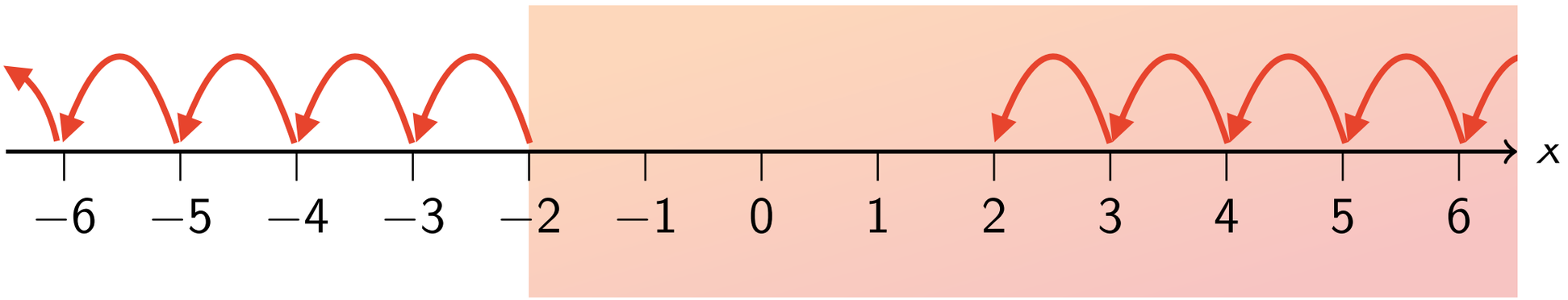
Example

int : x

while $^{1}(x < -2 \vee x > 2)$ {

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

$\}^{3}$



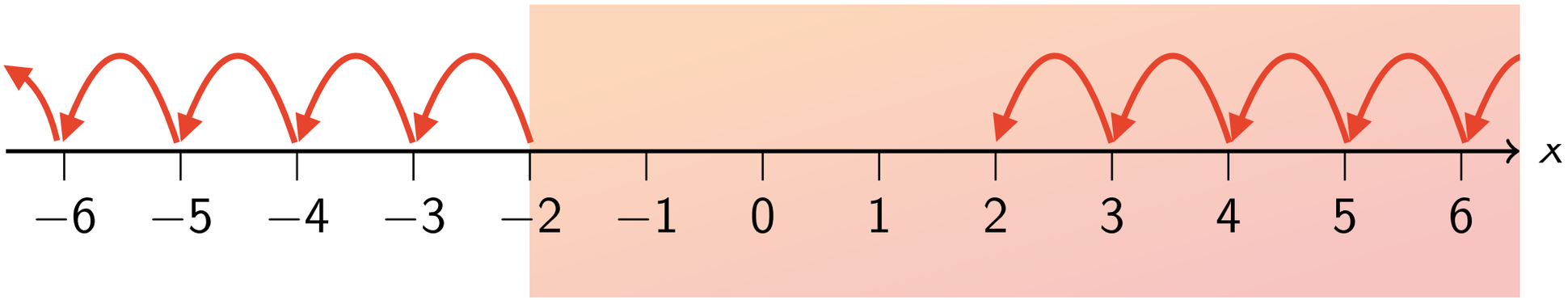
Example

int : x

while $^{1}(x < -2 \vee x > 2)$ {

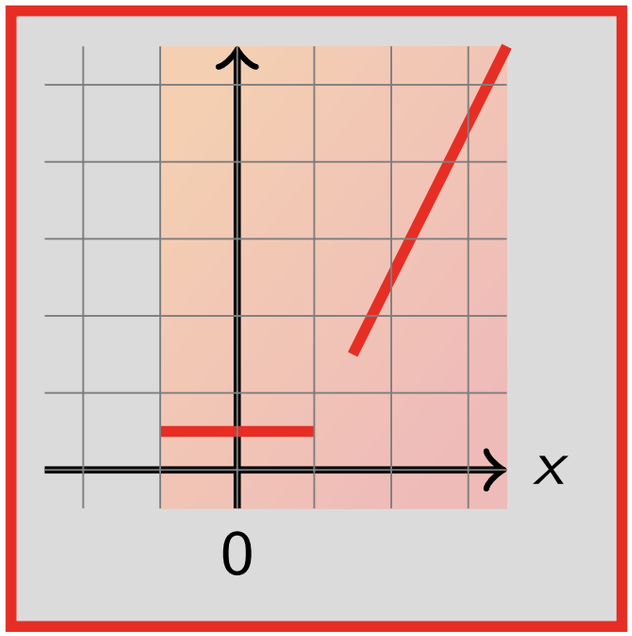
$\quad {}^{2}x := x - 1$

$\}^{3}$

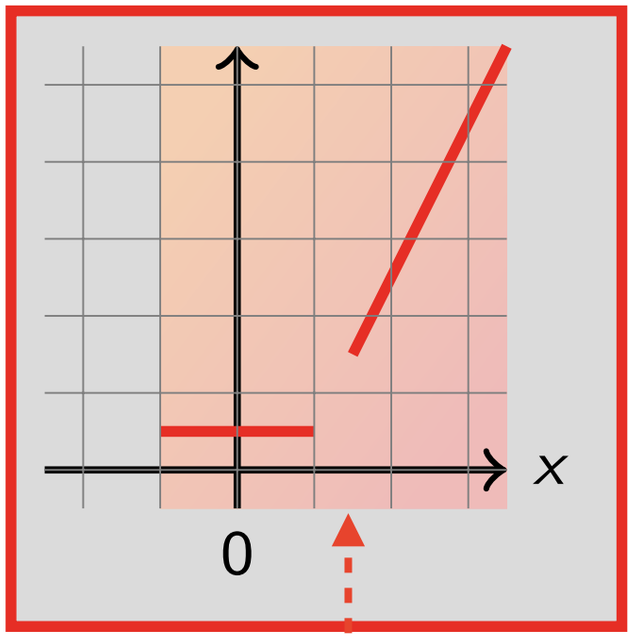


the program **conditionally terminates** for $-2 < x$

Piecewise-Defined Ranking Functions

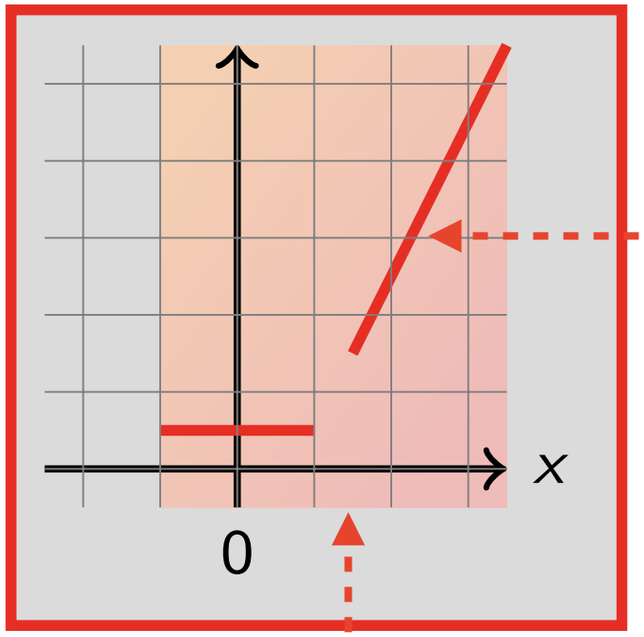


Piecewise-Defined Ranking Functions



the **domain** of the ranking function represents the terminating states

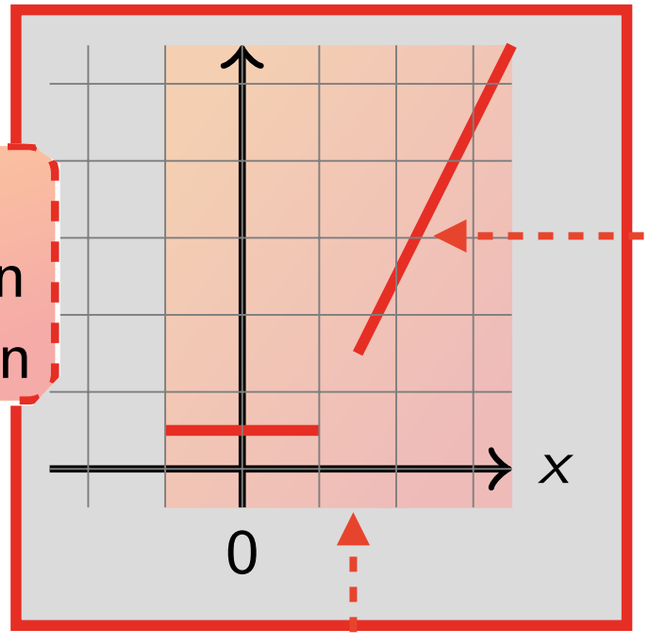
Piecewise-Defined Ranking Functions



the **value** of the ranking function yields an upper bound on the number of execution steps to termination

the **domain** of the ranking function represents the terminating states

Piecewise-Defined Ranking Functions



outside the domain of the ranking function we have no information

the **value** of the ranking function yields an upper bound on the number of execution steps to termination

the **domain** of the ranking function represents the terminating states

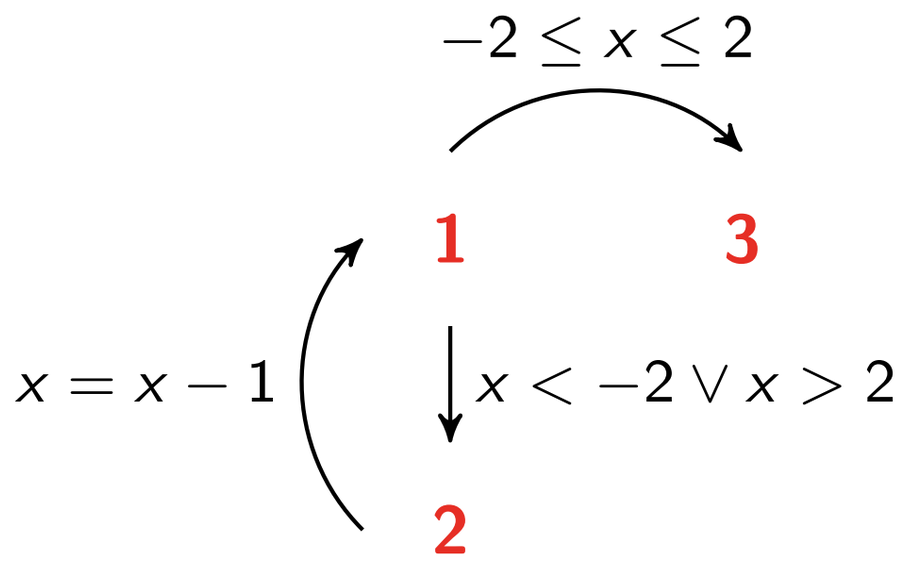
Example

int : x

while $^{1}(x < -2 \vee x > 2)$ {

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

$\}^{3}$



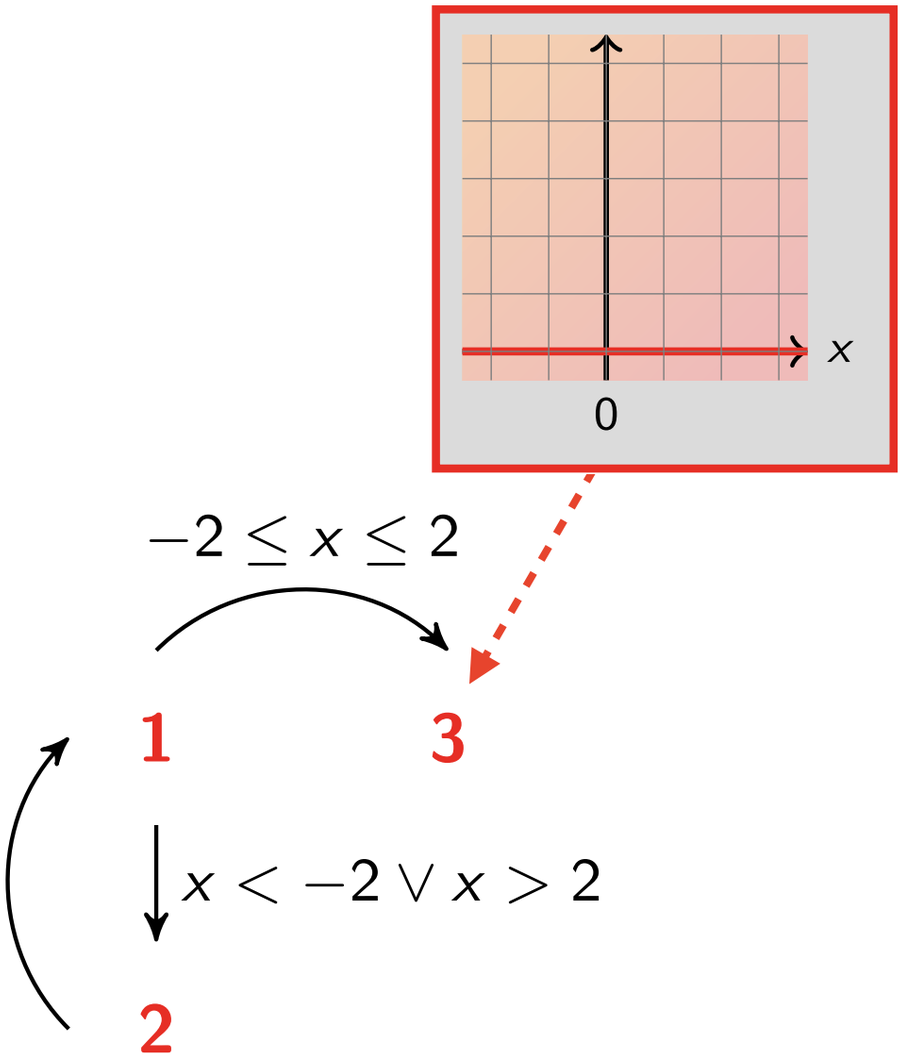
Example

int : x

while [1]$(x < -2 \vee x > 2)$ {

  [2]$x := x - 1$

}[3]



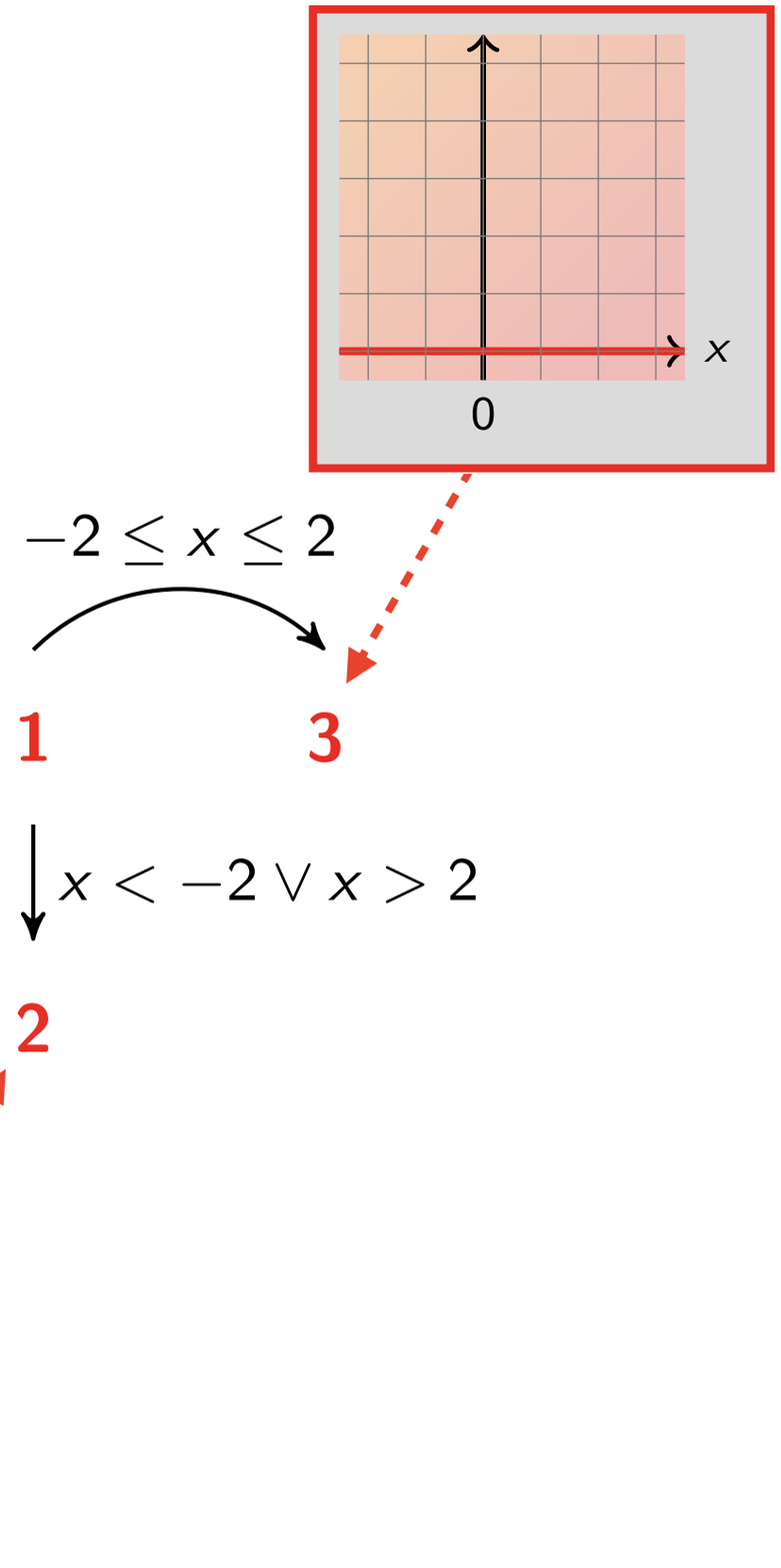
Example

int : x

while [1]$(x < -2 \vee x > 2)$ {

  [2]$x := x - 1$

}[3]



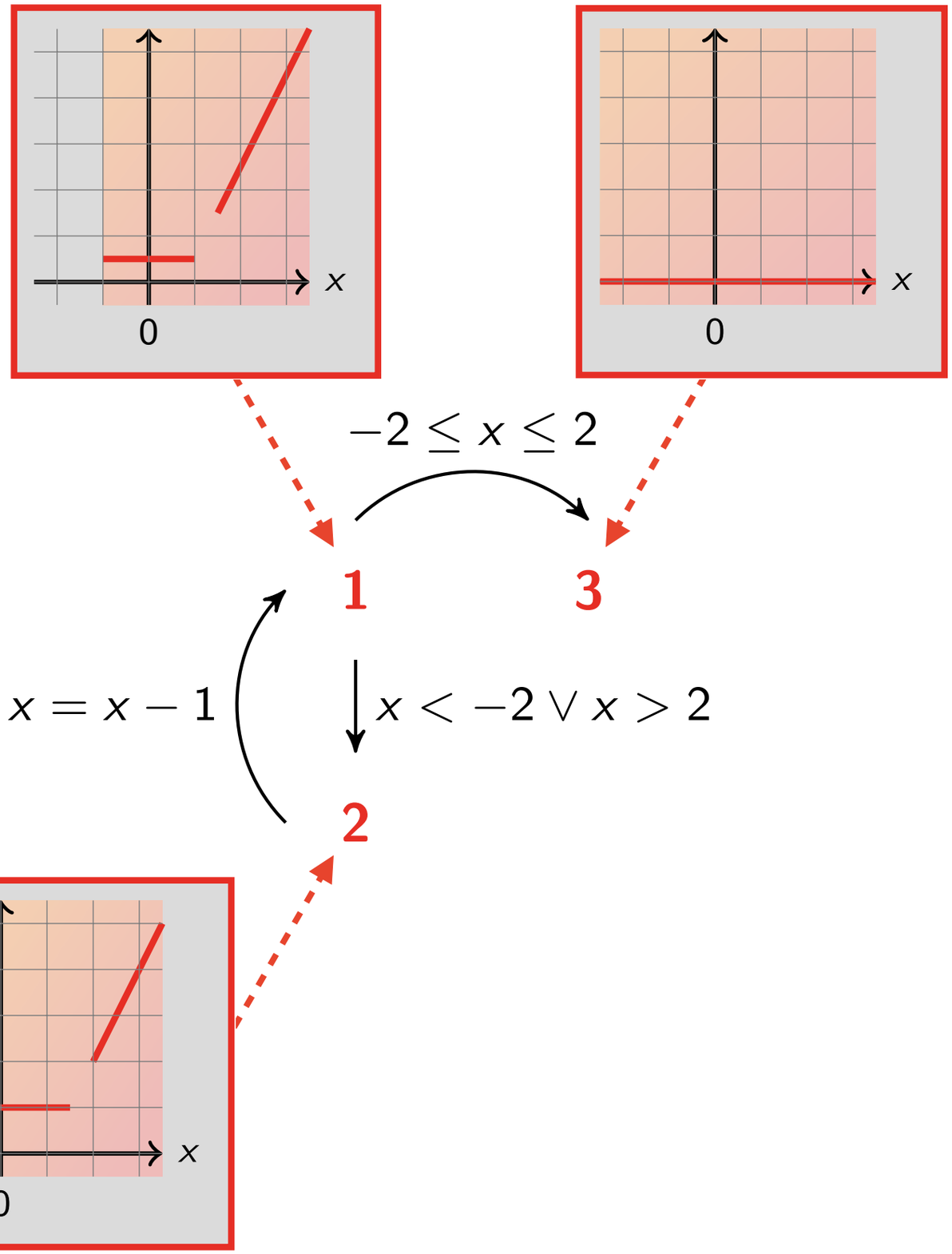
Example

int : x

while [1]$(x < -2 \vee x > 2)$ {

  [2]$x := x - 1$

}[3]



Example

int : x, $step$

if $^{1}(x < -10)$ {

  $^{2}step := 2$

} else if $^{3}(x < 0)$ {

  $^{4}step := 1$

} else if $^{5}(x > 10)$ {

  $^{6}step := -2$

} else {

  $^{7}step := -1$

}

while $^{8}(x < -2 \vee x > 2)$ {

  $^{9}x := x + step$

}^{10}

Example

int : x, $step$

if $^{1}(x < -10)$ {

  $^{2}step := 2$

} else if $^{3}(x < 0)$ {

  $^{4}step := 1$

} else if $^{5}(x > 10)$ {

  $^{6}step := -2$

} else {

  $^{7}step := -1$

}

while $^{8}(x < -2 \vee x > 2)$ {

  $^{9}x := x + step$

}^{10}

different *step* values
on different **branches**

Example

int : x, $step$

if $^{1}(x < -10)$ {

  $^{2}step := 2$

} else if $^{3}(x < 0)$ {

  $^{4}step := 1$

} else if $^{5}(x > 10)$ {

  $^{6}step := -2$

} else {

  $^{7}step := -1$

}

while $^{8}(x < -2 \vee x > 2)$ {

  $^{9}x := x + step$

}^{10}

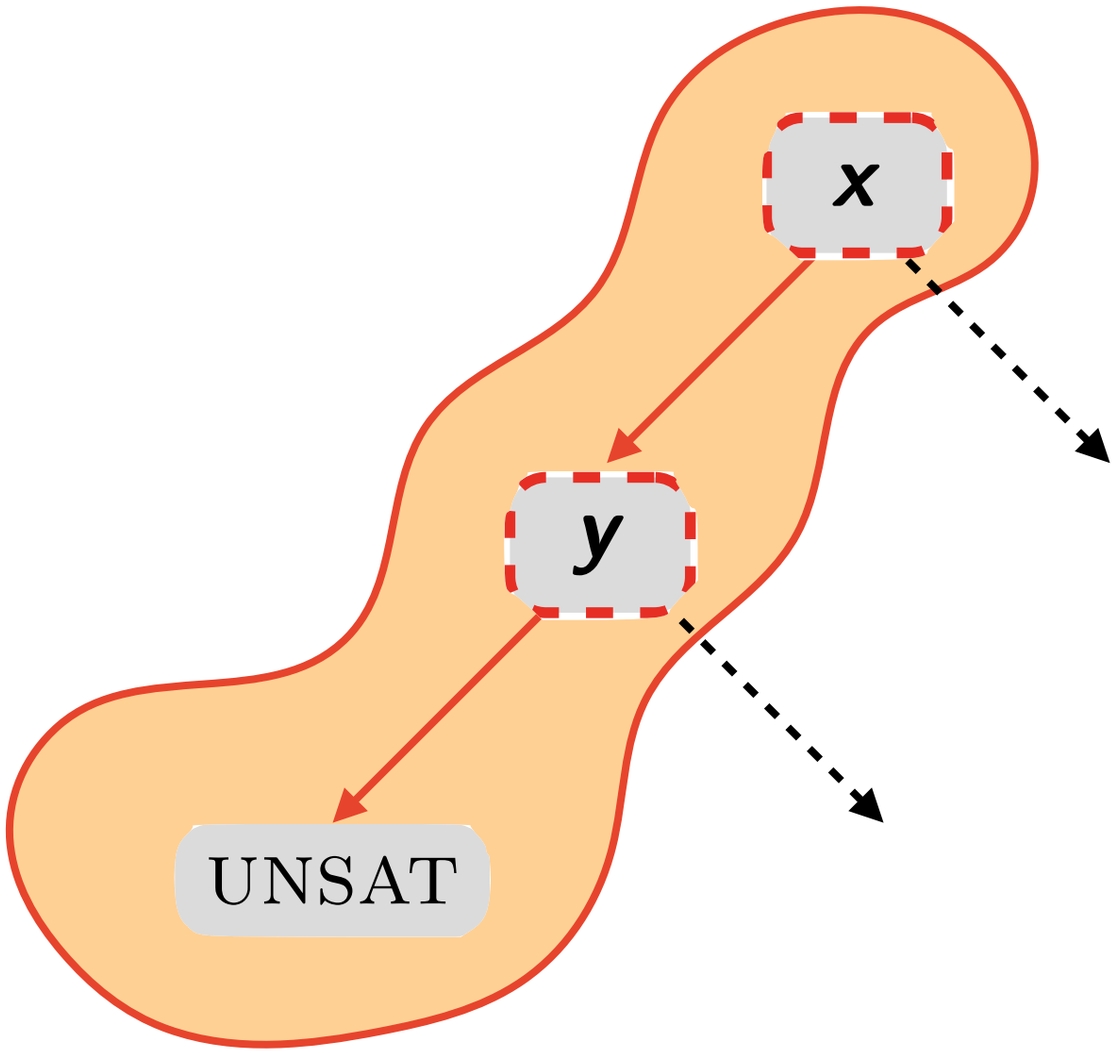
different *step* values on different **branches**

the analysis **fails**

$$(x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$$

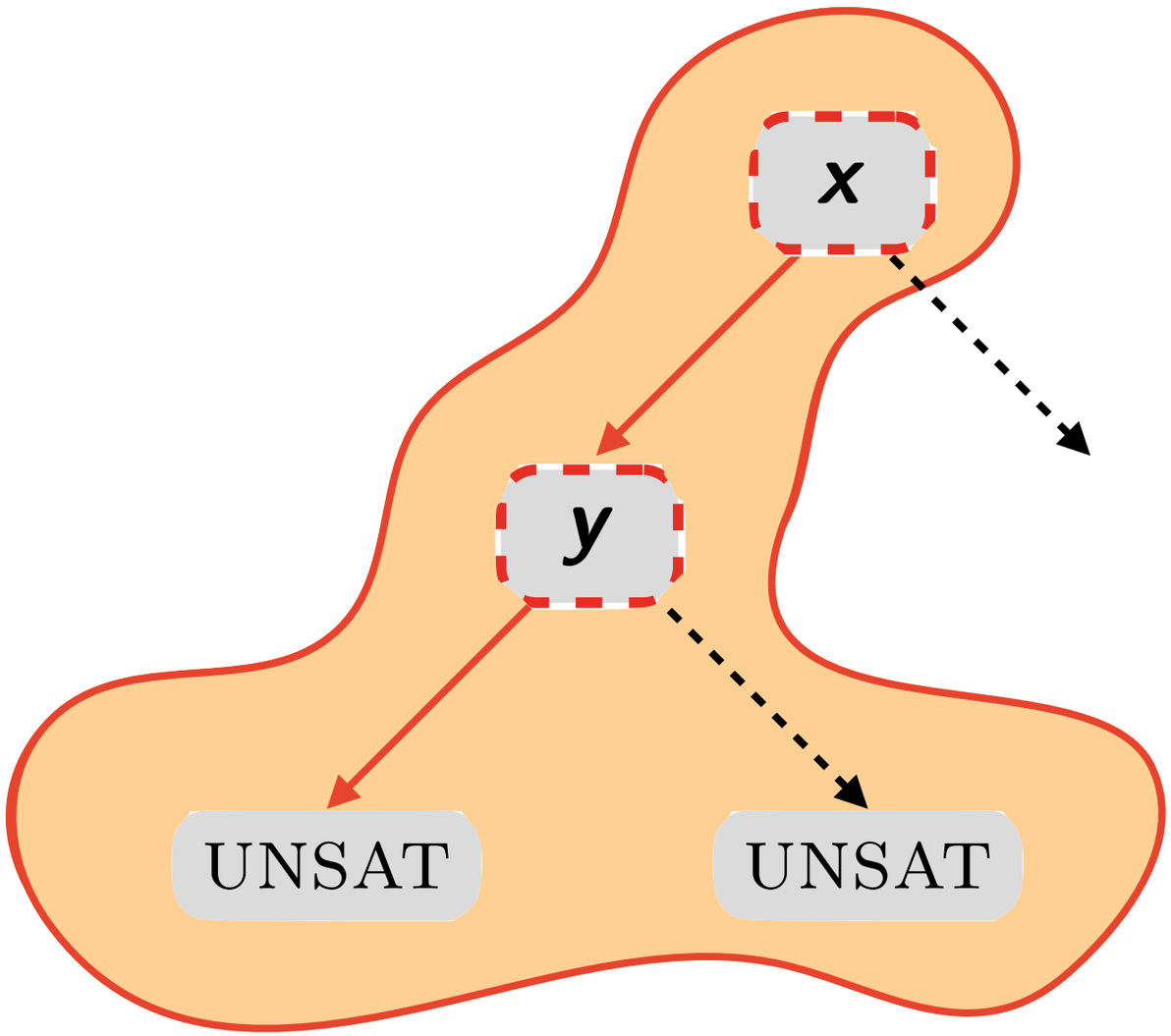
Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)

$$(x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$$



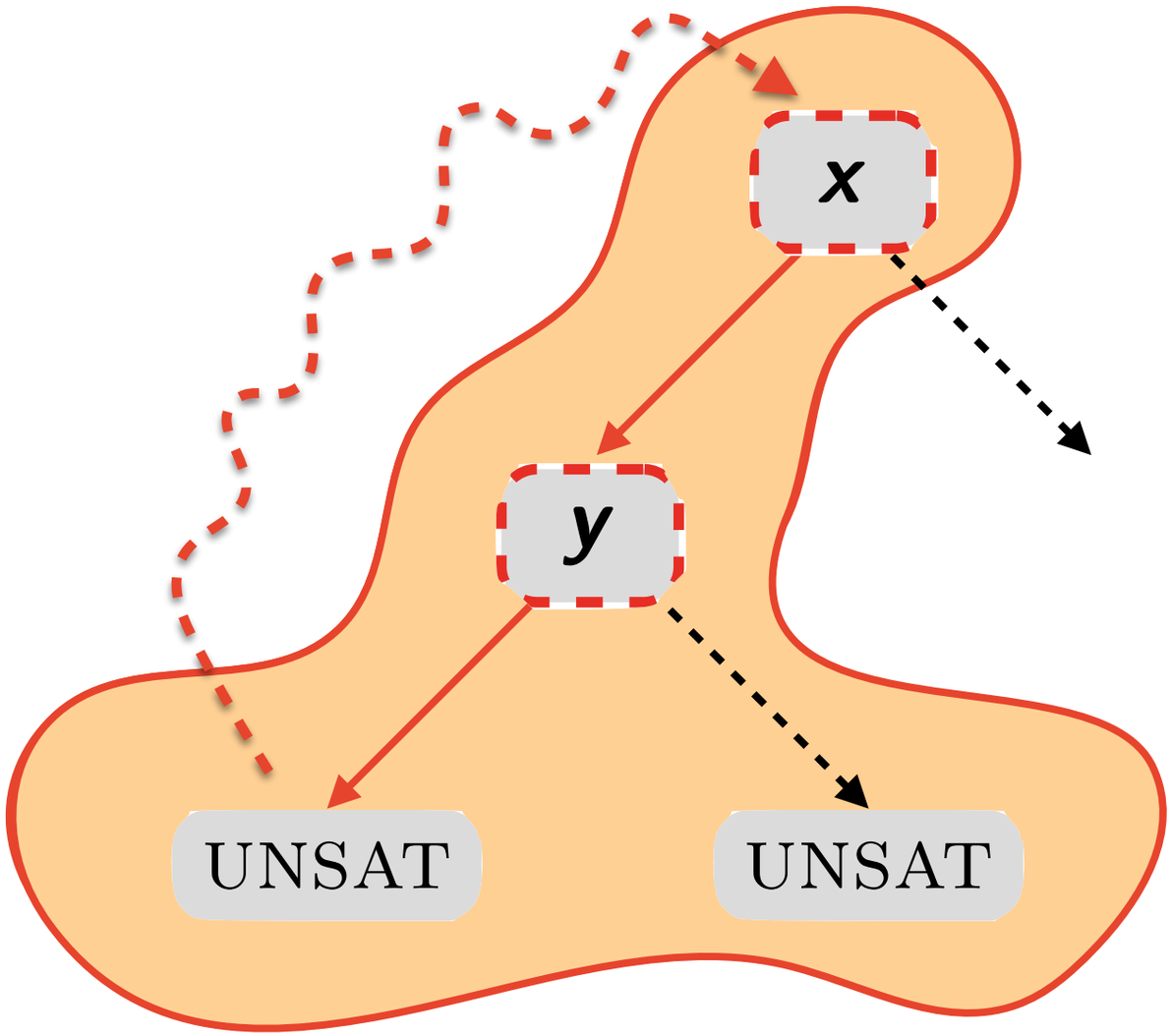
Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)

$$(x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$$



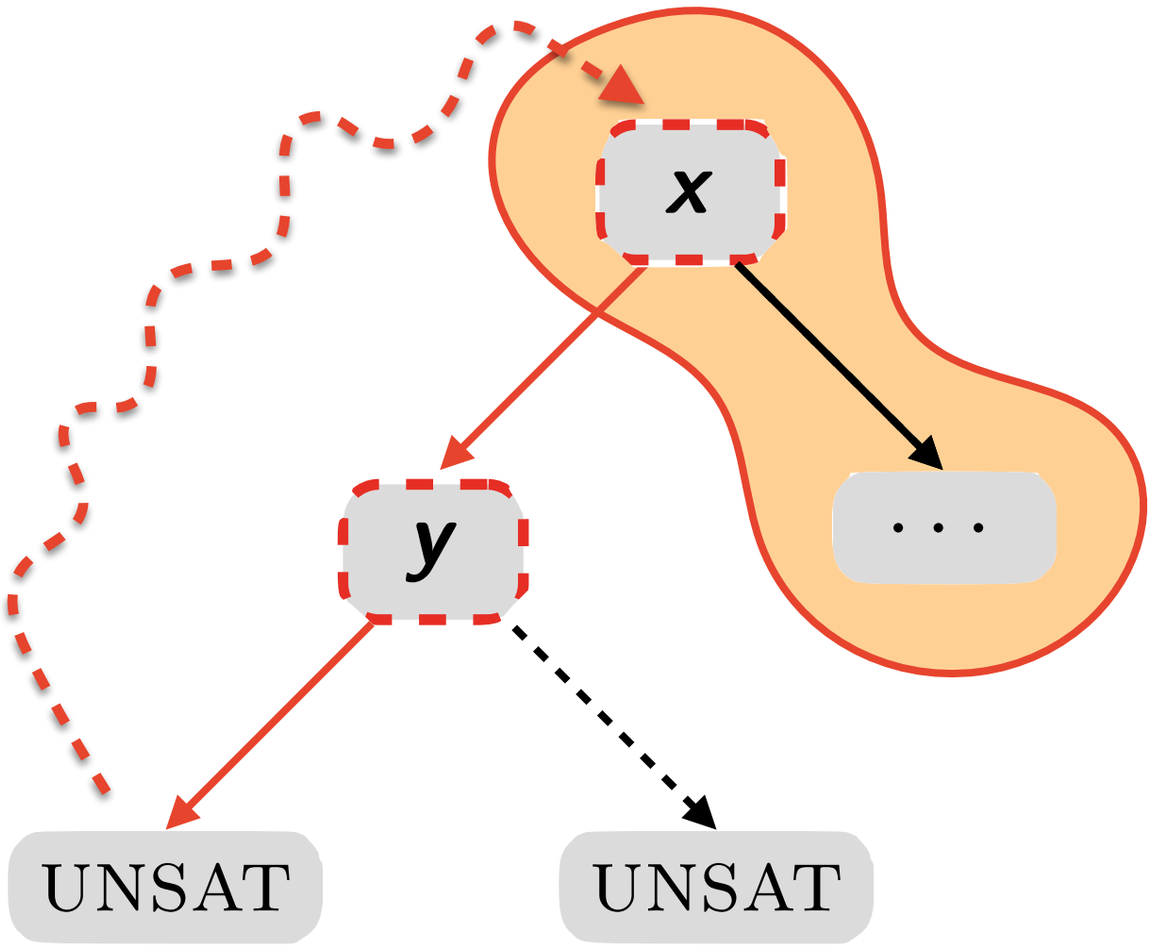
Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)

$$(x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$$

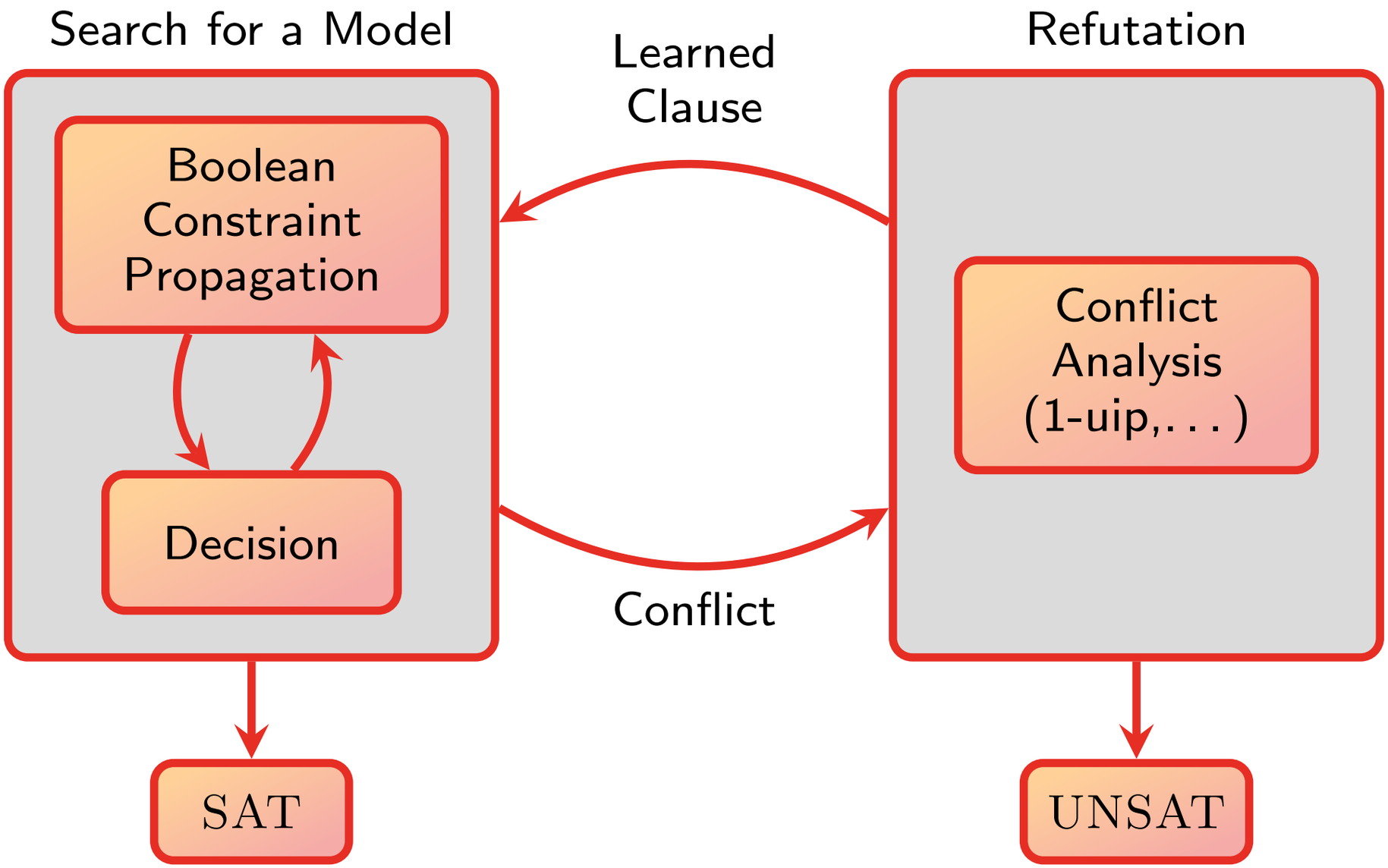


Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)

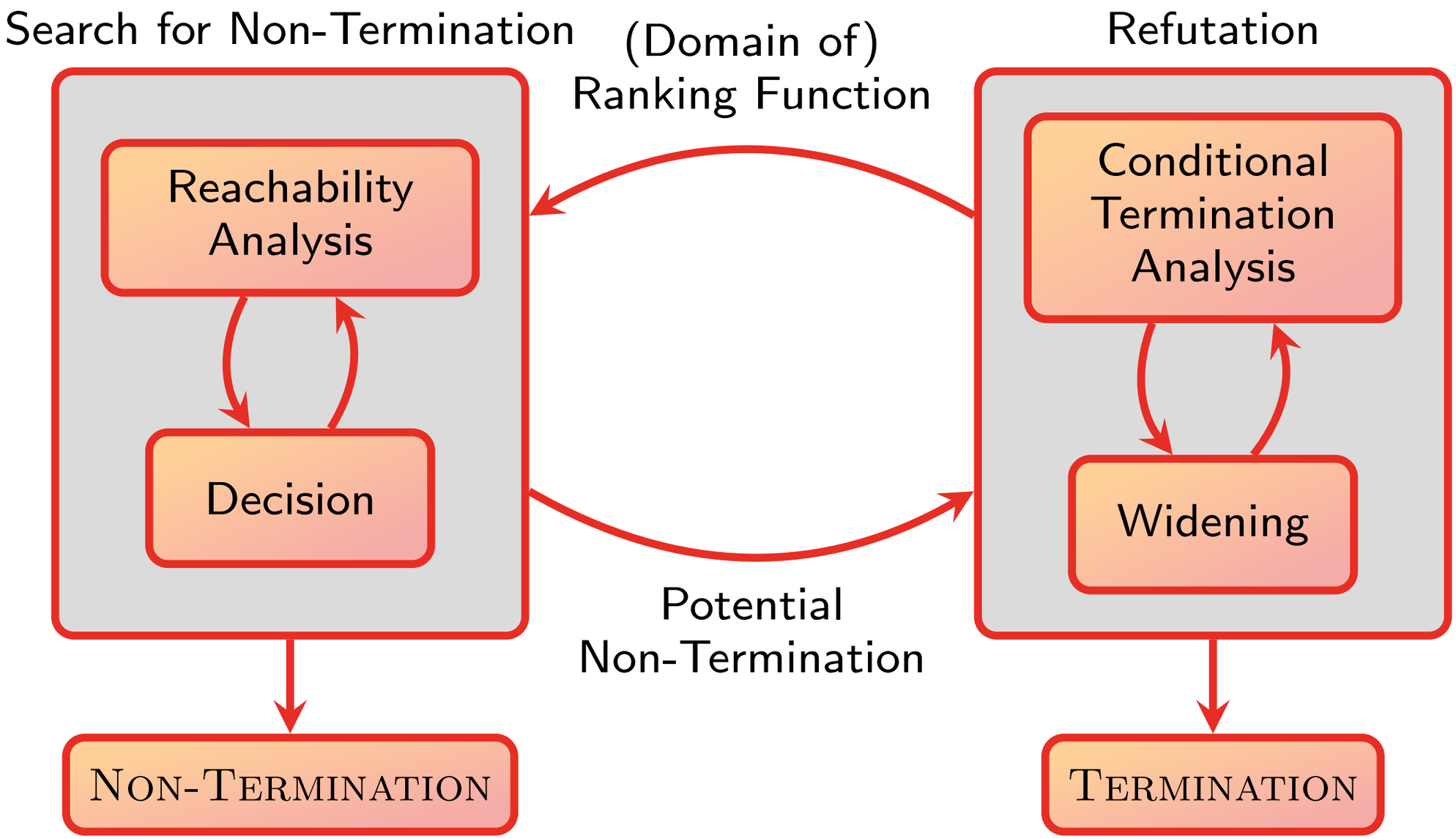
$$(x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z)$$

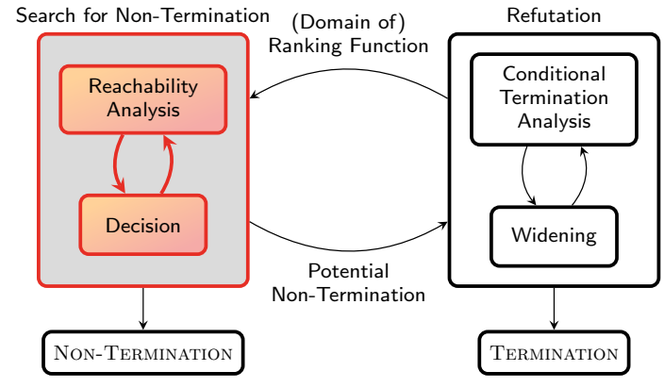


Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)



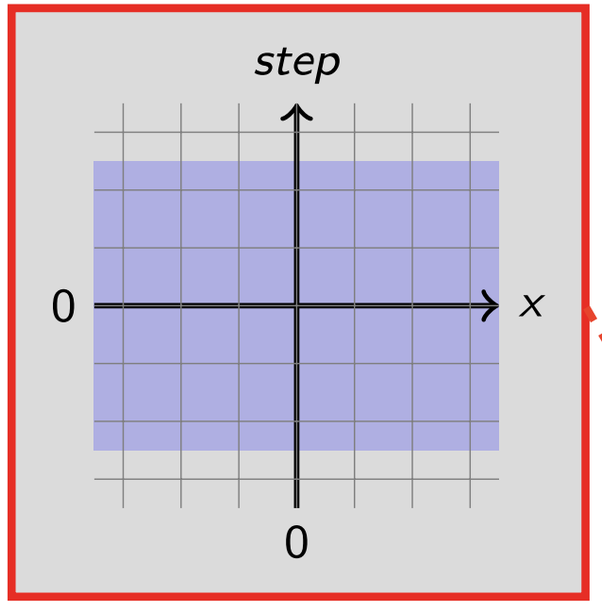
Bayardo & Schrag - Using CSP Look-Back Techniques to Solve Real World SAT Instances (1997)
Marques-Silva & Sakallah - GRASP: A Search Algorithm for Propositional Satisfiability (1999)


Search for Non-Termination
(Domain of)
Ranking Function
Refutation
Reachability
Analysis
Decision
Conditional
Termination
Analysis
Widening
Potential
Non-Termination
Non-Termination
Termination

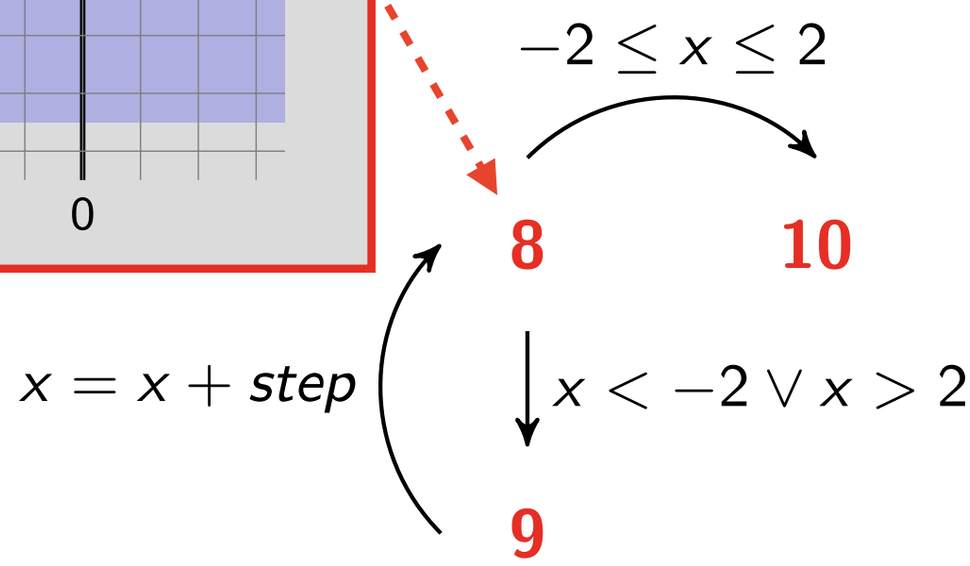


Example

```
int : x, step
1 if (x < −10) {
    2 step := 2
} else if 3 (x < 0) {
    4 step := 1
} else if 5 (x > 10) {
    6 step := −2
} else {
    7 step := −1
}
while 8 (x < −2 ∨ x > 2) {
    9 x := x + step
}10
```



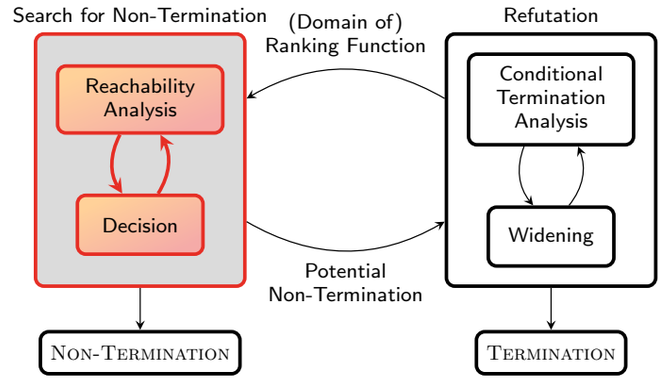
reachable states are potentially non-terminating

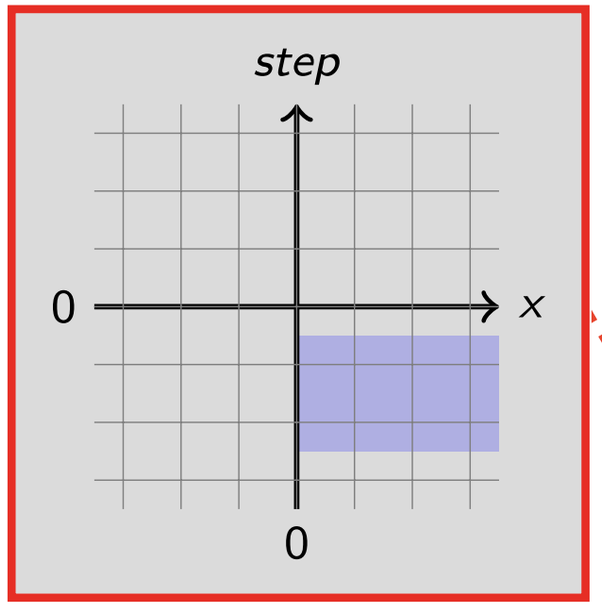


Example

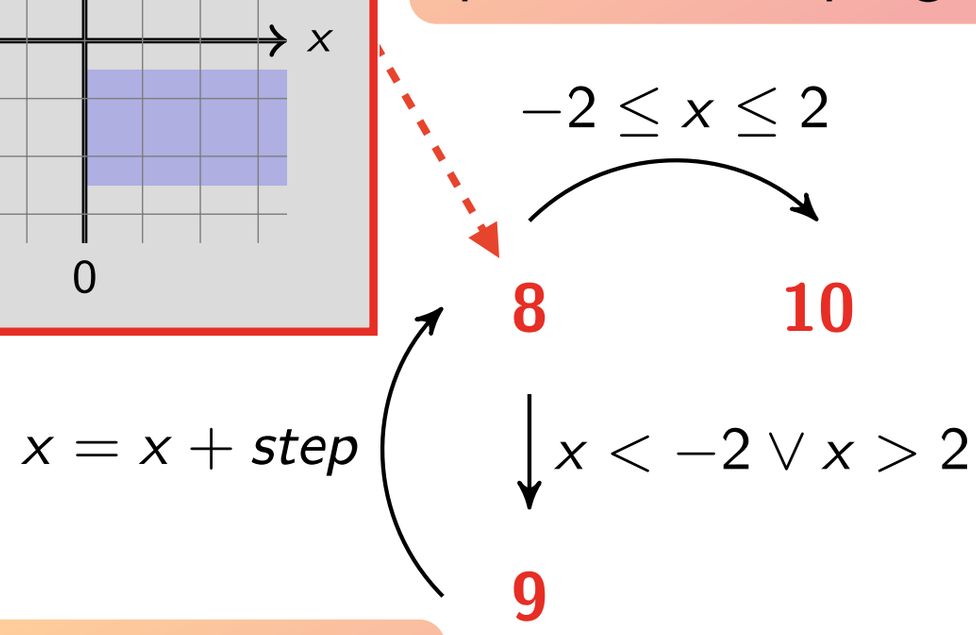
```
int : x, step
if [1](x < −10) {
    [2]step := 2
} else if [3](x < 0) {
    [4]step := 1
} else if [5](x > 10) {
    [6]step := −2
} else {
    [7]step := −1
}
while [8](x < −2 ∨ x > 2) {
    [9]x := x + step
}[10]
```

$x \geq 0$





the consequence of a decision is to focus the analyzer on certain paths of the program



a **decision** restricts an abstract element at a program control point

Example

int : x, $step$

if $^{1}(x < -10)$ {

  $^{2}step := 2$

} else if $^{3}(x < 0)$ {

  $^{4}step := 1$

} else if $^{5}(x > 10)$ {

  $^{6}step := -2$

} else {

  $^{7}step := -1$

}

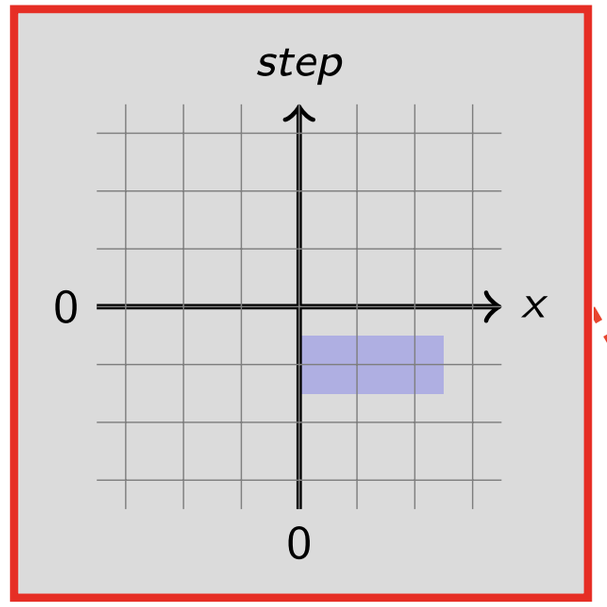
while $^{8}(x < -2 \vee x > 2)$ {

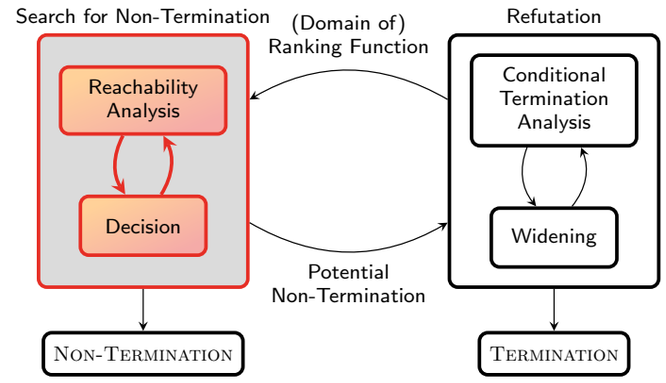
  $^{9}x := x + step$

}^{10}

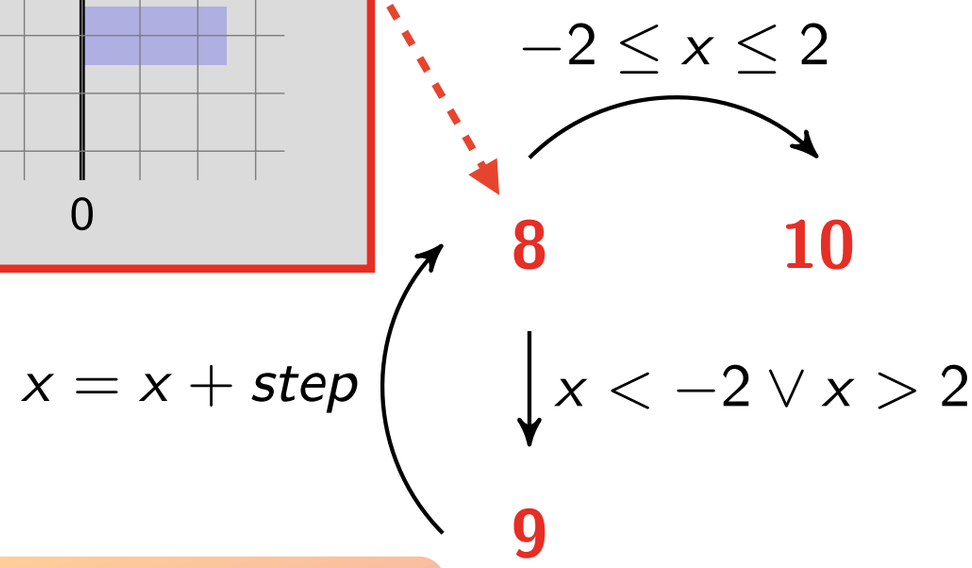
$x \geq 0$

$x \leq 10$

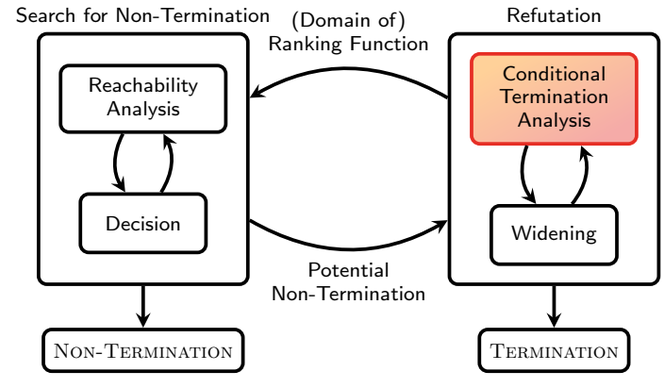




the consequence of a decision is to focus the analyzer on certain paths of the program



a **decision** restricts an abstract element at a program control point



Example

int : x, $step$

[1] if $(x < -10)$ {

  [2] $step := 2$

} else if [3] $(x < 0)$ {

  [4] $step := 1$

} else if [5] $(x > 10)$ {

  [6] $step := -2$

} else {

  [7] $step := -1$

}

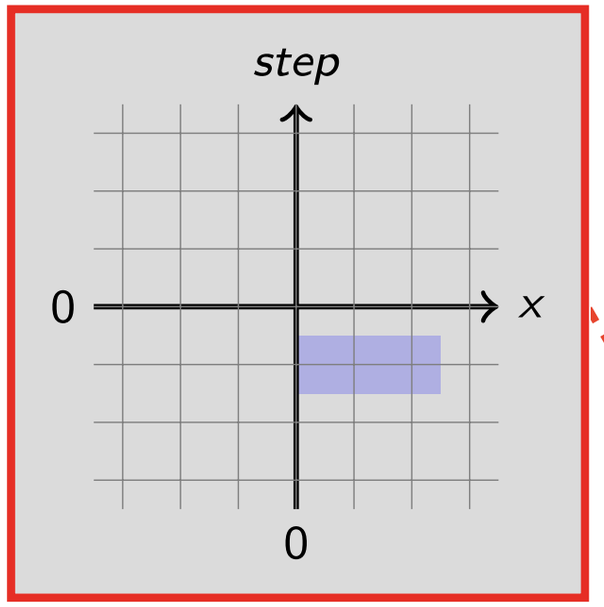
[8] while $(x < -2 \vee x > 2)$ {

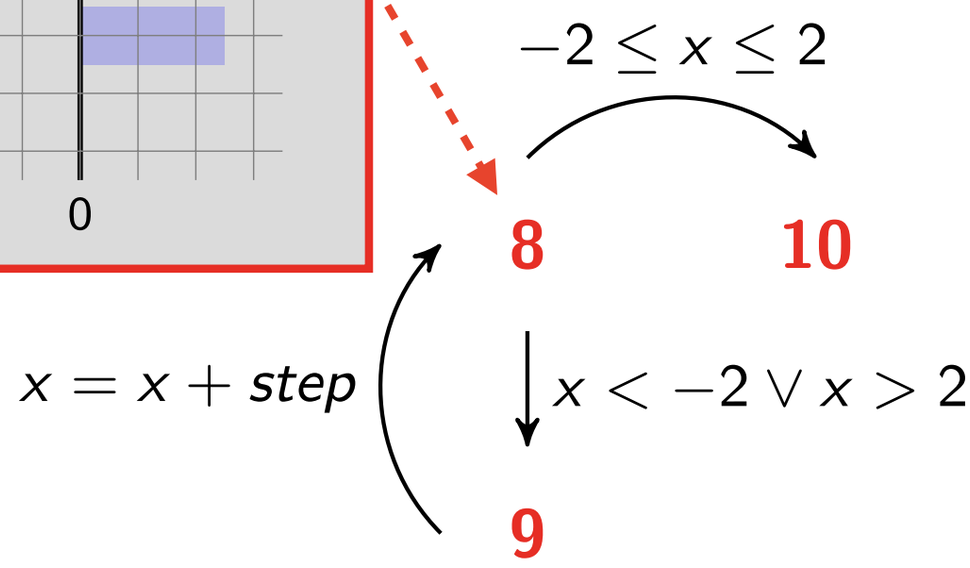
  [9] $x := x + step$

}[10]

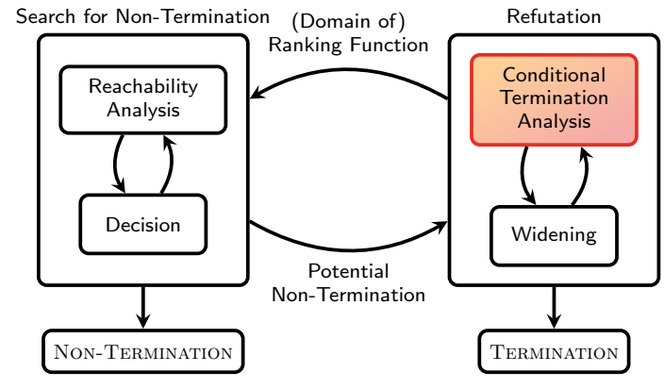
$x \geq 0$

$x \leq 10$





the conditional termination analysis is limited to the states identified by the reachability analysis

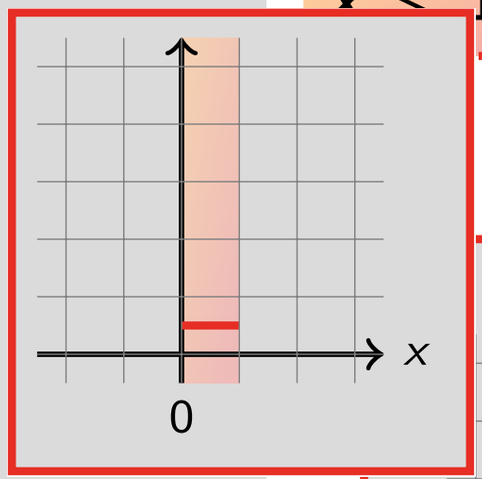


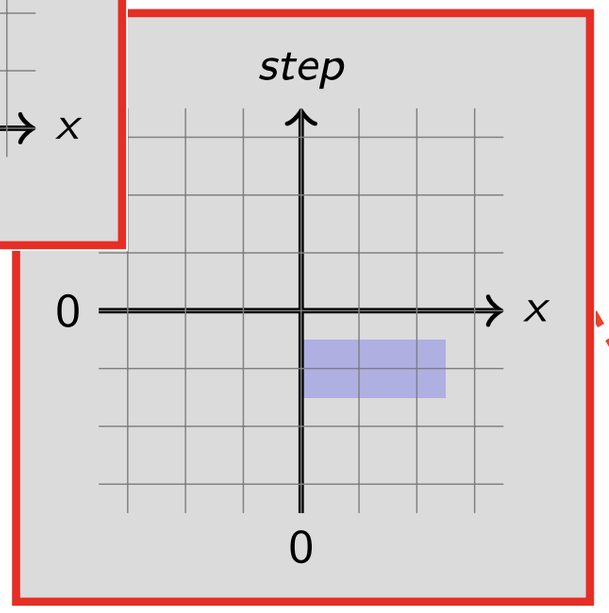
Example

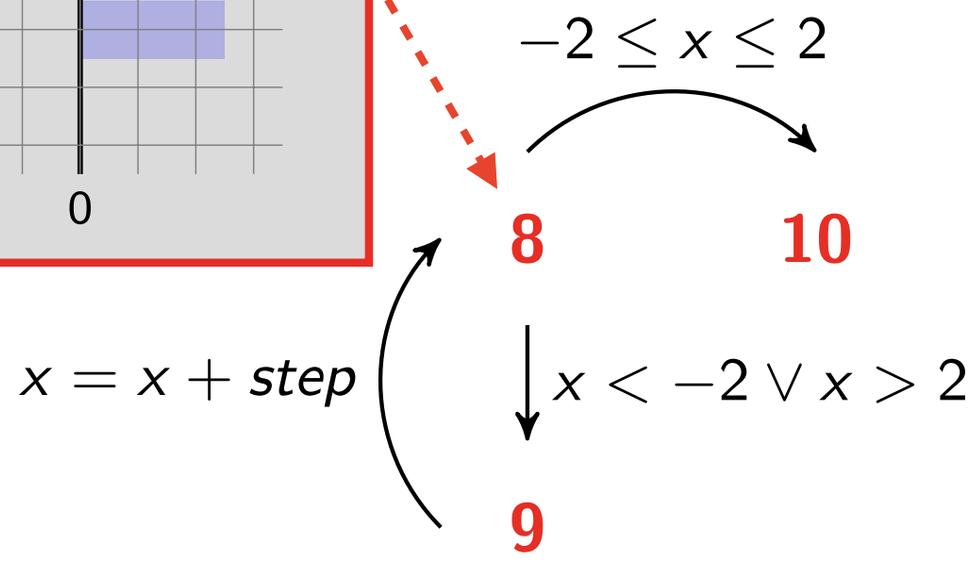
```
int : x, step
if [1](x < −10) {
    [2]step := 2
} else if [3](x < 0) {
    [4]step := 1
} else if [5](x > 10) {
    [6]step := −2
} else {
    [7]step := −1
}
while [8](x < −2 ∨ x > 2) {
    [9]x := x + step
}[10]
```

$x \geq 0$

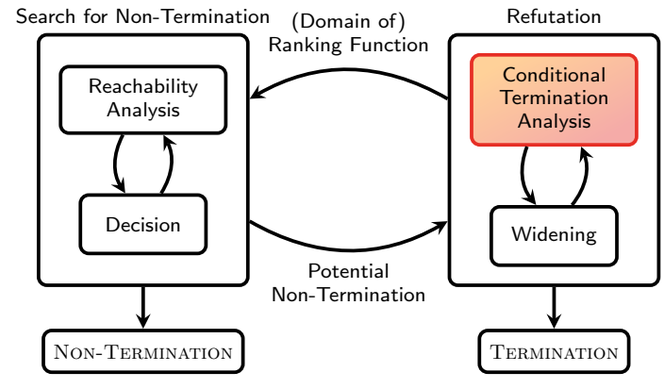
$x < 10$





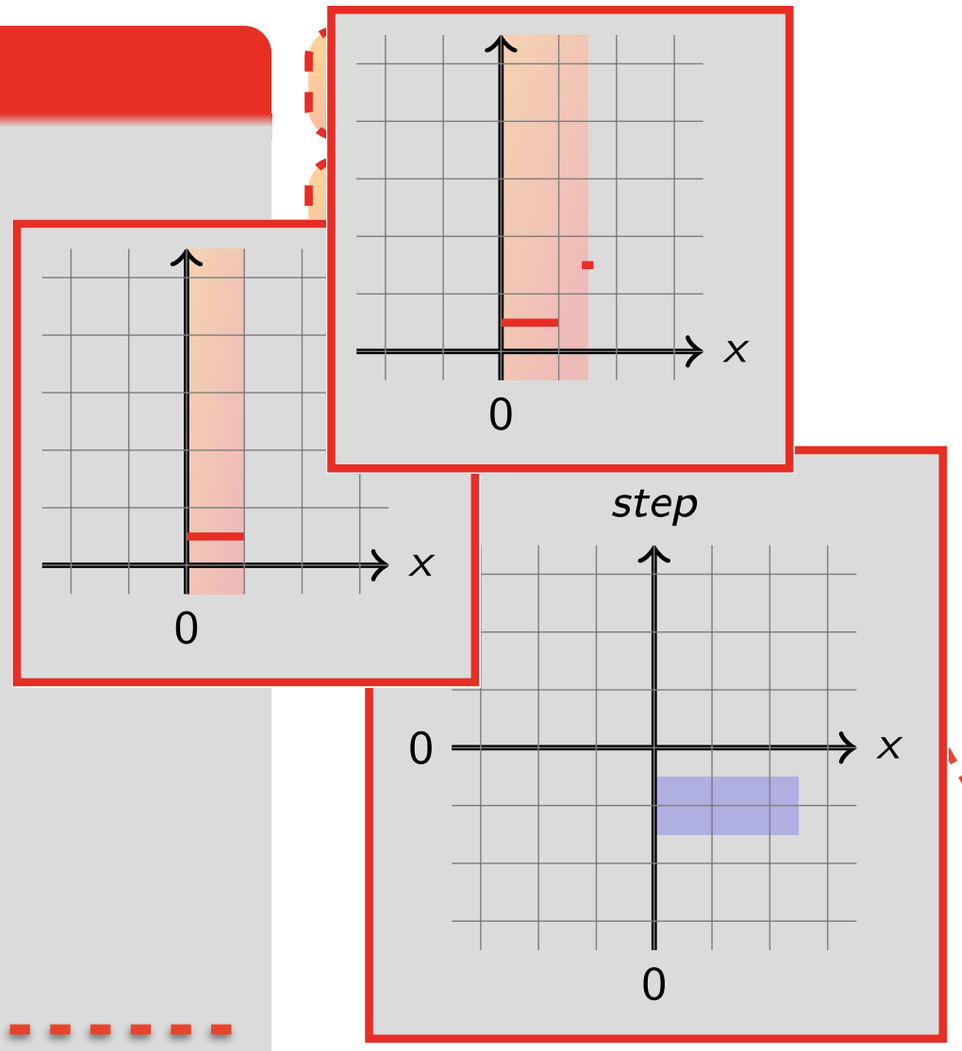


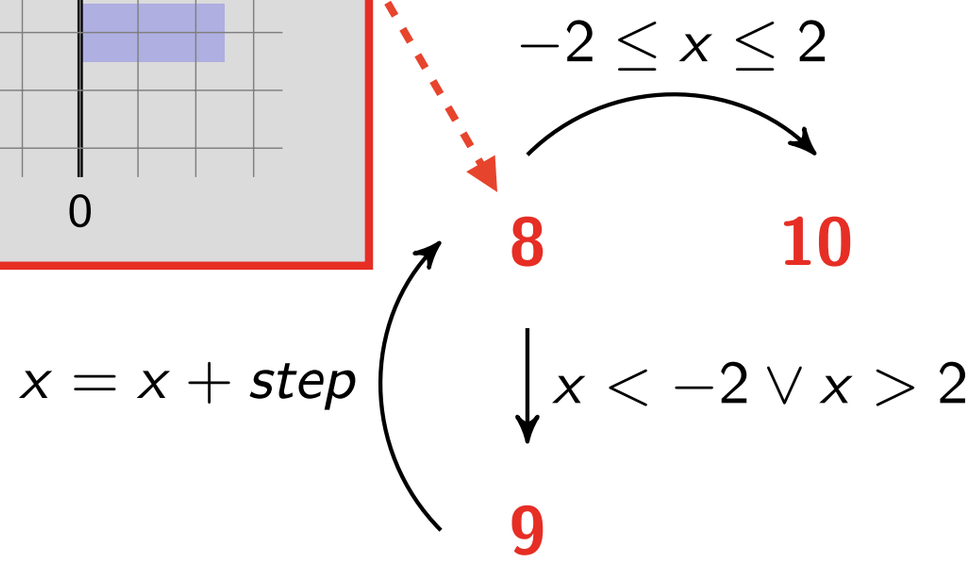
the conditional termination analysis is limited to the states identified by the reachability analysis



Example

```
int : x, step
if [1](x < −10) {
    [2]step := 2
} else if [3](x < 0) {
    [4]step := 1
} else if [5](x > 10) {
    [6]step := −2
} else {
    [7]step := −1
}
while [8](x < −2 ∨ x > 2) {
    [9]x := x + step
}[10]
```





the conditional termination analysis is limited to the states identified by the reachability analysis

Example

int : x, $step$

if [1] $(x < -10)$ {

  [2] $step := 2$

} else if [3] $(x < 0)$ {

  [4] $step := 1$

} else if [5] $(x > 10)$ {

  [6] $step := -2$

} else {

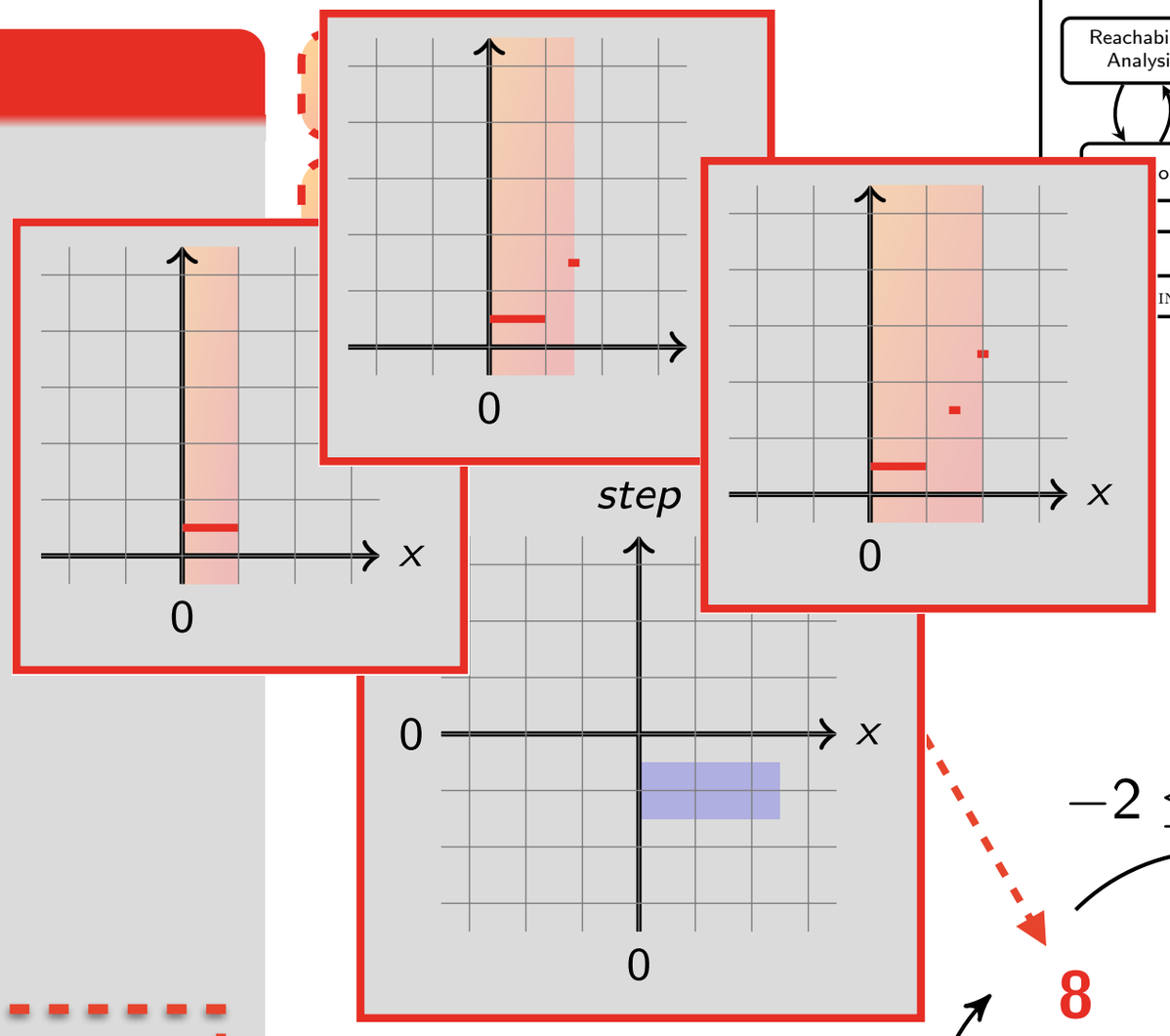
  [7] $step := -1$

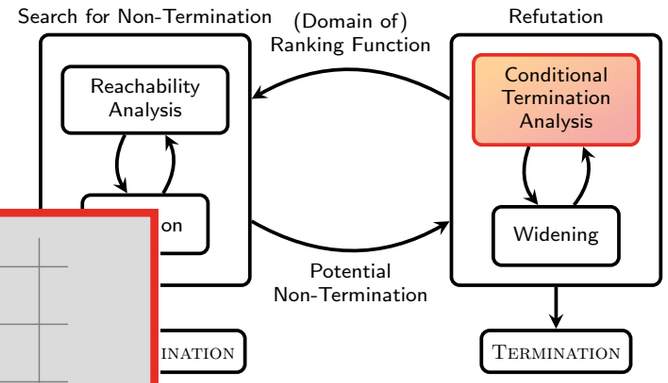
}

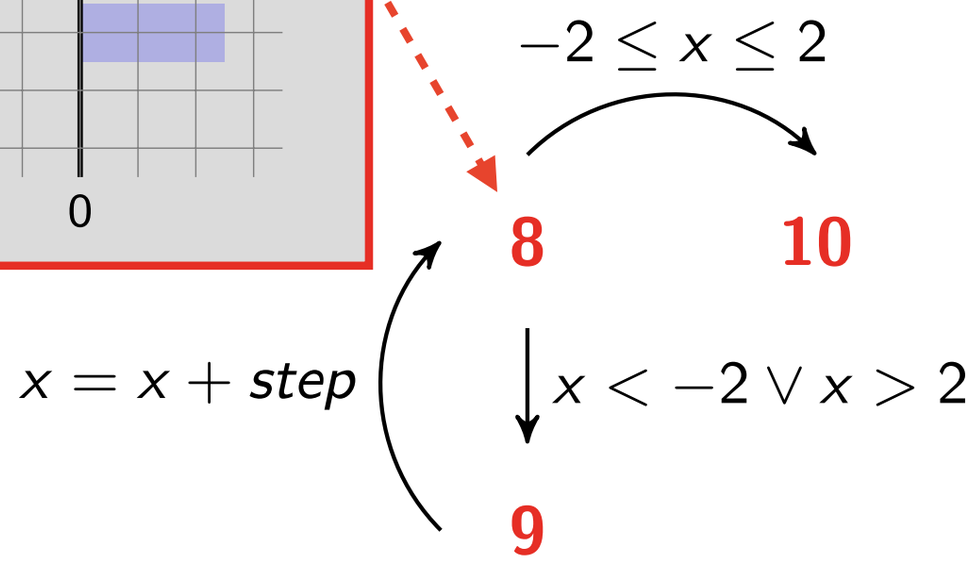
while [8] $(x < -2 \vee x > 2)$ {

  [9] $x := x + step$

}[10]

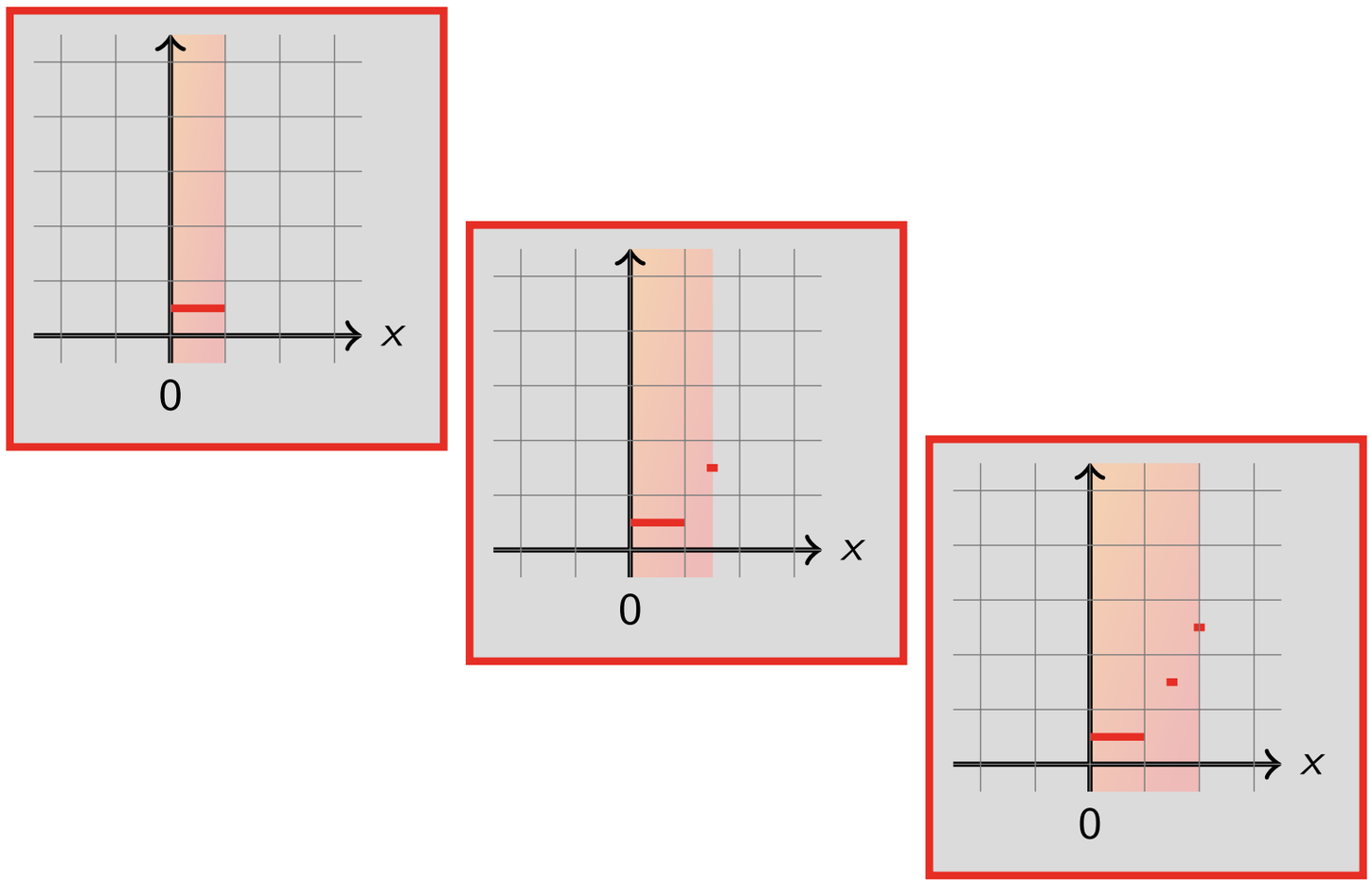






the conditional termination analysis is limited to the states identified by the reachability analysis

Widening



widening is the abstract interpretation approach to generalization

Example

int : x, $step$

if [1]$(x < -10)$ {

  [2]$step := 2$

} else if [3]$(x < 0)$ {

  [4]$step := 1$

} else if [5]$(x > 10)$ {

  [6]$step := -2$

} else {

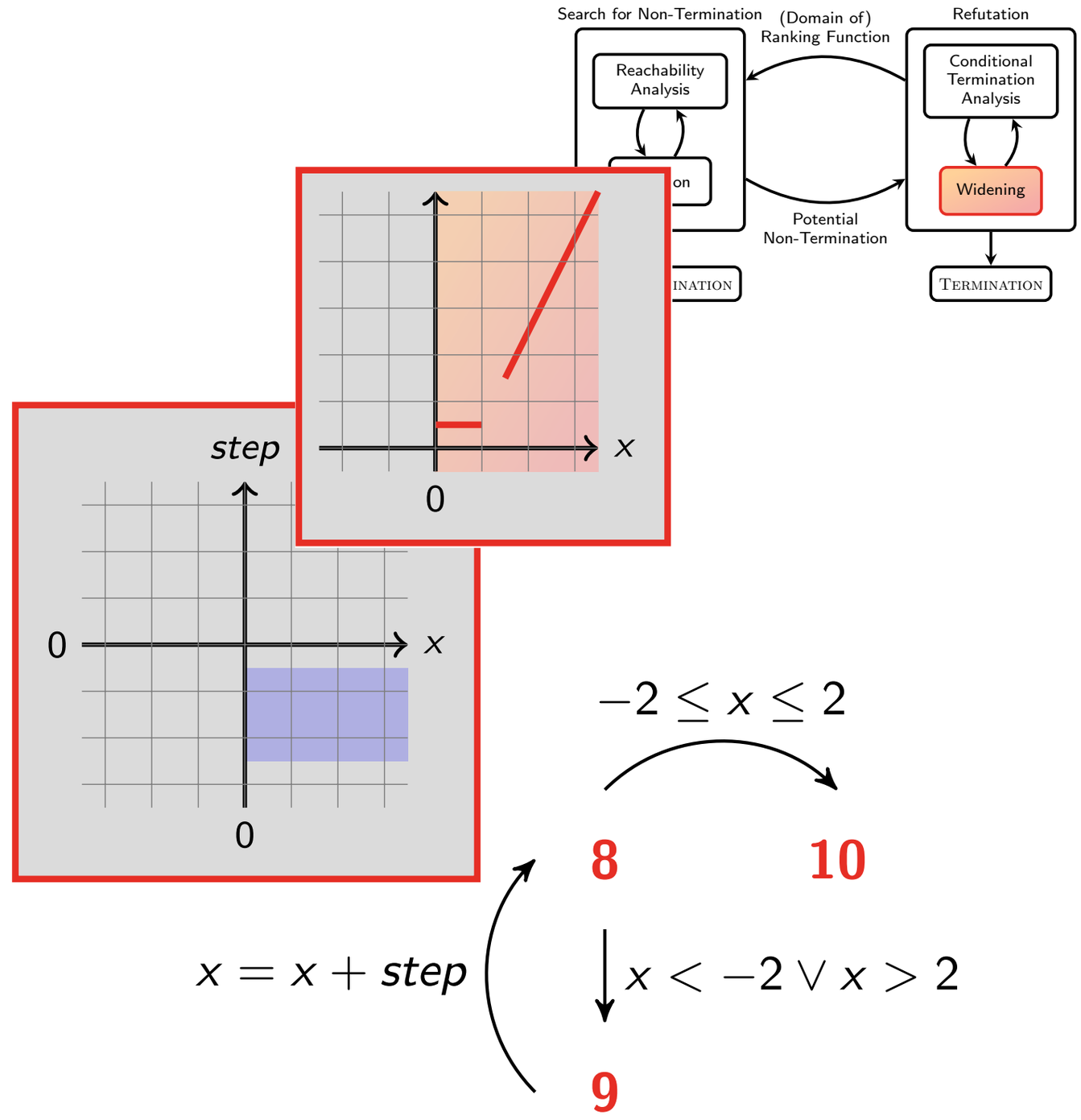
  [7]$step := -1$

}

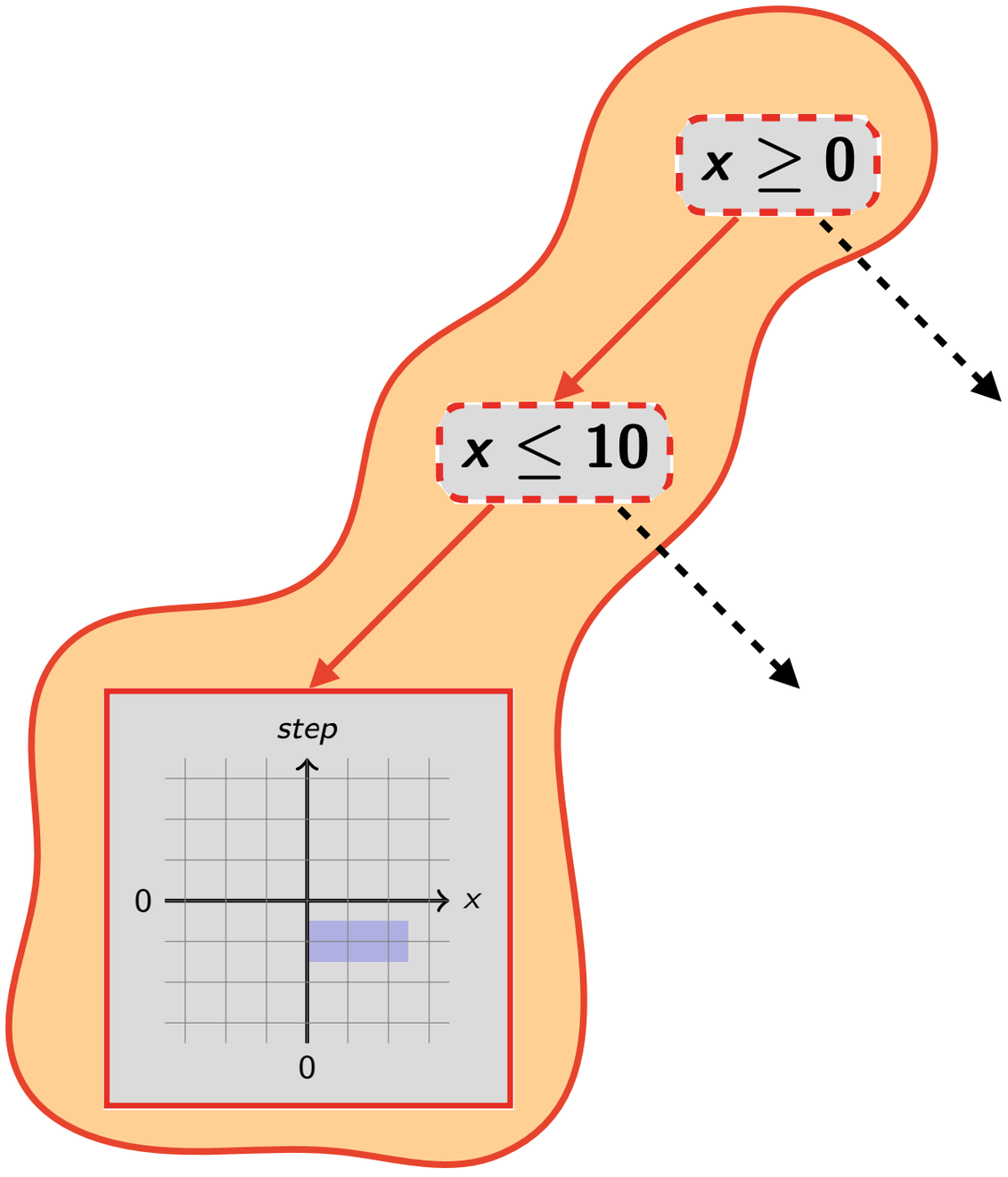
while [8]$(x < -2 \vee x > 2)$ {

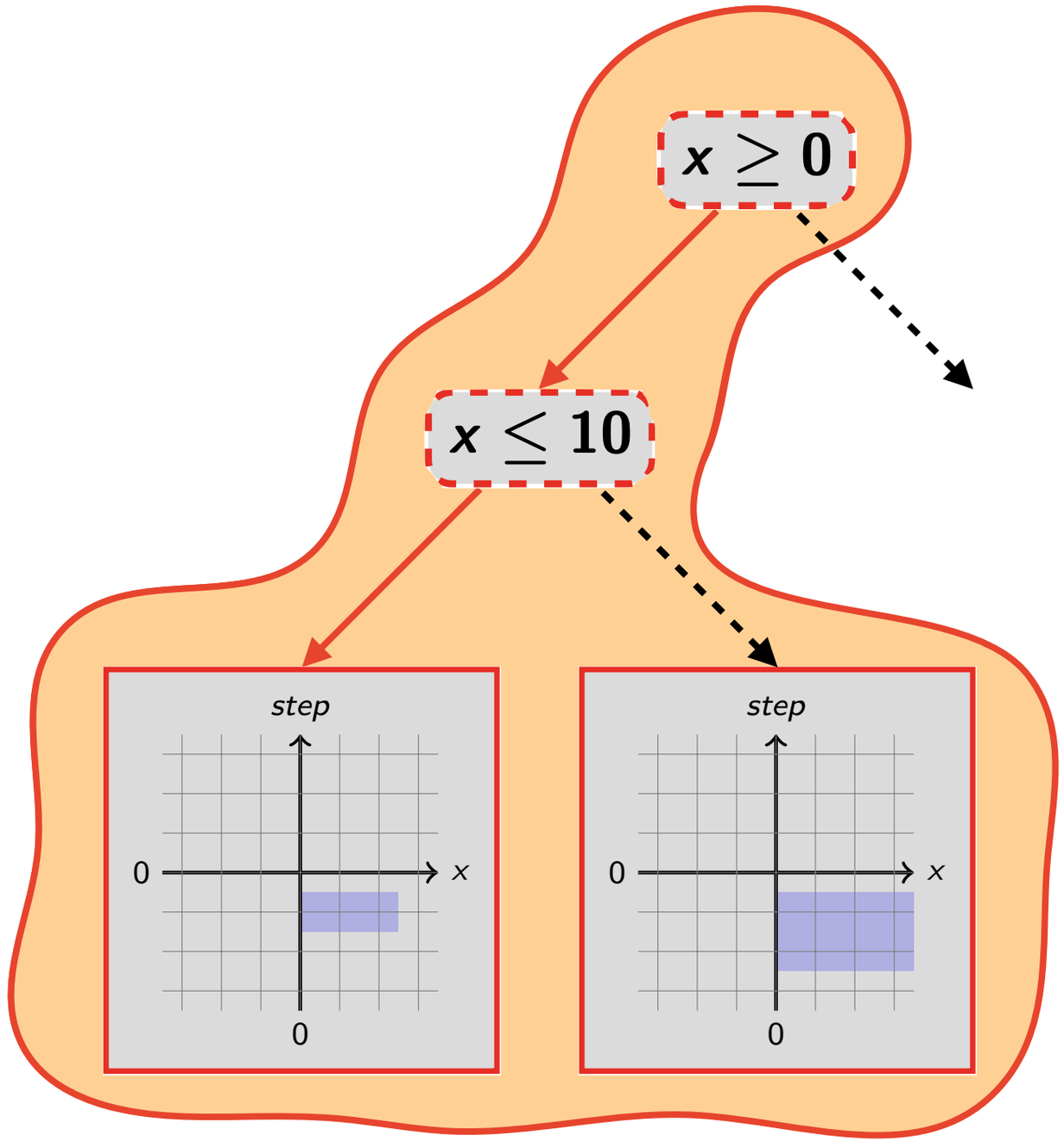
  [9]$x := x + step$

}[10]

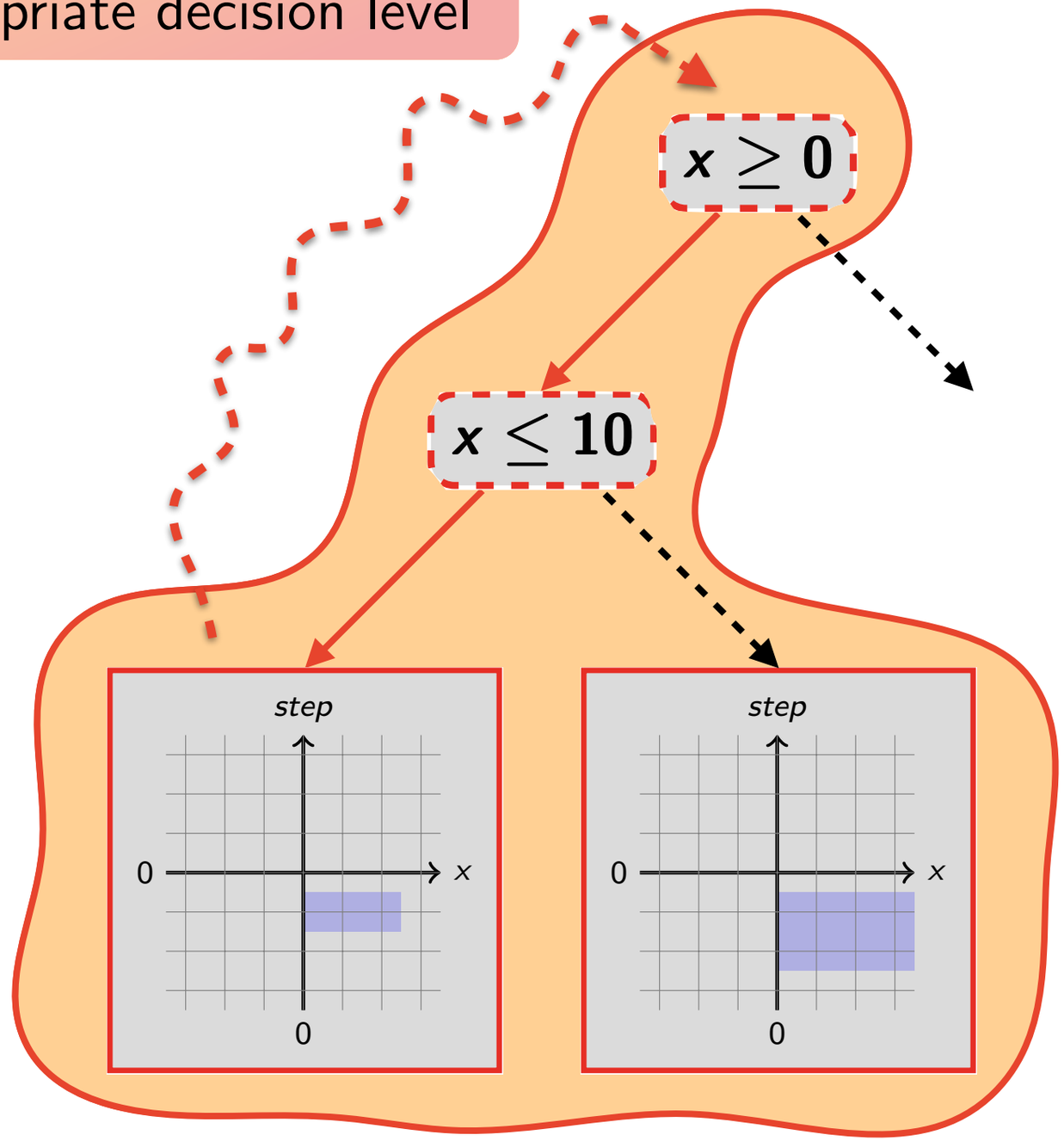


the widening generalizes the **value** and the **domain** of a ranking function

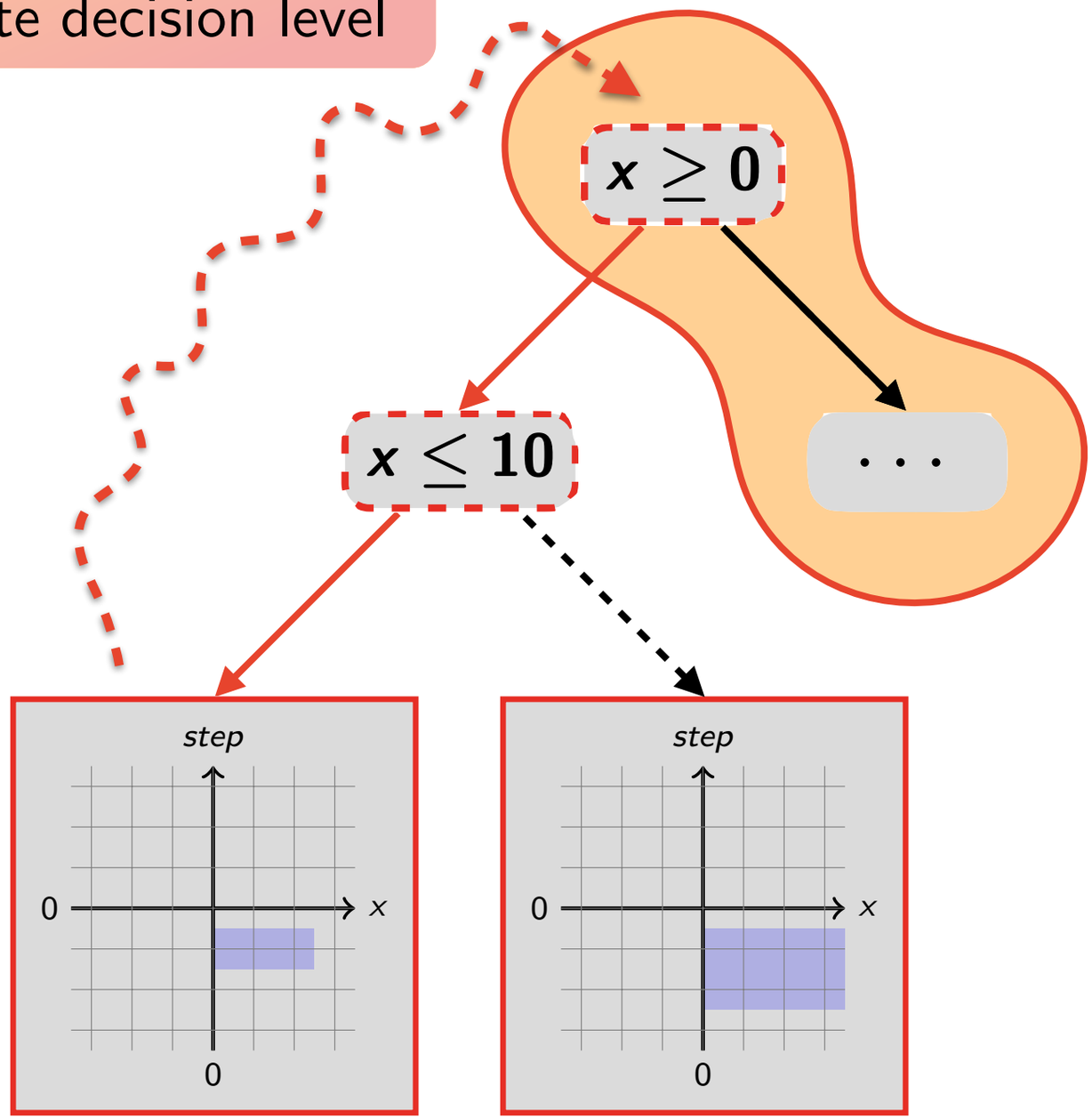

$x \geq 0$
$x \leq 10$
step
0
x
0


$x \geq 0$
$x \leq 10$
step
0
x
0
step
0
x
0

the **domain** of the ranking function guides the **backtracking** at the appropriate decision level

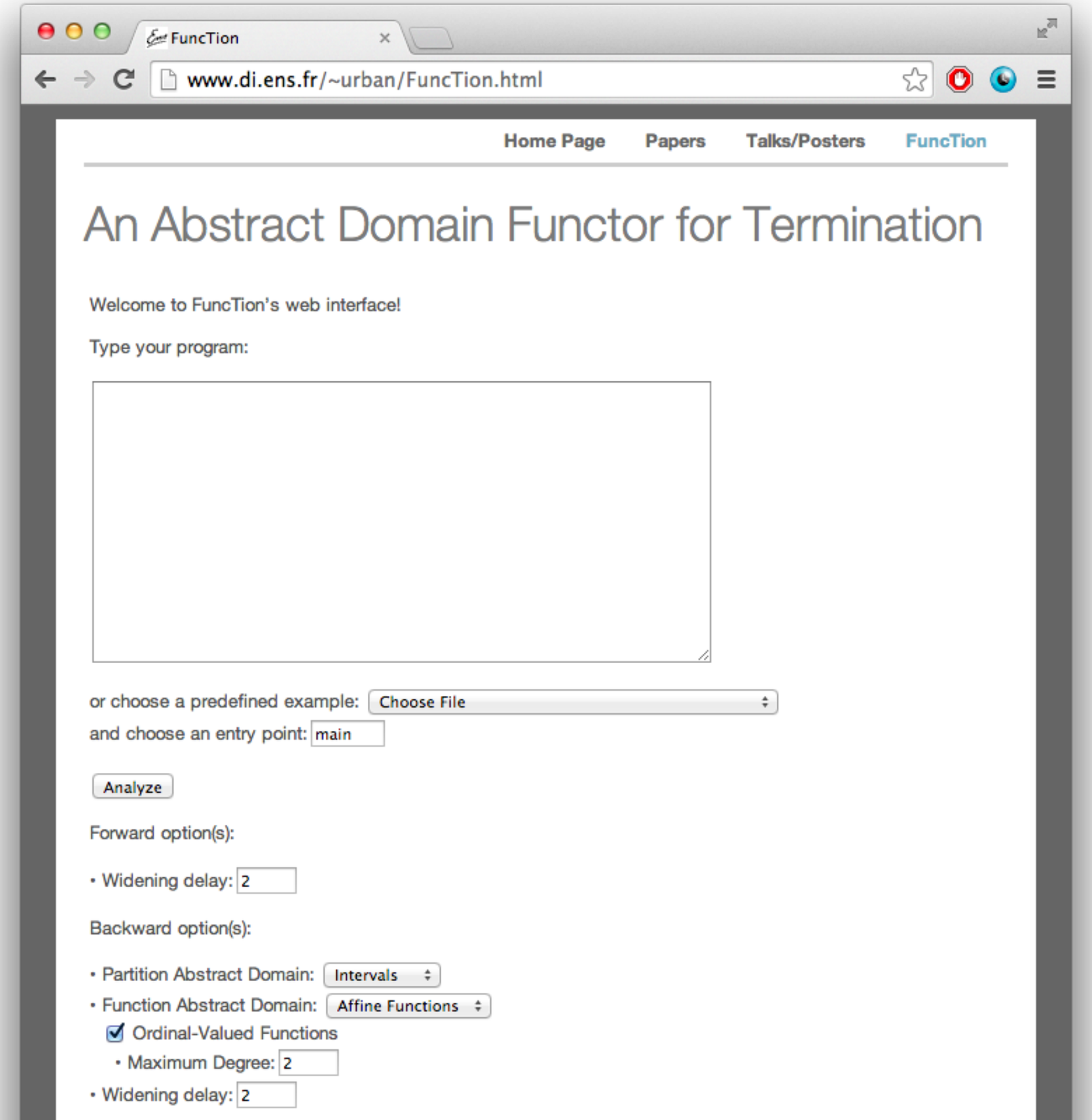


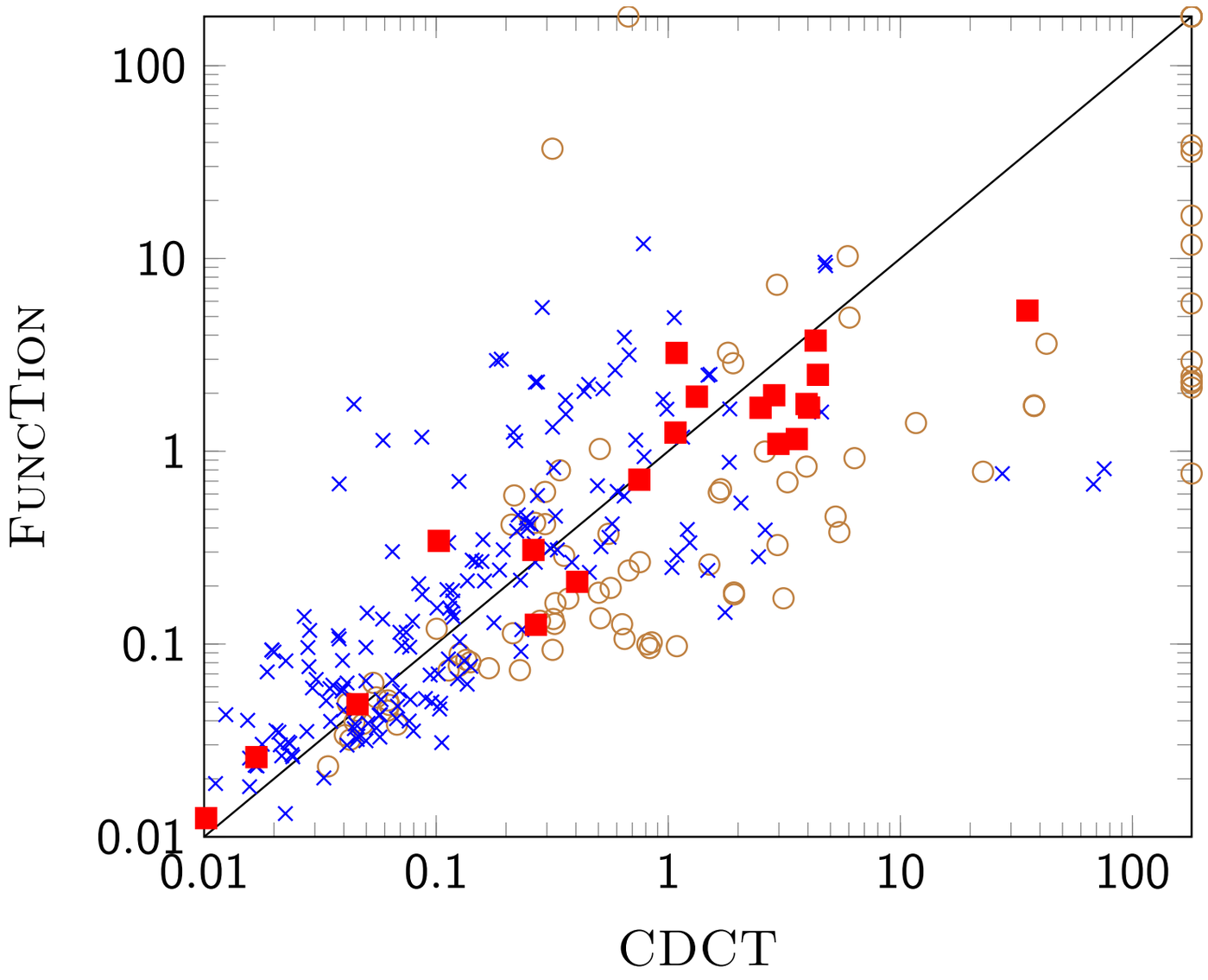
the **domain** of the ranking function guides the **backtracking** at the appropriate decision level



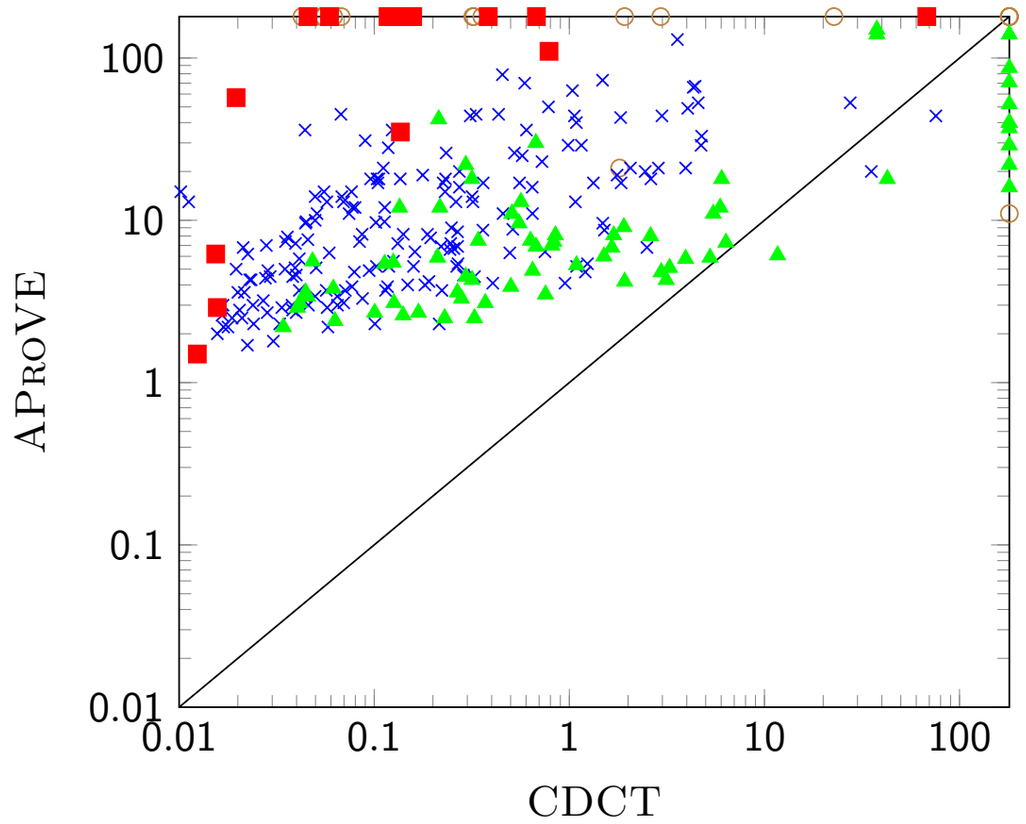
More in the paper...

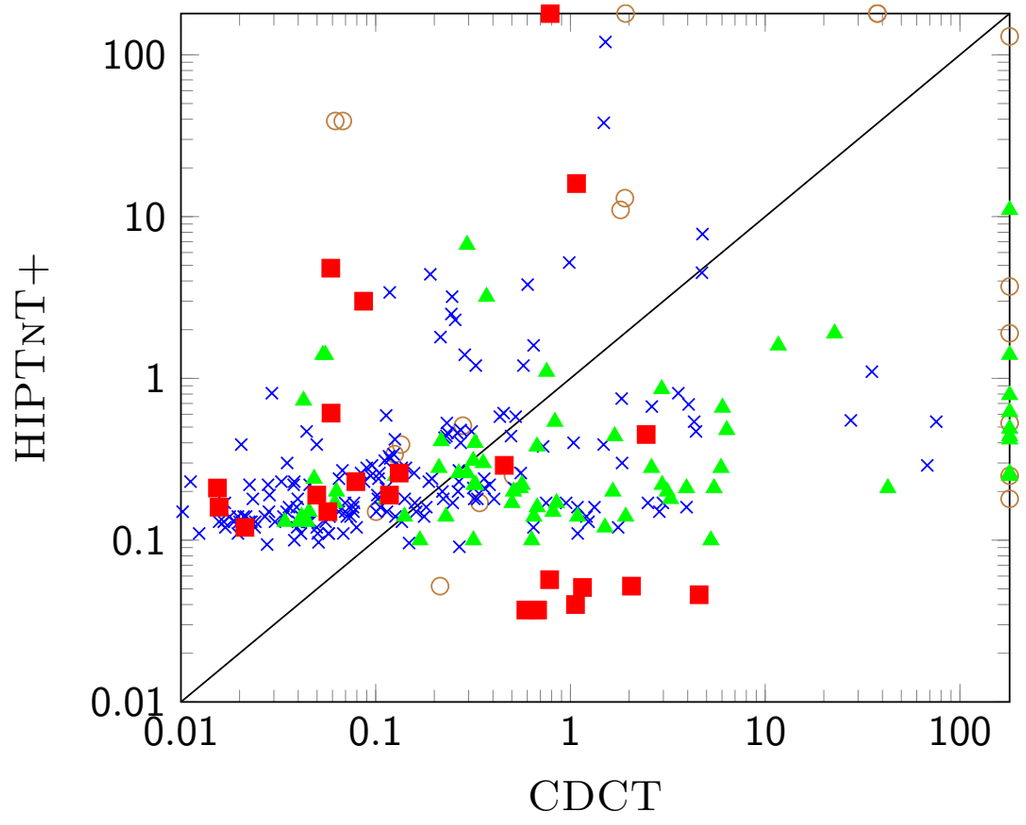
- non-termination as satisfiability via **Büchi's theorem**
- use of a **trail** to represent progress of an abstract interpreter
- **clausal representation** of conditional termination results
- a **generalized unit rule** to integrate clauses in propagation

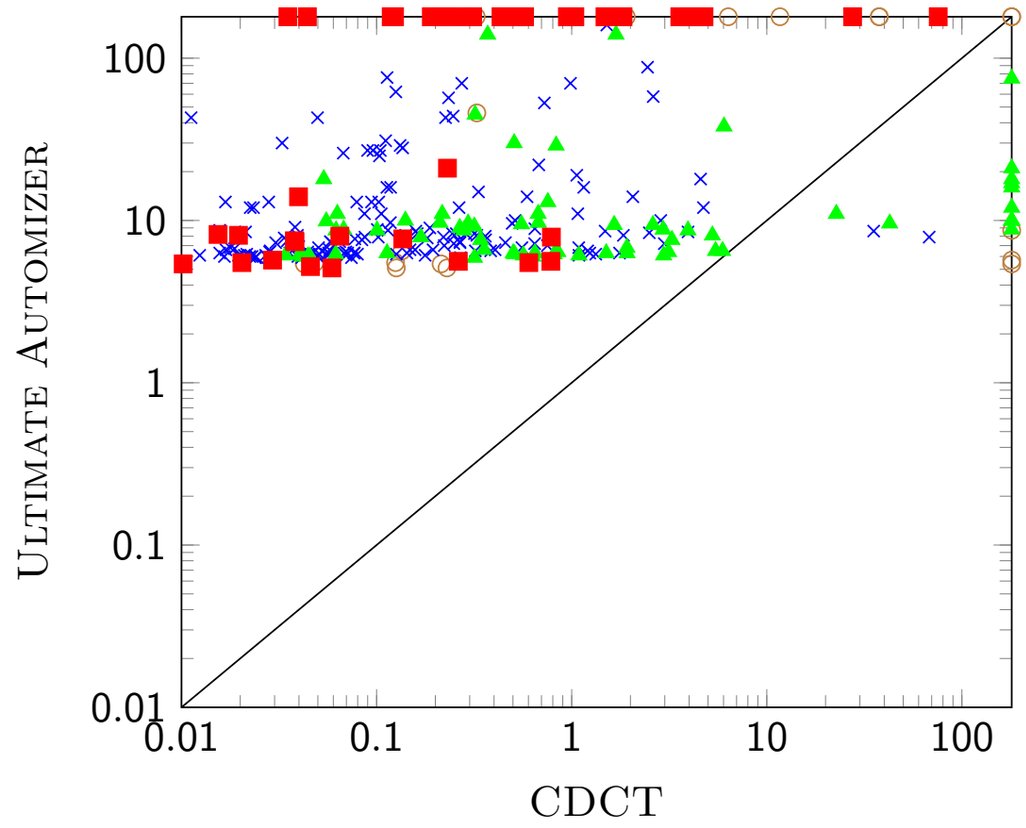

FuncTion
www.di.ens.fr/~urban/FuncTion.html
Home Page
Papers
Talks/Posters
FuncTion
An Abstract Domain Functor for Termination
Welcome to FuncTion's web interface!
Type your program:
or choose a predefined example:
Choose File
and choose an entry point:
main
Analyze
Forward option(s):
• Widening delay:
2
Backward option(s):
• Partition Abstract Domain:
Intervals
• Function Abstract Domain:
Affine Functions
Ordinal-Valued Functions
• Maximum Degree:
2
• Widening delay:
2



- 288 terminating programs
- 8.7% CDCT is more precise (■)
- 65.7% CDCT is faster (×)


AProVE
CDCT
0.01
0.1
1
10
100


HIPTnT+
CDCT
0.01
0.1
1
10
100


Ultimate Automizer
CDCT
0.01
0.1
1
10
100

Conclusion

- **conflict-driven learning** for conditional termination
  - search: potential non-termination
  - refutation: conditional termination analysis
- application to **abstract interpretation-based** analysis
  - potential for application to other termination analyses

Future Work

- better potential **non-termination** analysis
- better **decision heuristics**
- better **widening**
- conflict-driven learning for **liveness properties**