Completeness Thresholds for Memory Safety of Array Traversing Programs

Tobias Reinhard, Justus Fasse, Bart Jacobs
KU Leuven

Workshop on the State Of the Art in Program Analysis
June 17, 2023

What This Work Is About

- Connection between bounded & unbounded proofs
- Ideas to increase trust in bounded model checking

What This Work Is About

- Connection between bounded & unbounded proofs
- Ideas to increase trust in bounded model checking
- When is a bounded “proof” a proof?

Model Checking: Easy Off-by-1 Error

- WHILE language with pointer arithmetic
- Targeted property: Memory safety
- Memory assumption $\texttt{array}(a, s)$:
  $a[0] \ \ldots \ a[s{-}1]$ allocated

```
for i in [0 : s-1] do
    !a[i+1]
```

Model Checking: Easy Off-by-1 Error

- WHILE language with pointer arithmetic
- Targeted property: Memory safety
- Memory assumption $\texttt{array}(a, s)$:
  $a[0] \ \ldots \ a[s-1]$ allocated

```
for i in [0 : s-1] do
    !a[i+1]
```

Which bounds should we choose for s?

- $s = 0$: No error
- $s = 1$: Error

Model Checking: “Harder” Off-by-N Error

Memory assumption:
$\texttt{array}(a, s)$

```
for i in [0 : s-2] do
    !a[i+2]
```

Which bounds should we choose for s?

Model Checking: “Harder” Off-by-N Error

Memory assumption: $\texttt{array}(a, s)$	for i in [0 : s-2] do !a[i+2]

Which bounds should we choose for s?

- $s = 0$: No error
- $s = 1$: No error
- $s = 2$: Error

Model Checking: No Off-by-N Error

Memory assumption:
$\texttt{array}(a, s)$

```
for i in [0 : s-1] do
    !a[i]
```

Which s can convince us?

Model Checking: No Off-by-N Error

Memory assumption: $\texttt{array}(a, s)$	for i in [0 : s-1] do !a[i]

Which s can convince us?

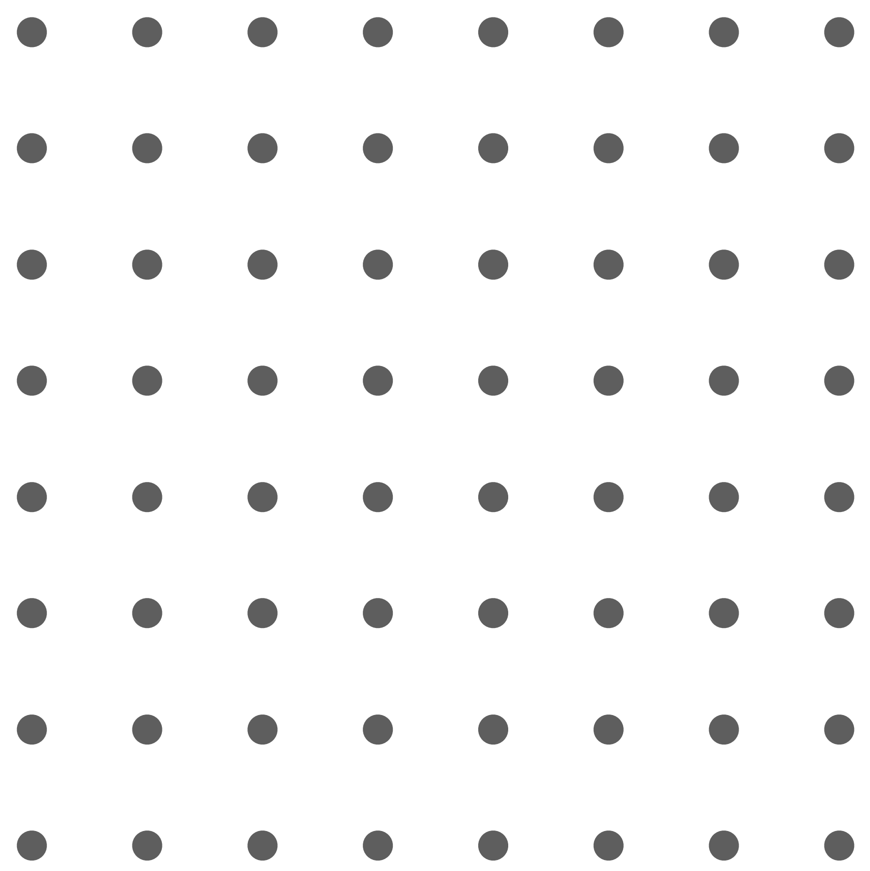
- $s = 0$: No error
- $s = 1$: No error
- $s = 2$: No error
- $s = 3$: No error

$\vdots$

$\Rightarrow$ Which size bound is large enough?

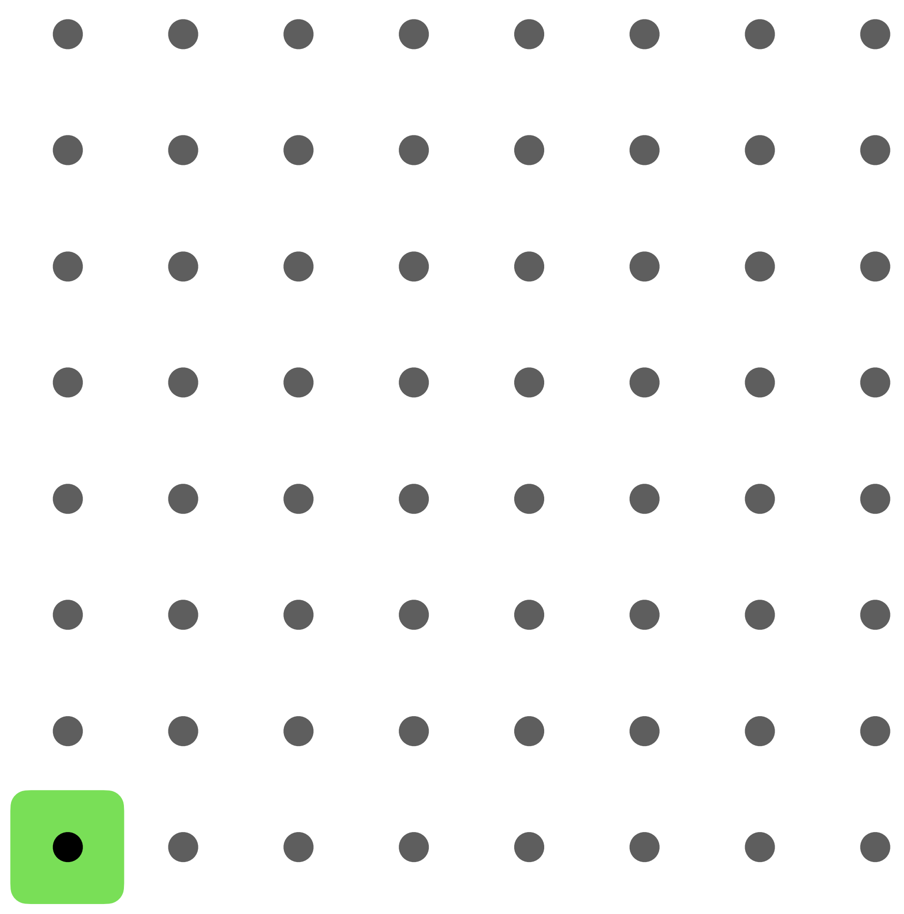
Model Checking Finite Systems

- Finite state transition system T
- Prove property Gp
  G ≈ globally ≈ p holds in every state
- Approach:
  Prove Gp for all paths up to length k
  $T \vDash_k \mathrm{G}p$



Model Checking Finite Systems

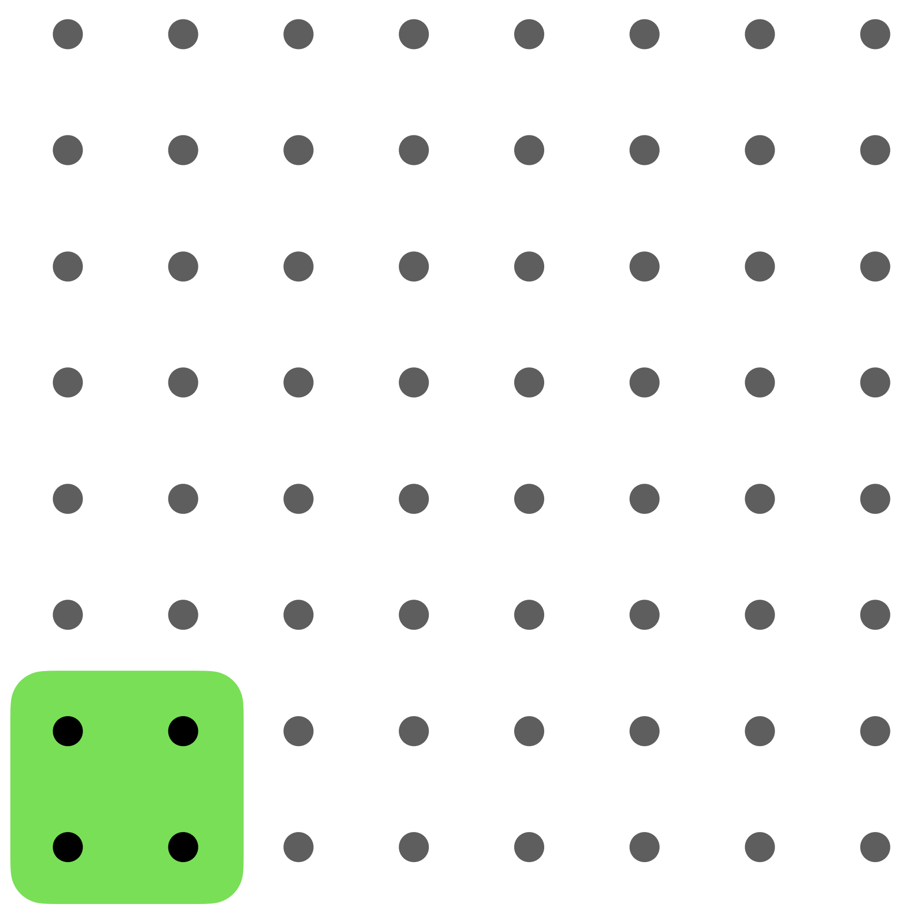
- Finite state transition system T
- Prove property Gp
  G ≈ globally ≈ p holds in every state
- Approach:
  Prove Gp for all paths up to length k
  $T \vDash_k \mathrm{G}p$



$T \vDash_0 \mathrm{G}p$

Model Checking Finite Systems

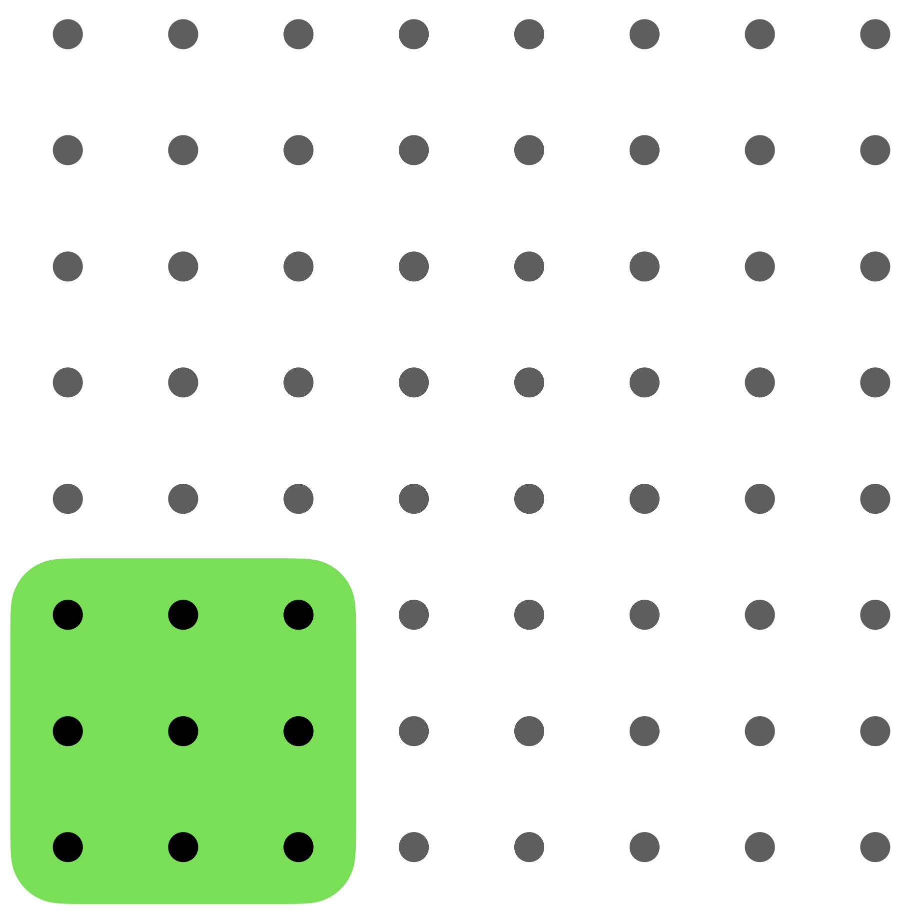
- Finite state transition system T
- Prove property Gp
  G ≈ globally ≈ p holds in every state
- Approach:
  Prove Gp for all paths up to length k
  $T \vDash_k \mathrm{G}p$



$T \vDash_1 \mathrm{G}p$

Model Checking Finite Systems

- Finite state transition system T
- Prove property Gp
  G ≈ globally ≈ p holds in every state
- Approach:
  Prove Gp for all paths up to length k
  $T \vDash_k \mathsf{G}p$



$T \vDash_2 \mathsf{G}p$

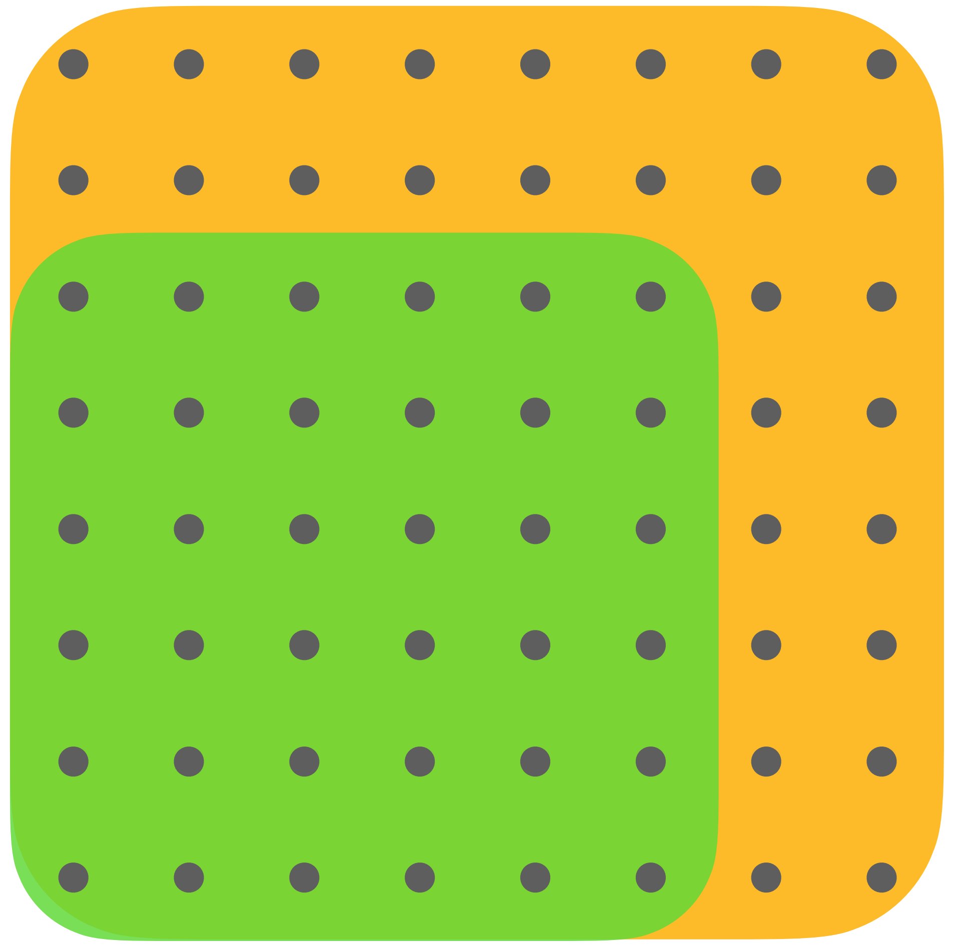
When should we stop?

Completeness Thresholds for Finite Systems

- k is completeness thresholds (CT) iff

  $$\boxed{T \models_k \phi} \Rightarrow \boxed{T \models \phi}$$

- For specific ϕ:
  Can over-approximate CT via of key props of T



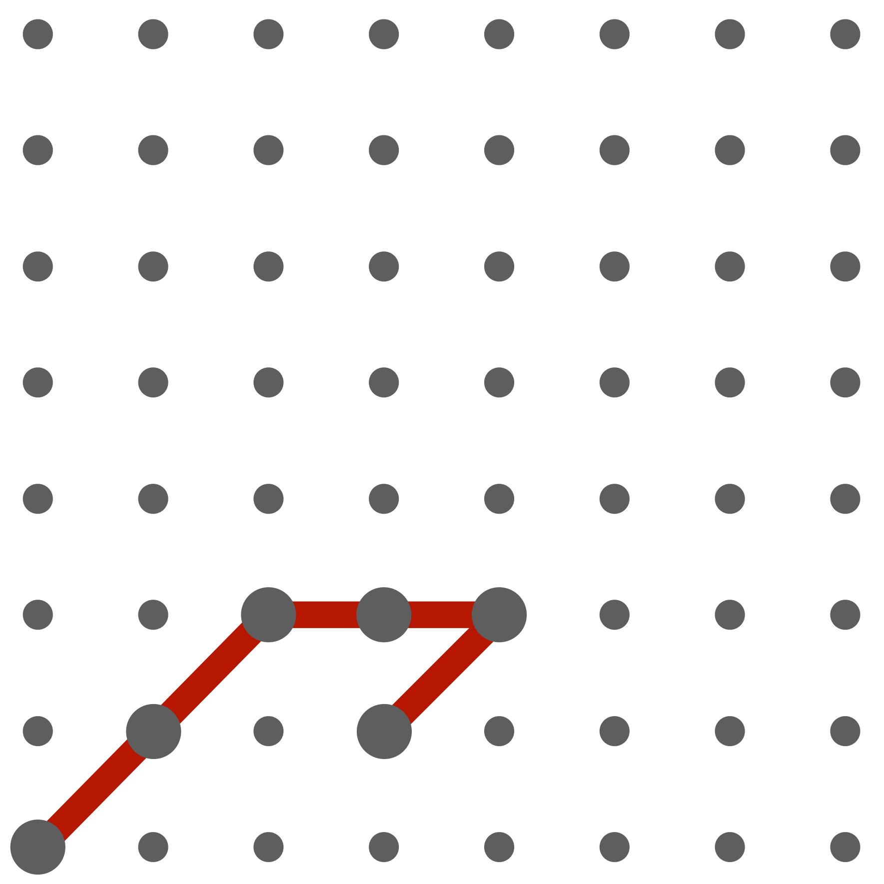
Completeness Thresholds for Finite Systems

- k is completeness thresholds (CT) iff

$$T \vDash_k \phi \;\; \Rightarrow \;\; T \vDash \phi$$

- For specific ϕ:
  Can over-approximate CT via of key props of T

- For $\phi = \mathrm{G}p$ we know
  $\mathrm{CT}(T, \mathrm{G}p) = \text{recurrence\_diameter}(T)$
  (length of longest loop-free path)



recurrence_diameter$(T) = 5$

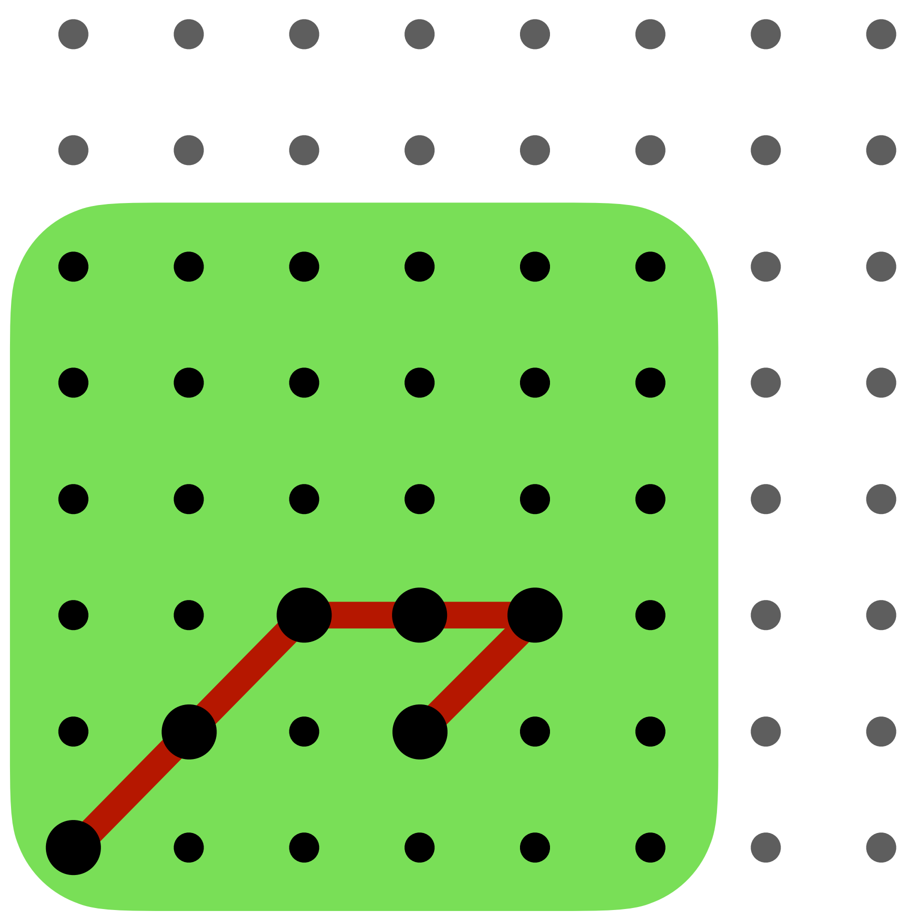
Completeness Thresholds for Finite Systems

- k is completeness thresholds (CT) iff

$$T \vDash_k \phi \;\; \Rightarrow \;\; T \vDash \phi$$

- For specific ϕ:
  Can over-approximate CT via of key props of T

- For $\phi = \mathsf{G}p$ we know
  $\mathrm{CT}(T, \mathsf{G}p) = \text{recurrence\_diameter}(T)$
  (length of longest loop-free path)



recurrence_diameter$(T) = 5$

$T \vDash_5 \mathsf{G}p$

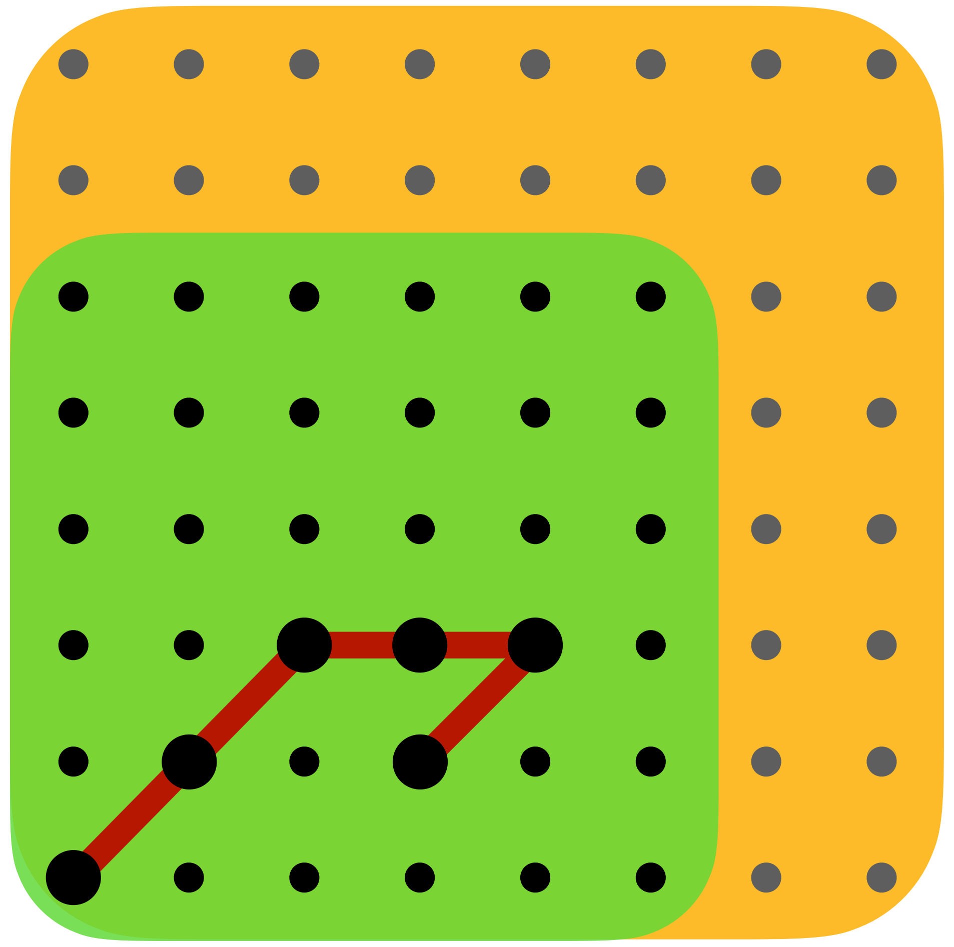
Completeness Thresholds for Finite Systems

- k is completeness thresholds (CT) iff

$$T \models_k \phi \;\;\Rightarrow\;\; T \models \phi$$

- For specific ϕ:
  Can over-approximate CT via of key props of T

- For $\phi = \mathsf{G}p$ we know
  $\mathrm{CT}(T, \mathsf{G}p) = \text{recurrence\_diameter}(T)$
  (length of longest loop-free path)



recurrence_diameter$(T) = 5$

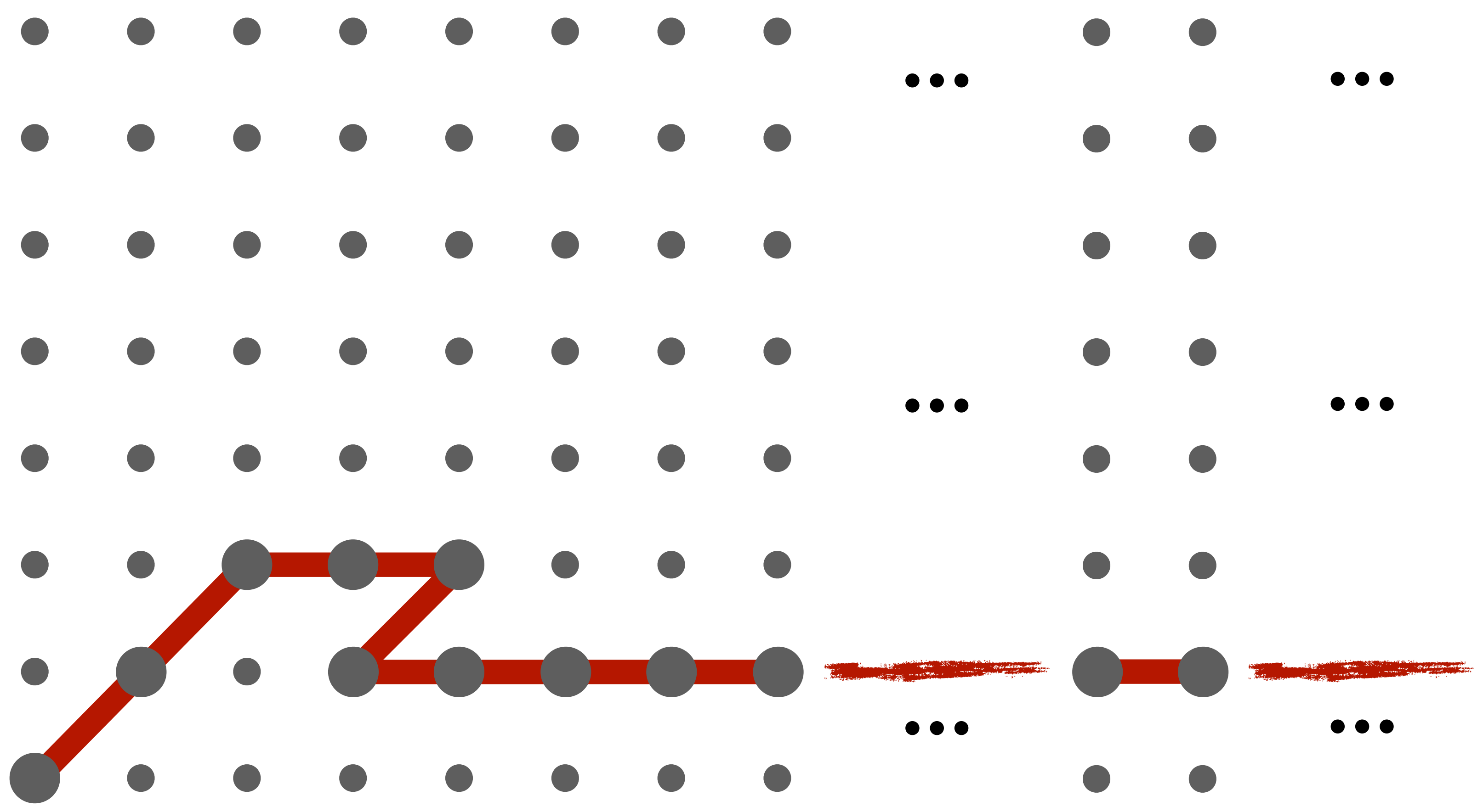
$\downarrow$

$T \models_5 \mathsf{G}p \longrightarrow T \models \mathsf{G}p$

CTs for Infinite Systems?

Problem

Key properties used to describe CTs may be ∞



recurrence_diameter$(T) = \infty$

CTs for Infinite Systems?

Problem

Key properties used to describe CTs may be ∞

Our Approach

Analyse program's *verification conditions*
instead of transition system

Verification Conditions

- Logical formula vc is VC for any spec $Spec(c)$ iff

$$\models vc \quad \Rightarrow \quad \models Spec(c)$$

- Can verify VC instead of program

- In general: VCs are over-approximations, i.e.,

  possible that $\not\models vc$ but $\models Spec(c)$

Completeness Thresholds

- Program variable x with domain X
- Specification $\forall x \in X \,.\, Spec(c)$

Completeness Thresholds

- Program variable x with domain X
- Specification $\forall x \in X \,.\, Spec(c)$
- Subdomain $Q \subseteq X$ is a CT for x in $\forall x \in X \,.\, Spec(c)$ iff

$$\models \forall x \in Q \,.\; Spec(c) \quad \Rightarrow \quad \models \forall x \in X \,.\; Spec(c)$$

- For us: CT are subdomains, not depths

How to Prove CTs

- Generate VC: $Spec(c) \quad \rightsquigarrow \quad \forall x \in X \,.\, vc(x)$

How to Prove CTs

- Generate VC: $Spec(c) \rightsquigarrow \forall x \in X \,.\, vc(x)$

- Identify subdomain $Y \subseteq X$ where choice $x \in Y$ does not influence validity of $vc(x)$

$$\Big(\models vc(x) \quad \Leftrightarrow \quad \models vc' \quad \text{with } \; x \notin \text{free}(vc') \Big)$$

$$\Longrightarrow \quad \text{Found CT:} \quad (X \backslash Y) \cup \{y\} \quad \text{(for any choice of } y \in Y\text{)}$$

How Does the Array Size Affect Memory Safety?

Memory assumption:

$\texttt{array}(a, s)$

```
for i in [L : s-R] do
  !a[i+Z]
```

How Does the Array Size Affect Memory Safety?

Memory assumption:
$\texttt{array}(a, s)$

```
for i in [L : s-R] do
    !a[i+Z]
```

Generate VC

(fully automated)

VC $vc_0 := \forall s \,.\, \texttt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\} \,.\, a[i+Z]$ alloc

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 := \forall s \,.\, \texttt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\} \,.\, a[i{+}Z] \text{ alloc}}$$

Range L, ..., s-R empty?

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 \;:=\; \forall s\,.\, \mathtt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\}\,.\, a[i{+}Z] \text{ alloc}}$$

Range L, ..., s-R empty?

Yes

$s^- < L + R$

Simplify VC!

$$vc_0 \;\equiv\; \forall s^-.\; \ldots \rightarrow \forall i \in \varnothing\,.\; \ldots$$
$$\equiv\; \text{True}$$

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 \ := \ \forall s\,.\ \texttt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\}\,.\ a[i{+}Z] \text{ alloc}}$$

Range L, ..., s-R empty?

Yes

$s^- < L + R$

No need to check

Simplify VC!

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 \ := \ \forall s \,.\, \mathtt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\} \,.\, a[i+Z] \text{ alloc}}$$

Range L, ..., s-R empty?

Simplify VC!

Yes

$s^- < L + R$

No need to check

No

$s^+ \geq L + R$

$$vc_0 \ \equiv \ \forall i \,.\, (L \leq i < s^+ - R) \rightarrow (0 \leq i+Z < s^+)$$

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 \ := \ \forall s\,.\ \texttt{array}(a, s) \to \forall i \in \{L, \ldots, s - R\}\,.\ a[i{+}Z] \text{ alloc}}$$

Range L, ..., s-R empty?

Simplify VC!

Yes

$s^- < L + R$

No need to check

No

$s^+ \geq L + R$

$$vc_0 \ \equiv \ \forall i\,.(L \leq i < \cancel{s^+} - R) \to (0 \leq i{+}Z < \cancel{s^+})$$

$$\equiv \ \forall i\,.(L \leq i \to 0 \leq i{+}Z)$$

$$\wedge \ (i \leq -R) \to i{+}Z < 0)$$

$\Rightarrow$ Validity does not depend on size

How Does the Array Size Affect Memory Safety?

$$\boxed{\text{VC } vc_0 \ := \ \forall s\,.\ \texttt{array}(a, s) \to \forall i \in \{L, \ldots, s - R\}\,.\ a[i{+}Z] \text{ alloc}}$$

Range L, ..., s-R empty?

Yes

$s^- < L + R$

No need to check

No

$s^+ \geq L + R$

Can check for any

$s^+ \geq L + R$

How Does the Array Size Affect Memory Safety?

$\boxed{\text{VC } vc_0 \;:=\; \forall s\,.\,\texttt{array}(a, s) \rightarrow \forall i \in \{L, \ldots, s - R\}\,.\,a[i{+}Z] \text{ alloc}}$

Range L, ..., s-R empty?

Yes

$s^- < L + R$

No need to check

No

$s^+ \geq L + R$

Can check for any

$s^+ \geq L + R$

Found CT: $\{s^+\}$

Workflow: How to Find CTs

Mem spec
+ program

Workflow: How to Find CTs

Mem spec
+ program

$\xrightarrow{\text{Generate VC}}$ VC $\forall x \in X \,.\, vc(x)$

Workflow: How to Find CTs

Mem spec
+ program

Generate VC

VC $\forall x \in X\,.\; vc(x)$

Select subdomain

$Y \subseteq X$

Restricted VC
$vc(x)$ with $x \in Y$

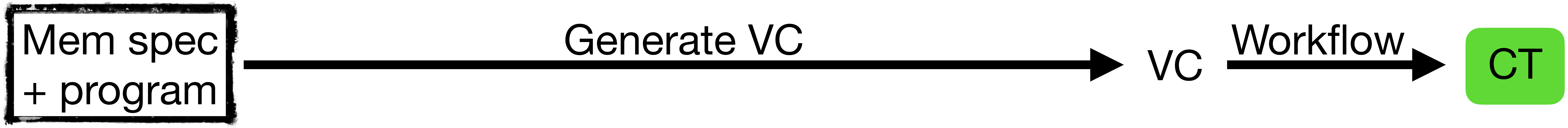
Try to eliminate x

$vc'(x)$ with $x \in Y$

Workflow: How to Find CTs

Scalability

Program Slicing

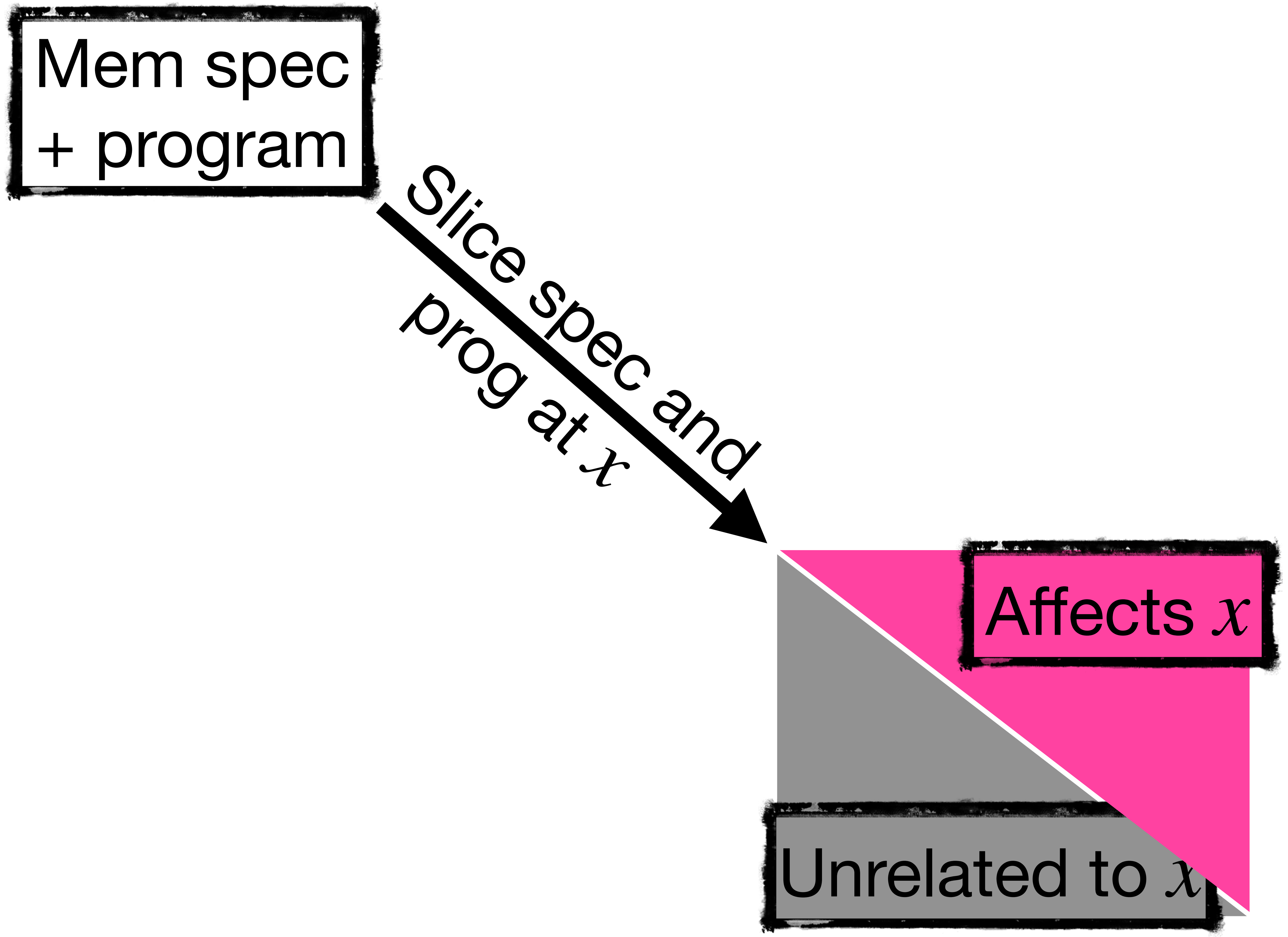


Scalability

Program Slicing

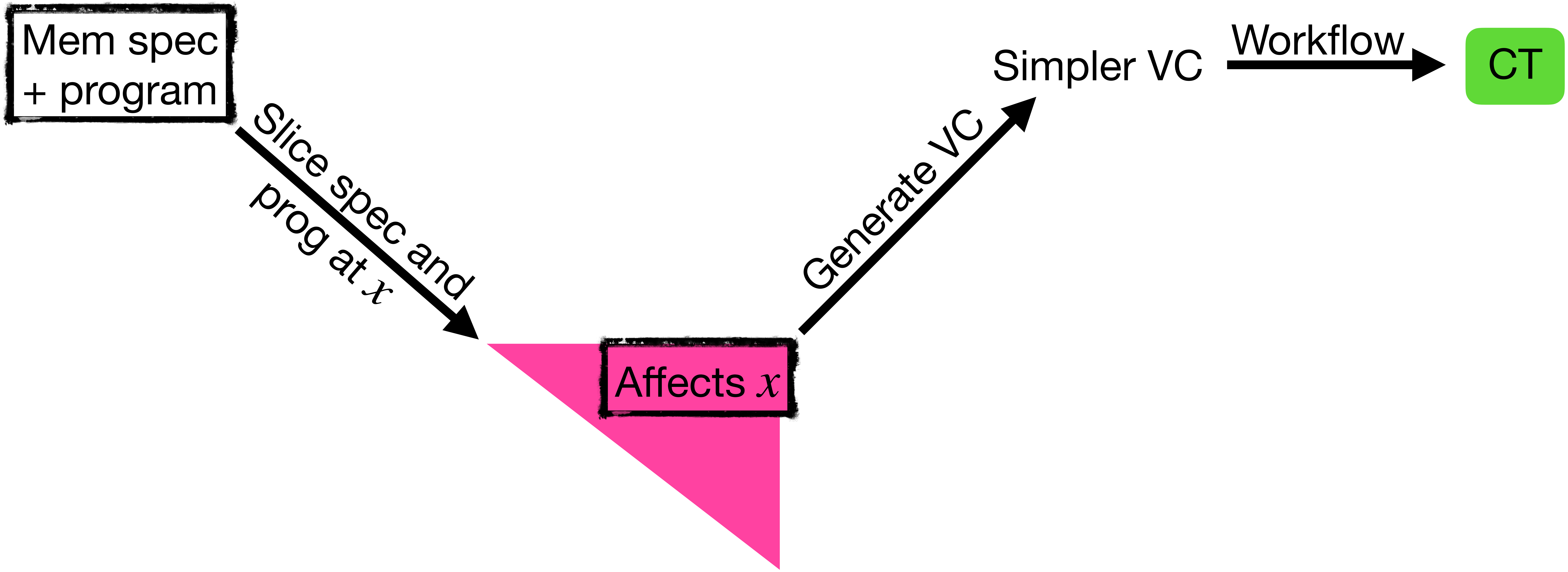
Scalability

Program Slicing



Scalability

Program Slicing



Scalability

CT Combinators

Sequencing

$c_1; c_2$ CTs Q_1, Q_2

$$\downarrow$$

$$Q = Q_1 \cup Q_2$$

Scalability

CT Combinators

Sequencing

$c_1; c_2$

CTs Q_1, Q_2

$Q = Q_1 \cup Q_2$

Branching

if e then c_1 else c_2

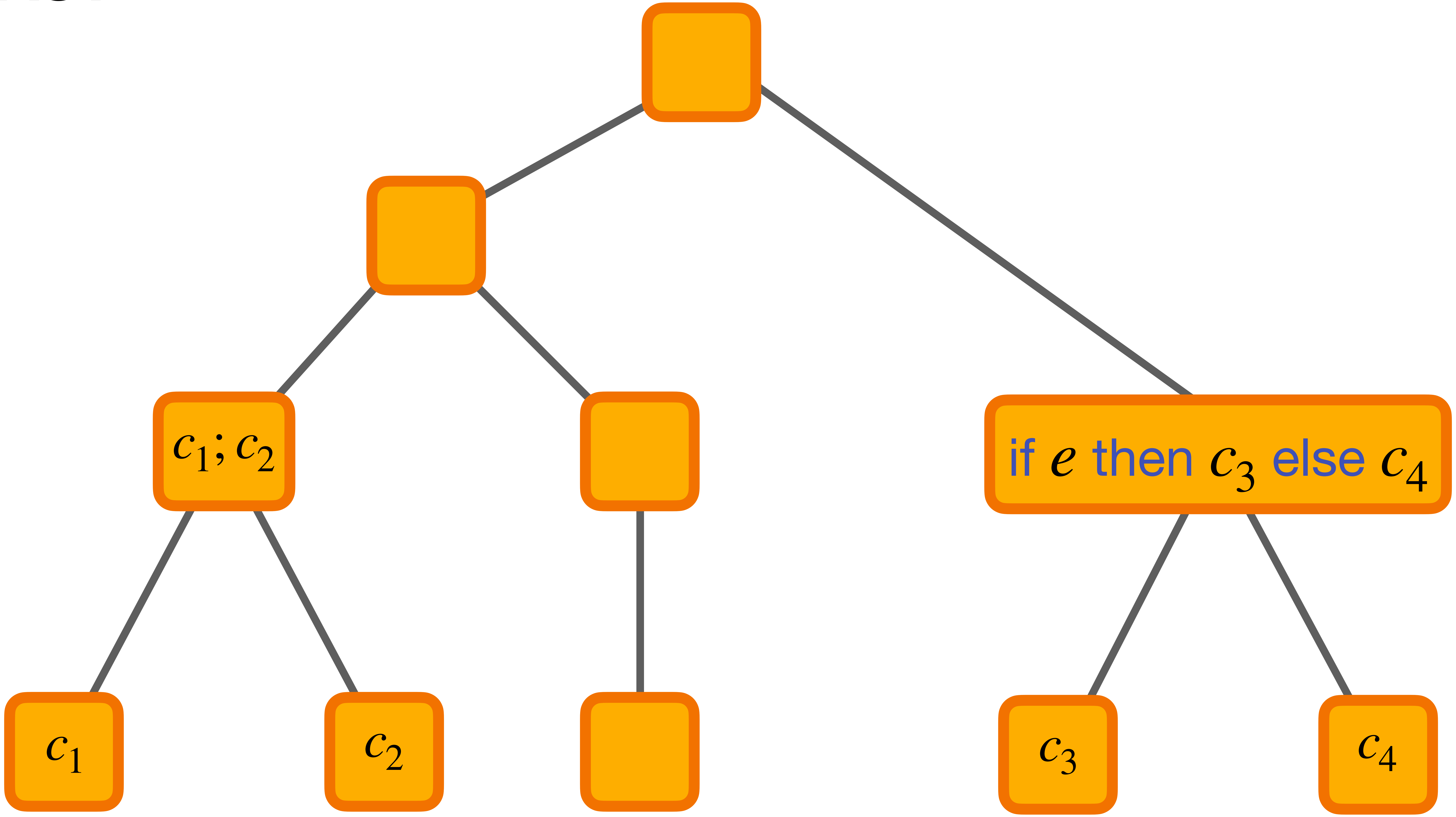
CTs as contraints

$Q_i \sim K_i$

$Q \ \sim \ (e \wedge K_1) \vee (\neg e \wedge K_2)$

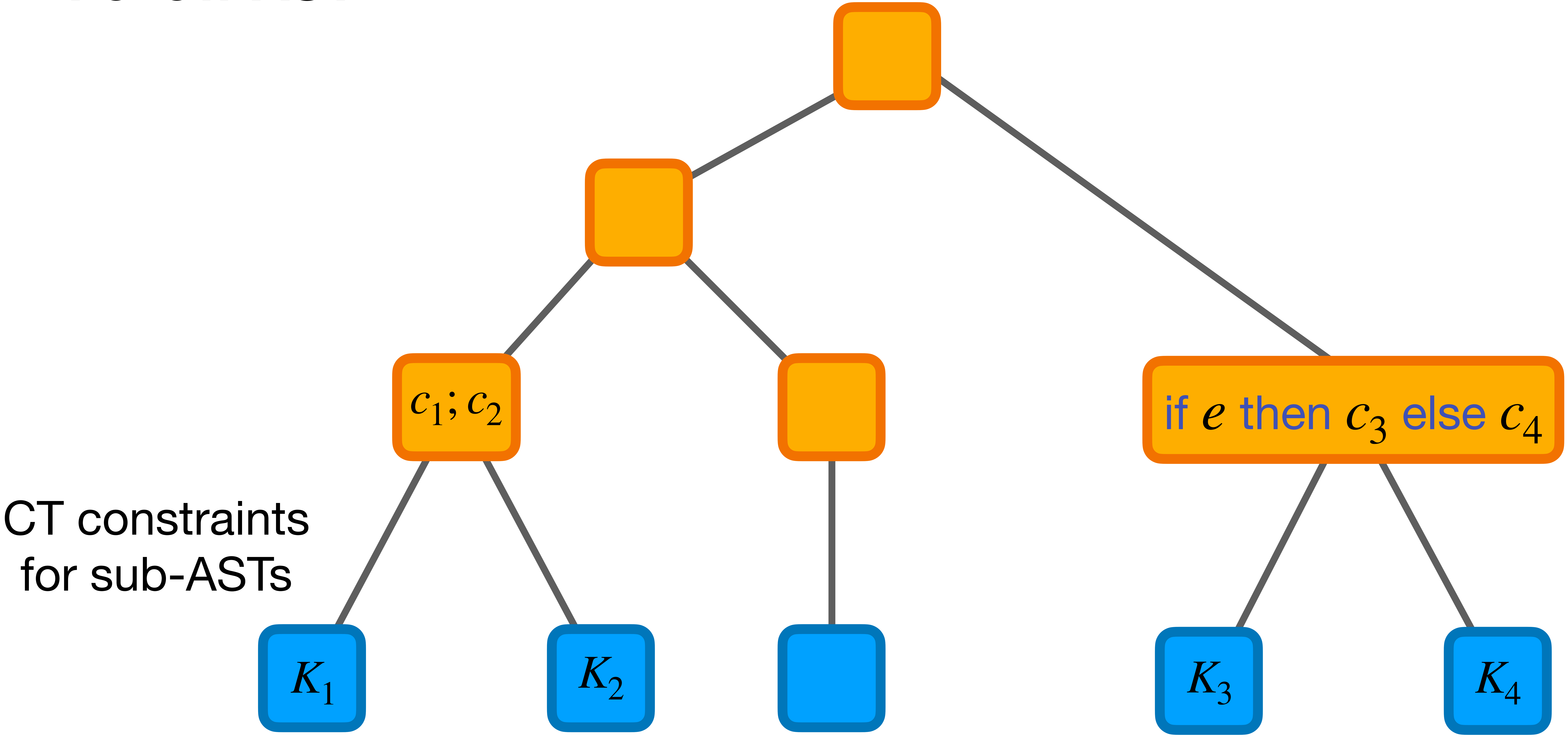
Scalability

Follow AST



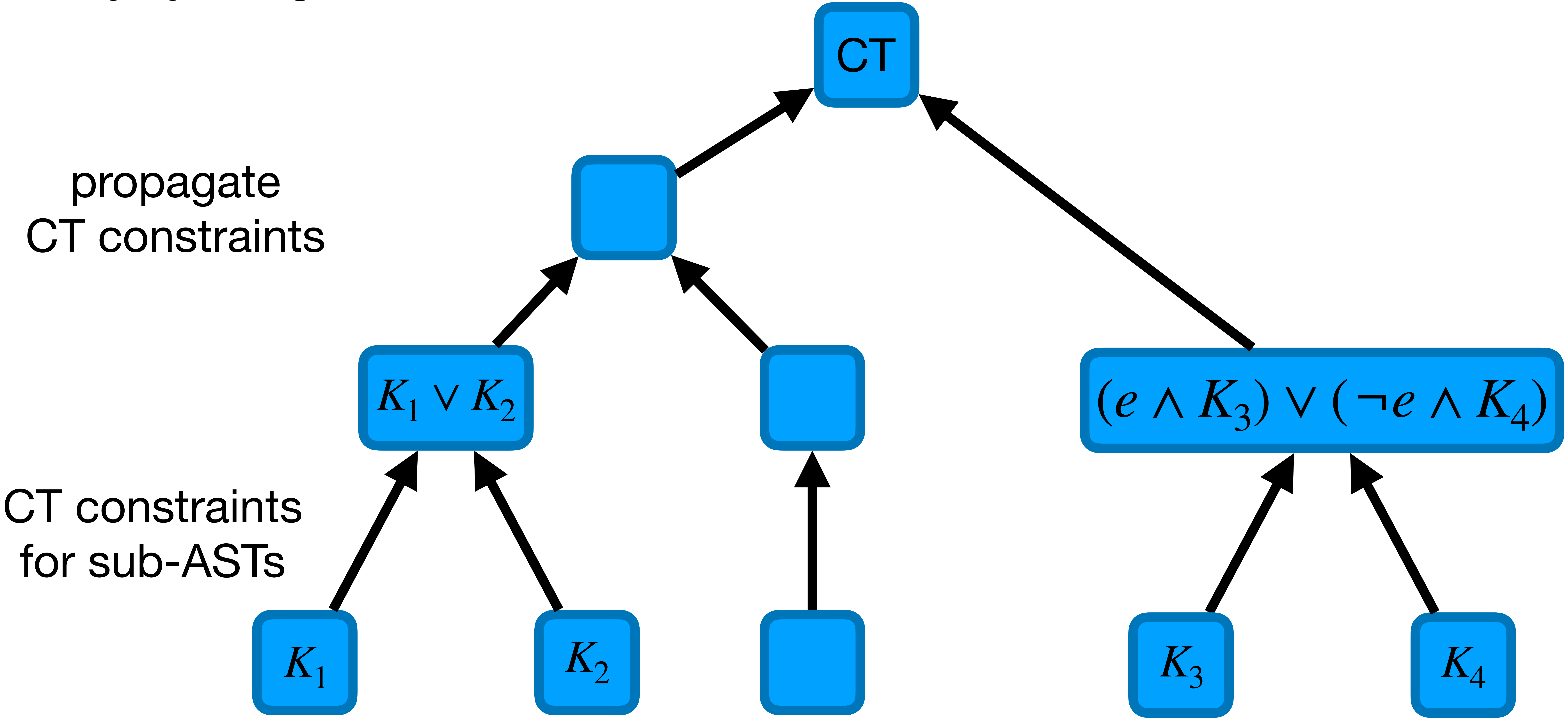
Scalability

Follow AST



Scalability

Follow AST



Outlook: Challenges

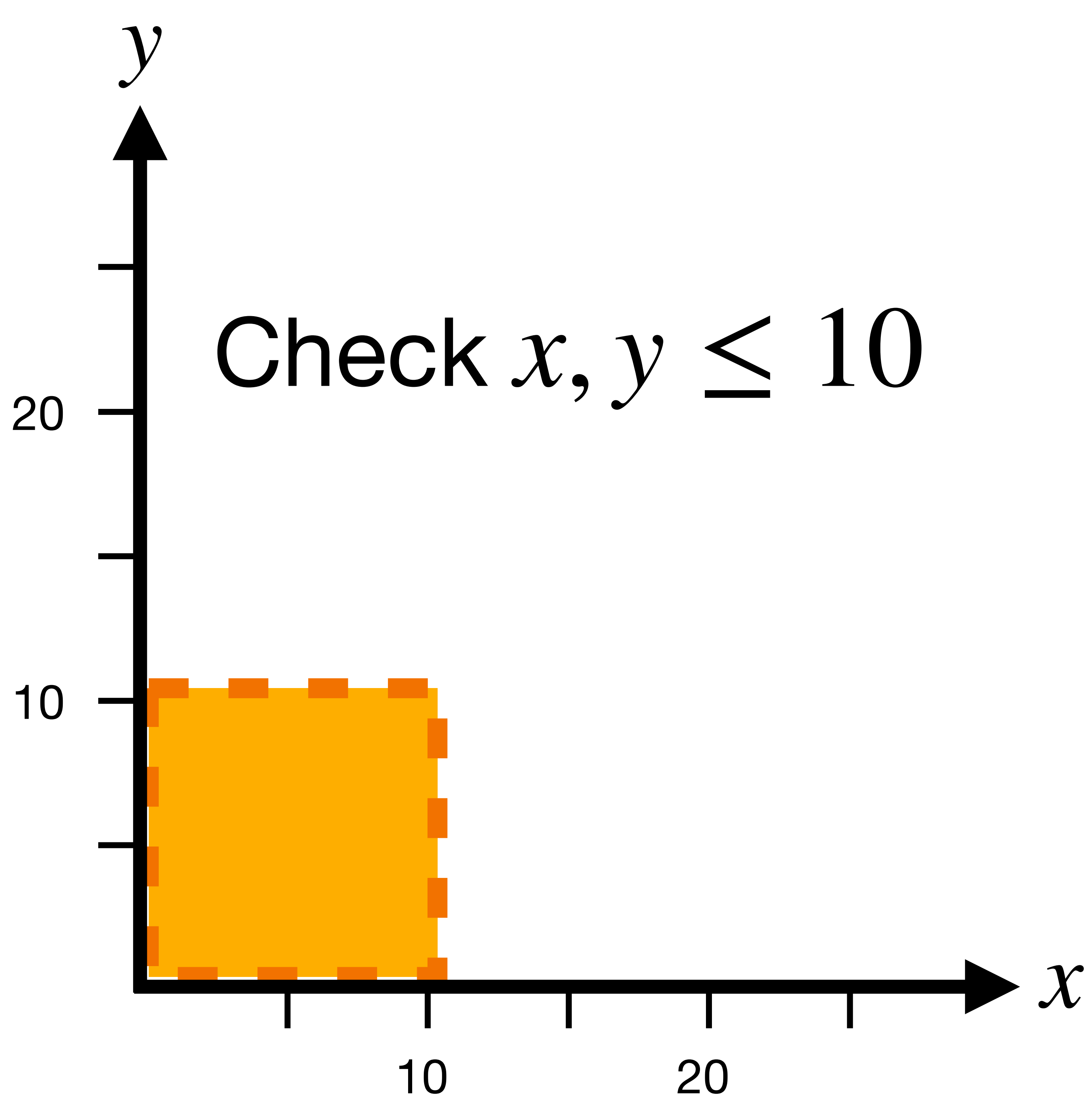
- Automation, e.g., automatic VC rewriting
- Demo scalability: Complex programs & data (e.g. lists, trees)

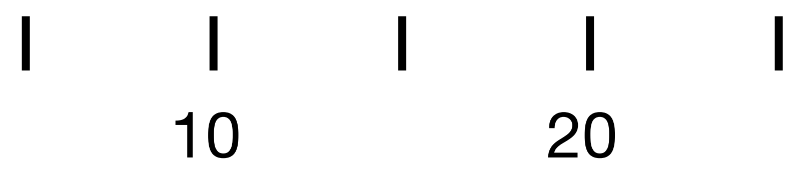
Outlook: Increase Trust in BMC

- Turn bounded into unbounded proof

Outlook: Increase Trust in BMC

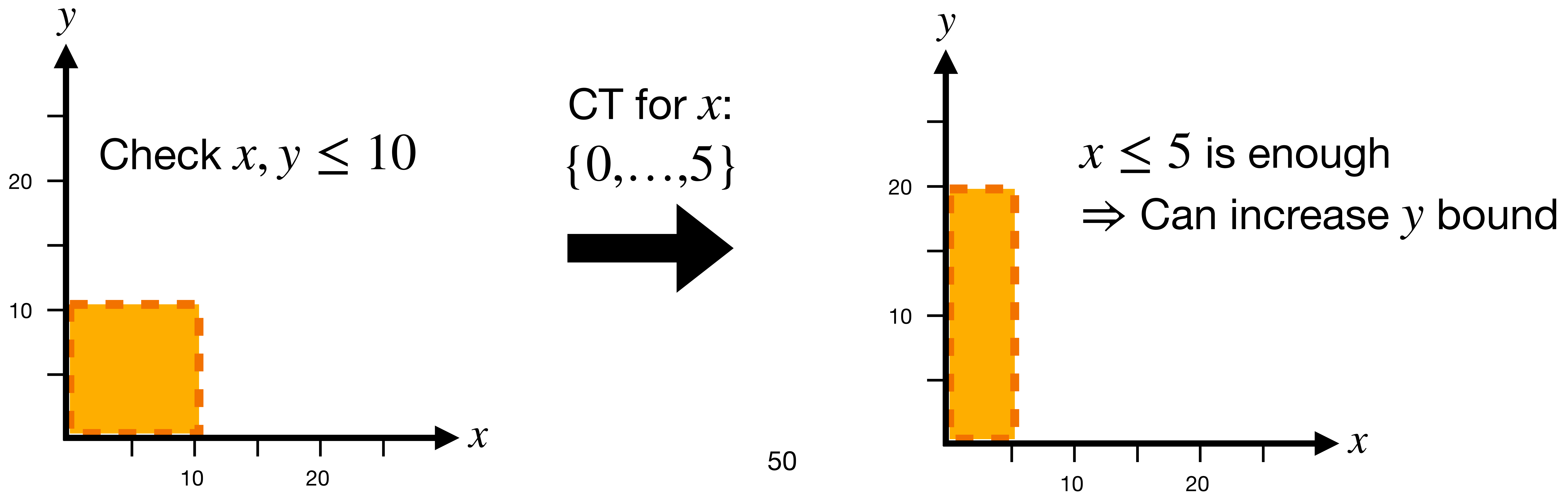
- Turn bounded into unbounded proof
- Shift resources to critical bounds





Outlook: Increase Trust in BMC

- Turn bounded into unbounded proof
- Shift resources to critical bounds



Conclusion

- First generalisation of CTs to infinite state systems
- Connection between bounded & unbounded proofs in program verification
- Foundational research but potential for integration into BMC

Backup Slides

Precise VCs

- VC vc is *precise* for x in $Spec$ iff

  $$\forall v\,.\ \left(\ \models Spec[x \mapsto v] \ \Rightarrow\ \models vc[x \mapsto v]\ \right)$$

  Intuition: vc does not over-approximate wrt. x

- Q is CT $vc\ \wedge\ vc$ is precise $\ \Rightarrow\ Q$ is CT $Spec$

Precise VCs

$$
\begin{tikzcd}
\forall x\,.\ Spec \arrow[r, "\text{Unbounded proof}"] \arrow[d] & \models \forall x\,.\ Spec \\
\forall x\,.\ vc \arrow[d] & \\
Q \text{ is CT for } vc \arrow[d, "vc \text{ precise for } x"] & \\
Q \text{ is CT for } Spec \arrow[r, "\text{Bounded proof}"] & \models \forall x \in Q\,.\ Spec \arrow[uuu, "\text{CT } Q"']
\end{tikzcd}
$$