Forward Analysis for Recurrent Sets

Alexey Bakhirkin[1] Josh Berdine[2] Nir Piterman[1]

[1]University of Leicester, Department of Computer Science
[2]Microsoft Research

Why (non-)termination

A non-termination bug in the below code made many Zune devices freeze on 31 Dec 2008.

```
days ← // days since 1 Jan 1980
year ← 1980
while days > 365:
    if leap(year):
        if days > 366:
            days ← days − 366
            year ← year + 1
    else:
        days ← days − 365
        year ← year + 1
```

The official response was, "Wait until battery dies".

Why (non-)termination

- Many programs are supposed to terminate.
- People are bad at finding (non-)termination bugs.
- There are other analyses (for example, CTL model checking) that rely on (non-)termination results.

Termination and Nontermination

A family of undecidable problems.

Find a set of states, such that from every state:

Every trace is finite (what termination provers do)	There exists an infinite trace
There exists a finite trace	Every trace is infinite

A sub-problem of showing non-termination

- We search for a set of states that the program cannot escape – a *recurrent set*.
- Recurrent sets can be characterized as fixed points of backward transformers.
- Because of incompleteness, we may not be able to find the largest set.
- To show non-termination, we would need to show reachability of this set from the initial states. *We do not do it*.

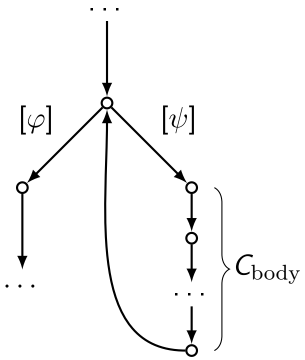
Recurrent set of a loop

We search for recurrent sets of individual loops:

$R_\forall$ satisfies $\neg\varphi$

$\forall s \in R_\forall.\, \big(\forall s'.\, (s, s') \in [\![C_{\mathrm{body}}]\!] \Rightarrow s' \in R_\forall\big)$

Under reasonable assumptions, every execution from $R_\forall$ is infinite.



Recurrent sets with forward analysis

Can we restrict ourselves to a forward over-approximating analysis and still be good?

- Forward analyses have more features, e.g., more abstract domains are available.
- For example, for separation logic, backward analysis is known to be harder (Calcagno, Yang, and O'Hearn 2001).
- We used shape analysis with 3-valued logic (Sagiv, Reps, and Wilhelm 2002). It is less popular, but a good representative of non-numeric abstract domain.

Recurrent sets with forward analysis

(Recap of) Goals

- Find recurrent sets of individual loops.
- Forward analysis.
- Prove non-termination of “textbook” numeric programs. They often rely on unbounded numbers.
- Prove non-termination of some heap-manipulating programs.

Sketch of the analysis

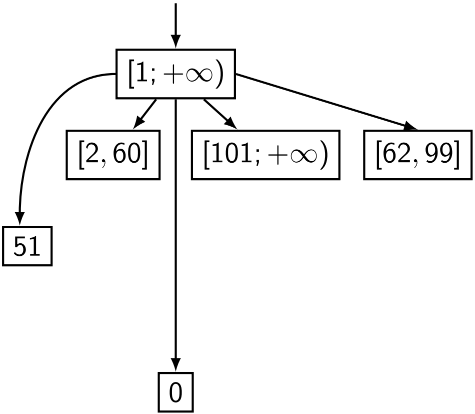
Assuming unbounded integers

$$\downarrow$$
$$\boxed{[1;+\infty)}$$

while $x \geq 1$:
  if $x = 60$: $x \leftarrow 50$
  $x \leftarrow x + 1$
  if $x = 100$: $x \leftarrow 0$

Sketch of the analysis

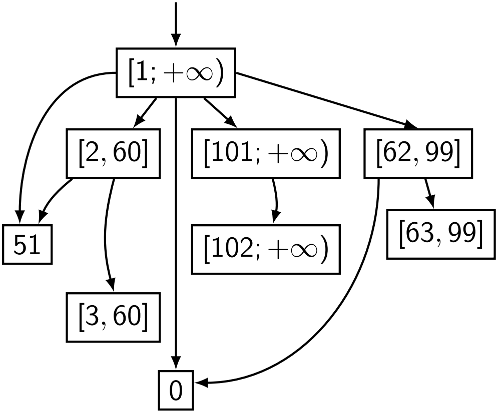
Assuming unbounded integers




```
while x ≥ 1:
  if x = 60:  x ← 50
  x ← x + 1
  if x = 100:  x ← 0
```

Sketch of the analysis

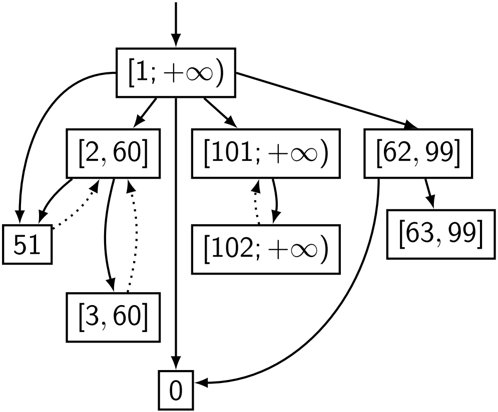
Assuming unbounded integers



while $x \geq 1$:
  if $x = 60$: $x \leftarrow 50$
  $x \leftarrow x + 1$
  if $x = 100$: $x \leftarrow 0$

Sketch of the analysis

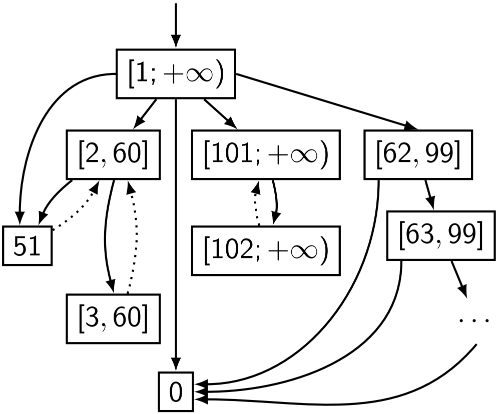
Assuming unbounded integers




```
while x ≥ 1:
  if x = 60:  x ← 50
  x ← x + 1
  if x = 100:  x ← 0
```

Sketch of the analysis

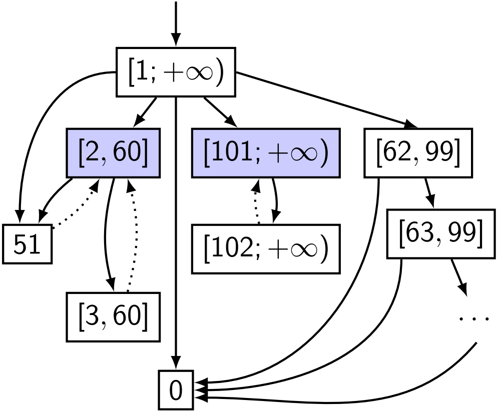
Assuming unbounded integers




```
while x ≥ 1:
  if x = 60:  x ← 50
  x ← x + 1
  if x = 100:  x ← 0
```

Sketch of the analysis

Assuming unbounded integers, note how states in $[101; +\infty)$ are not re-visited



while $x \geq 1$:
  if $x = 60$: $x \leftarrow 50$
  $x \leftarrow x + 1$
  if $x = 100$: $x \leftarrow 0$

Recurrent sets with forward over-approximation

- Seems, we cannot *characterize* a recurrent set via a fixpoint of forward transformers.
- Intuitively, we would characterize states that have infinite traces *into* them. Not suitable when infinite traces do not re-visit states.
- Instead, we produce a condition:

$$\forall s \in R_\forall. \left(\forall s' \, (s, s') \in [\![C_{\mathrm{body}}]\!] \Rightarrow s' \in R_\forall\right)$$

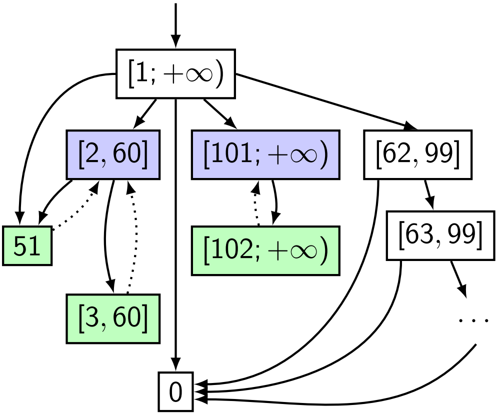
$$\Leftrightarrow \quad post(C_{\mathrm{body}}, R_\forall) \subseteq R_\forall$$

$$\Leftarrow \quad \boxed{post^{\mathcal{D}}(C_{\mathrm{body}}, d_\forall) \sqsubseteq_{\mathcal{D}} d_\forall}$$

In domain $\mathcal{D}$, with $\gamma(d_\forall) = R_\forall$

Sketch of the analysis

Assuming unbounded integers



- $\mathcal{D}$ is a finite powerset domain.
- A condition for $d_\forall$ to represent a recurrent set: $post^{\mathcal{D}}(C, d_\forall) \sqsubseteq_{\mathcal{D}} d_\forall$.
- Exploration via symbolic execution.
- A tractable way to find suitable subsets.

Conclusions

- Tractable way to find recurrent sets of abstract states.
- We need for the recurrent set to be materialized in the state graph.
  - When non-terminating traces take specific branching choices (seems to often be the case), simple symbolic execution works.
  - In shape analysis with 3-valued logic, abstract transformers themselves make relevant case splits.
- For more complicated cases, tailored heuristics would be needed. Currently, *we do not have them*.

Future(?) work

- Upgrade to abstract interpretation.
- For more complicated cases, heuristics for state partitioning would be needed. Currently, *we do not have those*.

```
k = //nondet
while x > 0:
    x ← x + k
```

```
while x > 0:
    x ← −2x + 9
```

- Obviously, cannot deal with too much nondeterminism (no universal recurrent set in the below).

```
while x > 0:
    k = //nondet
    x ← x + k
```

Future(?) work

- Upgrade to abstract interpretation.
- For more complicated cases, heuristics for state partitioning would be needed. Currently, *we do not have those*.

```
k =//nondet
while x > 0:
  x ← x + k
```

```
while x > 0:
  x ← −2x + 9
```

- Obviously, cannot deal with too much nondeterminism (no universal recurrent set in the below).

```
while x > 0:
  k =//nondet
  x ← x + k
```

Thanks

Related work

- (Brockschmidt et al. 2011) Implemented in AProVE. Builds a similar graph, but the rest is different.
- (Cook et al. 2014) Finds universal recurrent sets in over-approximated linear programs via Farkas' lemma.
- (Velroyen and Rümmer 2008) Invel. One of the early analyses, and a set of bechmarks.