Backward Analysis via Over-Approximate Abstraction and Under-Approximate Subtraction

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

[1]University of Leicester, Department of Computer Science

[2]Microsoft Research

University of Leicester

Microsoft Research

Goal

A backwards analysis inferring sufficient preconditions for safety.

```
while (x) {
    /* Possible invalid pointer */
    x = x->next;
    /* Possible null dereference */
    x = x->next;
}
```

Goal

A backwards analysis inferring sufficient preconditions for safety.

```
while (x) {
    /* Possible invalid pointer */
    x = x->next;
    /* Possible null dereference */
    x = x->next;
}
```

- In our model, unsafe actions bring the program to an error memory state.

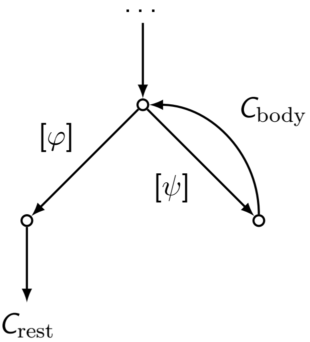
Goal

A backwards analysis inferring sufficient preconditions for safety.

```
while (x) {
    /* Possible invalid pointer */
    x = x->next;
    /* Possible null dereference */
    x = x->next;
}
```

- In our model, unsafe actions bring the program to an error memory state.
- General technique applicable to more than one domain.
- Hence, assume that backward transformers can be designed.
- Intraprocedural (I'll be mostly talking about loops).

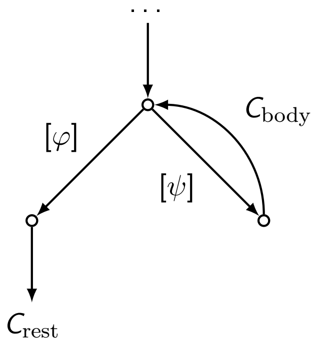
A loop




```
...
while (f(state)) {
  /* Loop body */
  ...
}
/* Rest of procedure */
...
```

Standard: gfp

C_{frag}:



An input state makes C_{frag} safe when

$$\varphi \Rightarrow (C_{\text{rest}} \text{ is safe})$$

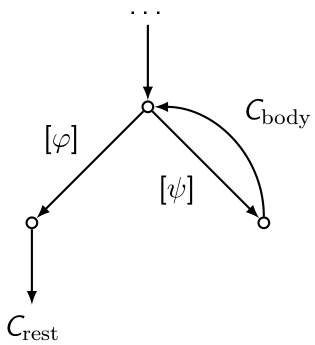
and

$$\psi \Rightarrow (C_{\text{body}} \,;\, C_{\text{frag}} \text{ is safe})$$

Leads to a system of recursive equations where (an under-approximation of) the greatest solution is of interest.

Standard: complement of an lfp

C_{frag}:



An input state makes C_{frag} unsafe when an unsafe state is reachable

$\varphi \wedge (C_{\text{rest}}$ is unsafe)

or

$\psi \wedge (C_{\text{body}}\ ;\ C_{\text{frag}}$ is unsafe)

- Find (an over-approximation of) the least solution of the resulting recursive equations.
- Complement the result.

Why alternative formulation?

Why not gfp?

Domains are often geared towards least fixed points and over-approximation. For example:

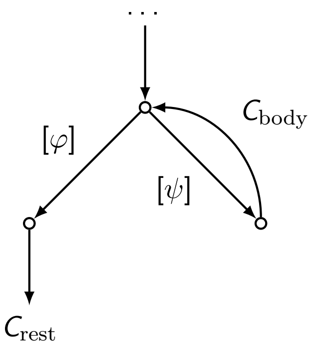
- For shape analysis with 3-valued logic (Sagiv, Reps, and Wilhelm 2002), over-approximation is the default way of ensuring convergence.
- For polyhedra, direct under-approximating analysis uses a different approach to representing states (Miné 2012).

Why not complement of lfp?

- Under-approximating complementation may not be readily supported (e.g., 3-valued structures).

Our formulation

C_{frag}:

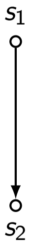


- Walk backwards.
- Over-approximate the unsafe states (*negative side*).
- Characterize the safe states (*positive side*) as an lfp above a *recurrent set*.
- Use the negative side to prevent over-approximation of the positive side.

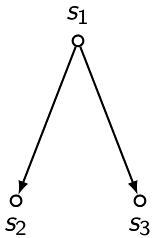
Semantics of statements

- $\mathcal{U}$ – all *memory* states, ϵ – a disjoint error state.
- For a statement, $[\![C]\!] \subseteq \mathcal{U} \times (\mathcal{U} \cup \{\epsilon\})$.
- Loop semantics is an lfp.

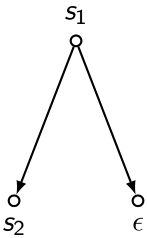
$x = x + 1$



$x = x + [1; 2]$



$x = 2x/[0, 1]$



Positive and negative sides

$P(C_{\mathrm{prg}}, \mathcal{U})$ is the goal, and $N(C_{\mathrm{prg}}, \varnothing)$ is its inverse. The analysis uses both.

Positive side $P(C, S)$

- Safe states assuming S is safe after the execution.
- Corresponds to *weakest liberal precondition*.
- $wp(C, S) = \{s \in \mathcal{U} \ \mid\ \forall s' \in \mathcal{U} \cup \{\epsilon\}.\ [\![C]\!](s, s') \ \Rightarrow\ s' \in S\}$

Negative side $N(C, V)$

- Unsafe states, assuming V is unsafe after the execution.
- Corresponds to the union of *predecessors* and *unsafe states*.
- $pre(C, V) = \{s \in \mathcal{U} \ \mid\ \exists s' \in V.\ [\![C]\!](s, s')\}$
- $fail(C) = \{s \in \mathcal{U} \ \mid\ [\![C]\!](s, \epsilon)\}$

Positive and negative sides

$P(C_{\mathrm{prg}}, \mathcal{U})$ is the goal, and $N(C_{\mathrm{prg}}, \varnothing)$ is its inverse. The analysis uses both.

Positive side $P(C, S)$

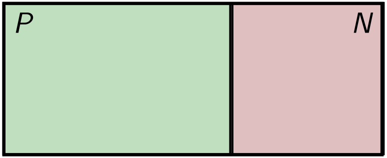
- Safe states assuming S is safe after the execution.
- $P(C, S) = wp(C, S)$
- Has a standard characterization as a gfp.
- We restate it as an lfp.

Negative side $N(C, V)$

- Unsafe states, assuming V is unsafe after the execution.
- $N(C, V) = pre(C, V) \cup fail(V)$
- Has a standard characterization as an lfp.

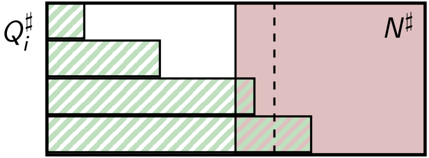
Under-approximating the positive side

- Over-approximate negative side $N^\sharp$ computed as usual (moving to an abstract domain with ascending chain condition or widening).
- Lfp-characterization of the positive side gives rise to an ascending chain of over-approximate positive side $Q_i^\sharp$.
- Subtraction of the negative side produces a sequence of under-approximate positive side $P_i^\flat$, from which one element (e.g., final) is picked.



Under-approximating the positive side

- Over-approximate negative side $N^\sharp$ computed as usual (moving to an abstract domain with ascending chain condition or widening).
- Lfp-characterization of the positive side gives rise to an ascending chain of over-approximate positive side $Q_i^\sharp$.
- Subtraction of the negative side produces a sequence of under-approximate positive side $P_i^\flat$, from which one element (e.g., final) is picked.



Under-approximating the positive side

- Over-approximate negative side $N^\sharp$ computed as usual (moving to an abstract domain with ascending chain condition or widening).
- Lfp-characterization of the positive side gives rise to an ascending chain of over-approximate positive side $Q_i^\sharp$.
- Subtraction of the negative side produces a sequence of under-approximate positive side $P_i^\flat$, from which one element (e.g., final) is picked.

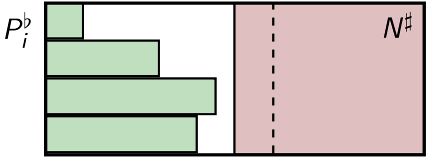
Abstract subtraction

Function $(\cdot - \cdot) \colon \mathcal{L} \to \mathcal{L} \to \mathcal{L}$ such that for $l_1, l_2 \in \mathcal{L}$

- $\gamma(l_1 - l_2) \subseteq \gamma(l_1)$
- $\gamma(l_1 - l_2) \cap \gamma(l_2) = \varnothing$

Under-approximating the positive side

- Over-approximate negative side $N^\sharp$ computed as usual (moving to an abstract domain with ascending chain condition or widening).
- Lfp-characterization of the positive side gives rise to an ascending chain of over-approximate positive side $Q_i^\sharp$.
- Subtraction of the negative side produces a sequence of under-approximate positive side $P_i^\flat$, from which one element (e.g., final) is picked.



Under-approximating the positive side

- Over-approximate negative side $N^\sharp$ computed as usual (moving to an abstract domain with ascending chain condition or widening).
- Lfp-characterization of the positive side gives rise to an ascending chain of over-approximate positive side $Q_i^\sharp$.
- Subtraction of the negative side produces a sequence of under-approximate positive side $P_i^\flat$, from which one element (e.g., final) is picked.

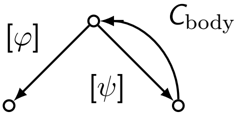
Abstract subtraction

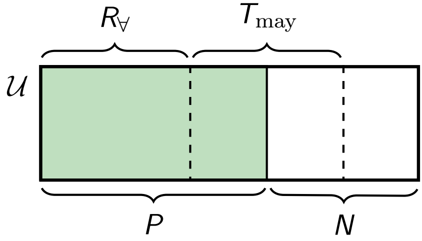
We claim that it is easier to implement than complementation.
E.g., for a powerset domain $\mathcal{P}(\mathcal{L})$ a coarse one can be used:

$$L_1 - L_2 = \{l_1 \in L_1 |\ \forall\, l_2 \in L_2.\ \gamma(l_1) \cap \gamma(l_2) = \varnothing\}$$

Positive side via universal recurrence

C_{loop}:





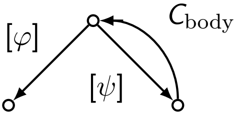
- $R_\forall$ – *universal recurrent set* (states that must cause non-termination):

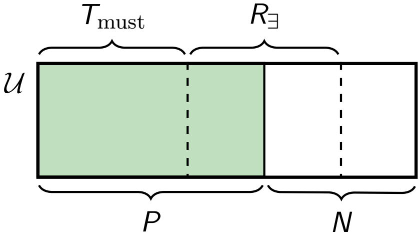
$$R_\forall \subseteq [\![\neg\varphi]\!]$$
$$\forall s \in R_\forall.\, \left(\forall s' \in \mathcal{U} \cup \{\epsilon\}.\, [\![C_{\text{body}}]\!](s, s') \Rightarrow s' \in R_\forall\right)$$

- T_{may} – states that may cause successful termination. An lfp involving *pre*.
- Characterize P as lfp involving $pre \setminus N$ above $R_\forall$.

Positive side via existential recurrence

C_{loop}:



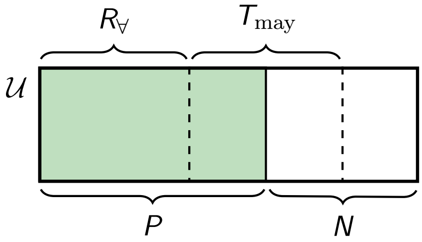


- $R_\exists$ – *existential recurrent set* (states that may cause non-termination):

$$R_\exists \subseteq [\![\psi]\!]$$
$$\forall s \in R_\exists.\ \exists s' \in R_\exists.\ [\![C_{\text{body}}]\!](s, s')$$

- T_{must} – states that must cause succesful termination. An lfp involving *wp*.
- Characterize P as lfp involving *wp* above $R_\exists \setminus N$.

Positive side via recurrence



- P characterized as lfp above a recurrent set.
- We claim that finding a recurrent set is a less general problem than approximating a gfp.
- Recurrent set is produced by an external procedure.

Evaluation

We evaluated the approach on simple examples of the level of

```
while (x) {
    x = x->next;
}
```

```
while (x ≥ 1) {
    if (x == 60)
        x = 50;
    ++x;
    if (x == 100)
        x = 0;
}
assert(!x);
```

- E-HSF (Beyene, Popeea, and Rybalchenko 2013) used to produce recurrent sets for numeric programs.
- An internal prototype procedure based on TVLA (Lev-Ami, Manevich, and Sagiv 2004) – for heap-manipulating programs.

Conclusion

- Theoretical construction based on recurrent sets and subtraction.
- Prototype implementation for two domains.
- Possible future work.
  - Lifting restrictions (program language, nested loops).
  - Recurrence search for various domains.
  - Feasibility of abstract counterexamples.
- Check out our technical report.

Thank you

Related work

- (Lev-Ami et al. 2007) – backwards analysis with 3-valued logic, via complementing an lfp.
- (Calcagno et al. 2009) – inferring pre-conditions with separation logic, bi-abduction, and over-approximation.
- (Popeea and Chin 2013) – numeric analysis with positive and negative sides.
- (Miné 2012) – backwards analysis with polyhedra and gfps.
- (Beyene, Popeea, and Rybalchenko 2013) – an solver for quantified Horn clauses allowing to encode search for pre-conditions in linear programs.