Verification of Top-Down Solvers

Sarah Tilscher
sarah.tilscher@tum.de

Supervisors: Helmut Seidl & Tobias Nipkow

Collaborators: Yannick Stade & Michael Schwarz (ConVeY) & Ralf Vogler & Helmut Seidl

ConVeY

Setting

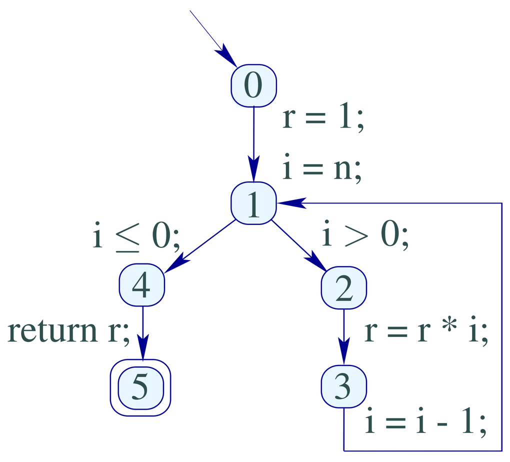
Compute a solution for a constraint system

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n)$$

where x_i are unknowns (points of interest) and
f_i are the constraints to be satisfied

Used for example for static program analysis:

```
int factorial(int n)
{
  int r = 1;
  int i = n;
  while(i > 0) {
    r = r * i;
    i = i - 1;
  }
  return r;
}
```



$$\mathcal{A}[1] \sqsupseteq [\![r = 1; i = n;]\!]^\sharp\, \mathcal{A}[0]$$
$$\sqcup\ [\![i = i - 1;]\!]^\sharp\, \mathcal{A}[3]$$
$$\mathcal{A}[2] \sqsupseteq [\![i > 0;]\!]^\sharp\, \mathcal{A}[1]$$
$$\mathcal{A}[3] \sqsupseteq [\![r = r * i;]\!]^\sharp\, \mathcal{A}[2]$$
$$\ldots$$

Motivation

Advanced solving strategies complicate reasoning about solver correctness. Implementations are often fragile and vulnerable to bugs.

→ Formal verification to justify the correctness

Top-Down Solver

- **Generic** fixpoint algorithm
- Computes partial solution for the queried unknown of interest and all unknowns it depends on
- Tracks dependencies between unknowns on-the-fly
- An improved version [3] is used by the static analyzer GOBLINT (**goblint/analyzer**)

```
let solve eq x =
    let rec iterate y =
        let query z =
            if z ∉ called:
                called := called ∪ {z}
                iterate z
                called := called - {z}
            infl := infl[z ↦ infl(z) ∪ {y}]
            σ(z) in

        if y ∉ stable:
            stable := stable ∪ {y}
            let d = eq y query in
            if d ≠ σ(y):
                σ := σ[y ↦ d]
                destabilize y
                iterate y in

    called := called ∪ {x}
    iterate x
    σ
```

```
let rec destabilize y =
    let work = infl(y) in
    infl := infl[y ↦ ∅];
    forall z ∈ work:
        stable := stable - {z};
        destabilize z
```

Objective

Prove the correctness of the solver's solution σ, i.e.
$\sigma\ x_i\ \sqsupseteq f_i\ (\sigma\ x_1, \ldots, \sigma\ x_n)$ for all $x_i \in reach$
where $reach$ is the set of unknowns evaluated during the final iteration

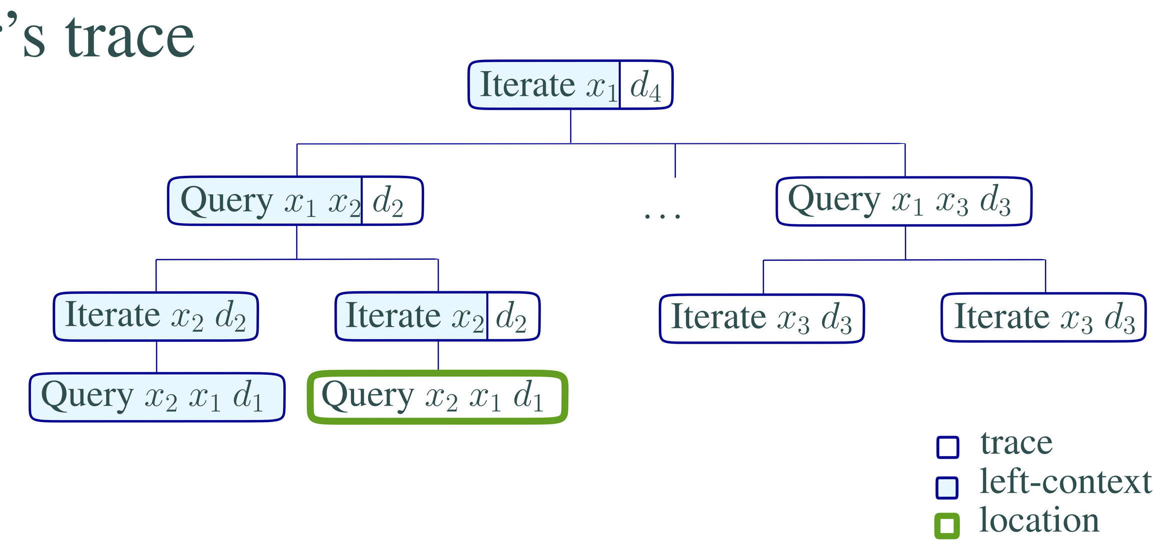
Approach

Apply abstract² interpretation [2]

1. Start with a trivially correct solver:

```
solve eq x =
    rec query y = eq y query
    query x
```

2. Add state as abstraction of the left-context of the solver's trace



The trace describes nested function calls with values of parameters and return values for a call to `solve`.

3. Build proofs with inductions over the trace
4. Introduce optimizations based on the state

Optimizations

- Reduce computation (skip unnecessary re-evaluations)
  - track stable unknowns that are unaffected since their last call to solve
  - track the values of unknowns with which the right-hand side was last evaluated (no re-evaluation when the values of influencing unknowns did not change)
- Reduce space
- Introduce widening and narrowing
- Introduce side-effects [1]

Future Work

- Design precision improvements based on self-observation
- Verify the TD-solver with side-effects
- Generic framework for verifying fixpoint algorithms

References

[1] K. Apinis et al. "Side-Effecting Constraint Systems: A Swiss Army Knife for Program Analysis". In: *APLAS 2012*. Vol. 7705. LNCS. Springer, 2012, pp. 157–172.
[2] P. Cousot et al. "A²I: Abstract² Interpretation". In: *Proc. ACM Program. Lang.* 3.POPL (2019), 42:1–42:31.
[3] H. Seidl et al. "Three Improvements to the Top-Down Solver". In: *Mathematical Structures in Computer Science* (2022), 1–45.