Unfolding Expressions for Gradual Verification

Hazel Torek, Long Tien Nguyen, Jonathan Aldrich

Motivation

- When formally verifying a program, functions must be annotated with:
  - **preconditions:** true **before** the function
  - **postconditions**: must be proven true **after** the function finishes
  - For better modularity, conditions can include **predicates** to **group** and **name** facts
- With static formal verification, programs must to be fully specified → **large upfront cost** for developers

Gradual verification allows developers to write **partially complete** specifications, which improves usability.

Research Objective

- **Unfolding expressions** temporarily open a predicate instance to access the facts inside. These allow writing more modular specifications.
- Our goal is to:
  - **Formalize** the behavior of unfolding expressions.
  - **Update the proof of soundness** for the verifier to include unfolding expressions.
- Our proof of soundness will be the first to include unfolding expressions and apply to **both static and gradual verifiers**.

```
struct Account { int balance; };
struct Withdrawal { int amount; };
```

The **predicate definition** `greaterOrEqual` groups facts which represent that the balance in the account is greater than the withdrawal.

```
/*@ predicate greaterOrEqual(struct Account* a,
                              struct Withdrawal* w) =
    acc(a->balance) && acc(w->amount) &&
    a->balance >= w->amount && w->amount >= 0; @*/
```

The **accessibility predicates** check that the fields `a->balance` and `w->amount` can be accessed before using their values.

The **imprecise specification**, written with ?, allows the developer to write partially complete specifications that are filled in by the verifier.

```
void deductOne(struct Account* a,
               struct Withdrawal* w)
/*@ requires ? && unfolding greaterOrEqual(a, w) in
     (w->amount == 1);
    ensures greaterOrEqual(a, w) &&
     unfolding greaterOrEqual(a, w) in
      (a->balance >= 1); @*/
{ … }
```

The **unfolding expression** is used to open the predicate `greaterOrEqual(a, w)` and use the relevant facts to show `a->balance >= 1`.

Selected Symbolic Evaluation Semantics

SEvalUnfoldingPreciseA

$$\frac{\begin{array}{c}\neg\iota(\sigma_1) \\ \text{predicate}(p) \text{ is precise} \qquad p \notin \mathcal{V}(\sigma_1) \qquad \overline{\sigma_1 \vdash e \Downarrow t \dashv \sigma_2,\ \mathcal{R}_1} \\ \sigma_2 \vdash p(\overline{e}) \rhd \sigma_3,\ \mathcal{R}_2 \qquad \overline{x} = \text{predicate\_params}(p) \\ \sigma_3[\gamma = \gamma(\sigma_3)[\overline{x \mapsto t}], \mathcal{V} = \mathcal{V}(\sigma_3) \cup \{p\}] \vdash \text{predicate}(p) \lhd \sigma_4 \\ \sigma_4[\gamma = \gamma(\sigma_3)] \vdash e_0 \Downarrow t_0 \dashv \sigma_5,\ \mathcal{R}_3 \\ \mathcal{H}' = \mathcal{H}(\sigma_2) \cup \mathcal{H}(\sigma_5) \qquad \sigma_6 = \sigma_5[\mathrm{H} = \mathrm{H}(\sigma_2), \mathcal{H} = \mathcal{H}'] \end{array}}{\sigma_1 \vdash \texttt{unfolding}\ p(\overline{e})\ \texttt{in}\ e_0 \Downarrow t_0 \dashv \sigma_6,\ \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3}$$

Dynamic Semantics

EvalUnfolding

$$\frac{\langle H, \rho\rangle \vdash e_0 \Downarrow v}{\langle H, \rho\rangle \vdash \texttt{unfolding}\ p(\overline{e})\ \texttt{in}\ e_0 \Downarrow v}$$

FrameUnfolding

$$\frac{\langle H, \alpha, \rho\rangle \vdash_{\text{frmI}} p(\overline{e}) \qquad \langle H, \alpha, \rho\rangle \vDash p(\overline{e}) \qquad \langle H, \alpha, \rho\rangle \vdash_{\text{frm}} e_0}{\langle H, \alpha, \rho\rangle \vdash_{\text{frm}} \texttt{unfolding}\ p(\overline{e})\ \texttt{in}\ e_0}$$

$$[\![\texttt{unfolding}\ p(\overline{e})\ \texttt{in}\ e_0]\!]_{\langle H,\rho\rangle} = [\![p(\overline{e})]\!]_{\langle H,\rho\rangle} \cup [\![e_0]\!]_{\langle H,\rho\rangle}$$

Soundness

Soundness means that the verifier only accepts programs which meet their specifications.

- We need to expand the proof of soundness to include unfoldings.

Future Work

- There are many features to implement and formalize in gradual verification, such as quantified specifications and pure functions.
- We will also continue to update the proof of soundness.