Imprecision Separation for Gradual Program Verification with Implicit Dynamic Frames

Kaige Liu

1 Introduction

Program verification is the process to ensure the correctness of a program by examining the program against a set of specifications. Program verification is becoming increasingly important in ensuring correctness and safety of program as the scale grows larger and the cost of manual verification increases. Current approaches to program verification are mostly based on either static verification, which is performed at compile time, or dynamic verification, which is performed at runtime.

1.1 Static Verification

Static verification tools verify a program without running any code. Current approaches to static verification are mostly based on formal methods like Hoare Logic [3], so the correctness of the program can be established with contracts formally. The limitation of static verification is that when contracts are insufficient or incomplete, the quality of static verification can be low in many cases. Insufficiency or incompleteness in contract specification will cause a purely static verifier to fail in many trivial cases.

1.2 Dynamic Verification

Dynamic verification is to verify whether the environment and states of the program adhere to its specification at runtime. This new approach effectively reduces the overhead and results in really high precision by running contracts directly at runtime. However, dynamic verification will create a massive performance overhead on the source program, since all verification is performed at runtime with the source program.

1.3 Gradual Verification

The idea of gradual verification [1] is to explore a combination of static and dynamic verification and utilize the advantages of both. Gradual verification introduces the concept of an imprecise contract, which can be manually specified by the user. Imprecise gradual formulas are defined as $? * \phi$, where ϕ is a precise formula, and the whole formula represents an unknown formula with ϕ

as the known part, and some unknown part ? that can be any formula that does not contradict the existing formula.

On one hand, by checking a gradually specified program, the verifier can warn the programmer of inconsistencies between specifications and code including contracts that contradict the code and contracts that are too weak to perform further analysis on; on the other hand, the static checker will not produce warnings from any contract that is specified manually to be imprecise, a dynamic verification is used instead in these cases. Gradual verification, compared to static verification, improve the preciseness of verification compared to purely static verification, and is more efficient than a purely dynamic verification. Moreover, gradual verification allows contracts to be added gradually in the program, allowing the user to run the verification tool given any degree of contract specification.

1.4 Implicit Dynamic Frame

Hoare logic has been extended to more powerful logics like separation logic [4] and implicit dynamic frames [5]. Implicit Dynamic Frames is a close relative to separation logic, in which we use the notion `acc(e.f)` to indicate a method with ability to access a certain field. An accessibility predicate `acc(e.f)` denotes exclusive access to the field `e.f`. It justifies accessing e.f both in the source code (e.g. `x.f := 3` or `y := x.f`) and in the formula itself (e.g. `acc(x.f) * (x.f == 4)`).

Specifications with accessibility predicates are crucial to the reasoning about the heap. Consider the following example OOP style code:

```
Class Counter {
     int x;
     Counter()
          requires acc(this.x)
          ensures this.x == 0 * acc(this.x) {
          this.x = 0;
     }
     void addOne()
          requires acc(this.x)
          ensures result = old(this.x) + 1 * acc(this.x) {
          this.x += 1;
     }
}
void main() {
     Counter counter1 = new Counter();
     Counter counter2 = new Counter();
     counter1.addOne();
     assert(counter1.x == 1);
     assert(counter2.x == 0);
}
```

It looks obvious that the assertions will hold, since we only call `addOne` on `counter0`. However,

since the heap is accessible to all methods in most modern object-oriented programming language, for a static verifier to reason about the heap, it's necessary to know which fields are modified by each method. We introduce accessibility predicates `acc(this.x)` in the method `addOne` to indicate the method will only access the field `this.x`. Therefore, when verifiers static verfies the line `counter1.x`, it can ensure that `counter2.x` is not modified by calling `counter1.addOne`, and therefore `counter2.x` stays 0.

Additionally, Implicit Dynamic Frames also provides aliasing information: whether two fields refer to the same memory location. Consider the following program:

```
Class Number {
     int x;
     Number(int val)
          requires acc(this.x)
          ensures this.x == 0 * acc(this.x) {
          this.x = val;
     }
}
void exchange3(Number a, Number b, Number c)
     requires acc(a.x) * acc(b.x) * acc(c.x)
     ensures acc(a.x) * acc(b.x) * acc(c.x)
            * b.x == old(a.x) * c.x == old(b.x) * a.x == old(c.x) {
     int temp = a.x;
     a.x = c.x;
     c.x = b.x;
     b.x = temp;
}
```

In order to verify the method `exhacnge3` statically, one thing we need to make sure is that `a.x`, `b.x` and `c.x` refer to 3 different fields (otherwise the postcondition will not be met after the exchange). In this case, accessibility predicates ensure the aliasing of the 3 fields.

Previous work by Bader [1] develops a combination of Gradual Verification and Implicit Dynamic Frame, named GVL_{IDF}. It includes the concept of accessibility predicates in formulas, and designs rules to use accessibility predicates to reason about and heap access.

1.5 Imprecision Separation

The use of accessibility predicates with the introduction of Implicit Dynamic Frames largely improve the precision when the static verifier reasons about heaps. However, one practical issue is that accessibility predicates are really lengthy and difficult to write in real-life settings. Moreover, in some cases, the set of fields that a method will access might not be decidable before execution.

The problem can be solved with previous work on Gradual verification, which examines the combination of Gradual Verification syntax and Implicit Dynamic Frames, where the unknown part of the gradual formula $? * \phi$ can be a combination of classical formulas (e.g. `e.f == 1`) and accessibility

predicates (e.g. `acc(e.f)`). One solution to this problem is to place a question mark for every contract with unspecified accessibility predicates. Consider the following example:

```
Class Account {
     int balance;
     Account(int balance) {
          requires acc(this.balance) * balance ≥ 0
          ensures acc(this.balance) * this.balance == balance) {
          this.balance = balance;
     }
     void withdraw(int amount)
          requires acc(this.balance) * (this.balance ≥ amount)
          ensures ?  * acc(this.balance) * (this.balance ≥ 0) {
          int newBalance = this.balance - amount;
          this.balance = newBalance;
     }
}
void main() {
     Account account = new Account(100);
     account.withdraw(40);
     account.withdraw(40);
}
```

The program models a bank account, where each account has a single field balance, and a method withdraw that reduces the balance by a certain amount. Consider if we want to eliminate the accessibility predicates in the method `withdraw` by replacing the accessibility predicates with `?`:

```
void withdraw(int amount)
     requires ?  * (this.balance ≥ amount)
     ensures ?  * (this.balance ≥ 0) {
     int newBalance = this.balance - amount;
     this.balance = newBalance;
}
```

Such an approach will succeessfully reduce the complexity of accessibility predicate specification, but will largely introduce imprecision into the program, where the classical parts of the contracts are precise originally (e.g the precondition of the method).

In this paper, we will introduce the idea of imprecision separation. Imprecision separation redefines gradual formulas with 2 types of question marks: $?_A$ indicating accessibility imprecision, and $?_C$ indicate classical imprecision. Now we consider an alternative program using the syntax with imprecision separation. We replace all accessibility predicates with $?_A$, and classical imprecision with $?_C$.

```
void withdraw(int amount)
     requires ?A * (this.balance ≥ amount)
     ensures ?A * (this.balance ≥ 0) * ?C {
     int newBalance = this.balance - amount;
     this.balance = newBalance;
}
```

The new program uses 2 different notions of imprecision. As a result, the precondition of `withdraw`

will have no classical imprecision, and therefore the method is specified more precisely than the previous one.

Another use case of imprecision separation, as mentioned previously, is when the set of fields accessed by a method cannot be determined. Consider the following example involving recursive functions:

```
Class ListNode {
    int val;
    ListNode next;
    ListNode findEnd()
        requires ?A
        ensures ?A * result != null * result.next == null{
        if (this.next == null) {
            return this;
        } else {
            return this.next.findEnd();
        }
    }
}
```

Before we call the method `findEnd`, it's difficult to determine the set of fields we are accessing, which all nodes in a linked list. The use of imprecision separation again helps us introduces accessibility imprecision without adding unnecessary classical imprecision.

1.6 Contributions

To demonstrate the flexibility of our imprecision separation approach, we extend GVL_{IDF} 2 different types of question marks. We propose the exact new definition of the formula syntax in section 2. In later section, we propose modifications to of GVL_{IDF} and show that the modified version satisfies the key properties of GVL_{IDF}: consistent formula lifting (Section 3), consistent implication lifting (Section 4), equivalence of dynamic semantics ((Section 5)), finally soundness(Section 6) and gradual guarantee(Section 7).

2 Syntax

We define $\textsc{OrdFormula} \subseteq \textsc{Formula}$ as the set of formulas that the accessibility predicate of any field appears before its usage. Formally,

$$\textsc{OrdFormula} = \{\phi_1 * \phi_2 ... * \phi_n \mid \forall 1 \leq i < j \leq n, \lfloor \phi_j \rfloor \not\vdash \phi_i\}$$

We define $\widetilde{\phi} \in \widetilde{\textsc{Formula}}$, given $\phi \in \textsc{OrdFormula}$, as

$$\widetilde{\phi} ::= \widehat{\phi} \mid ?_A * \phi \mid \widehat{\phi} * ?_C \mid ?_A * \phi * ?_C$$

Notice that $\widehat{\phi}$ must be self-framed and ϕ might not be self-framed. We use ϕ with $?_A$ since $?_A$ can provide framing for the later parts of the formula. We require ϕ to be an ordered formula so that there is a potential set of accessibility predicates that can frame the formula.

We defien the concretization of a gradual formula as the set of static formulas that it can represent. To define concretization, we first define the classical and accessibility part of a formula.

Definition 2.1. *We define ϕ_A as the accessibility parts, and ϕ_C as the classical parts of the formula ϕ, conjuated by $*$.*

Lemma 2.2. *Let $\phi, \phi' \in$ SatFormula. Then:*

$$\phi_C \Rightarrow \phi'_C \wedge \phi_A \Rightarrow \phi'_A \implies \phi \Rightarrow \phi'$$

Proof. (Proof for Lemma 2.2)
Assume $\phi_C \Rightarrow \phi'_C$ and $\phi_A \Rightarrow \phi'_A$. Take any formula component ϕ'' separted by * from ϕ' .

- If $\phi'' = \text{acc}(e.f)$, ϕ'' is part of ϕ'_A. Since $\phi \Rightarrow \phi_A$, $\phi_A \Rightarrow \phi'_A$ and $\phi'_A \Rightarrow \phi''$, then $\phi_A \Rightarrow \phi''$ by transitivity of implication.
- Otherwise, if ϕ'' is not an accessibility predicate, ϕ'' is part of ϕ'_C. Since $\phi \Rightarrow \phi_C$, $\phi_C \Rightarrow \phi'_C$, $\phi'_C \Rightarrow \phi''$, $\phi \Rightarrow \phi''$ by transitivity of implication. componenet

Therefore, each component ϕ'' of ϕ' is implied by ϕ. We can conclude $\phi \Rightarrow \phi'$. □

Finally, we propose a concretization definition of gradual formulas.

Definition 2.3. $\gamma : \widetilde{\text{Formula}} \to \mathbb{P}(\text{Formula})$ *is defined as:*

$$\gamma(\widehat{\phi}) = \{\widehat{\phi}\}$$

$$\gamma(?_A * \phi * ?_C) = \begin{cases} \{\widehat{\phi'} \in \text{SatFormula} \mid \widehat{\phi'} \Rightarrow \phi\} & (\phi \in \text{SatFormula}) \\ undefined & otherwise \end{cases}$$

$$\gamma(?_A * \phi) = \begin{cases} \{\widehat{\phi'} \in \text{SatFormula} \mid \widehat{\phi}'_C \Leftrightarrow \phi_C \wedge \widehat{\phi}'_A \Rightarrow \phi_A\} & (\phi \in \text{SatFormula}) \\ undefined & otherwise \end{cases}$$

$$\gamma(\widehat{\phi} * ?_C) = \begin{cases} \{\widehat{\phi'} \in \text{SatFormula} \mid \widehat{\phi}'_A \Leftrightarrow \phi_A \wedge \widehat{\phi}'_C \Rightarrow \phi_C\} & (\widehat{\phi} \in \text{SatFormula}) \\ undefined & otherwise \end{cases}$$

Notice that by our definition, every formula in the concretization implies the static part of the original formula. Also, if the gradual formula is classically precise, the classical part of the gradual formula will also imply the classical part of every formula in the conretization set. Same holds for the accessibility part.

3 Consistent Formula Lifiting

We define evaluation of a formula as the following:

Definition 3.1. *Let* $\cdot \widetilde{\models} \cdot \subseteq \text{MEM} \times \widetilde{\text{FORMULA}}$ *be defined inductively as:*

$$\frac{m \models \widehat{\phi}}{m \widetilde{\models} \widehat{\phi}} \text{ EVALSTATIC}$$

$$\frac{m \models \phi \qquad A \vdash_{frm} \phi \qquad \forall \langle e, f\rangle \in A.m \models acc(e.f)}{m \widetilde{\models} ?_A * \phi * ?_C} \widetilde{\text{EVALGRAD}}$$

$$\frac{m \models \phi \qquad A \vdash_{frm} \phi \qquad \forall \langle e, f\rangle \in A.m \models acc(e.f)}{m \widetilde{\models} ?_A * \phi} \widetilde{\text{EVALGRAD}}_A$$

$$\frac{m \models \phi}{m \widetilde{\models} \phi * ?_C} \widetilde{\text{EVALGRAD}}_C$$

The extra premises in $\widetilde{\text{EVALGRAD}}$ and $\widetilde{\text{EVALGRAD}}_A$ ensures that there is a possible framing of the gradual formula, that is supported by the footprint in m. Our definition is a proper lifting, formalized by the following lemma:

Lemma 3.2. *(Consistent Formula Evaluation)* $\widetilde{\models}$ *is a consistent lifting of* $\models$*. Formally,*

$$\exists \widehat{\phi} \in \gamma(\widetilde{\phi}).m \models \widehat{\phi} \iff m \widetilde{\models} \widetilde{\phi}$$

The rest of this section explicitly deals with the proof of Consistent Formula Evaluation (Lemma 3.2).

Definition 3.3. *Define a partial order on fields* $\leq$: $\text{FIELD} \times \text{FIELD}$ *as* $\langle e, f\rangle \leq \langle e', f'\rangle$ *iff* e' *contains* $e.f$ *in* e' *or* $(e = e') \wedge (f = f')$.

Lemma 3.4. $\leq$ *defines a partial order on* FIELD.

Proof. Proof for Lemma 3.4

- Reflexive: $\langle e, f\rangle \leq \langle e, f\rangle$ since $e = e$ and $f = f$ by definition.
- Anti-symmetric: Given $\langle e', f'\rangle \leq \langle e, f\rangle$ and $\langle e, f\rangle \leq \langle e', f'\rangle$. Suppose $e \neq e'$ or $f \neq f'$. Then if $e = e'$ and $f = f'$, we are done with the proof. Otherwise, assume $e \neq e' \vee f \neq f'$, then e contains $e'.f'$ and e' contains $e.f$, which implies $econtainse.f$. Contradiction.
- Transitive: Given $\langle e, f\rangle \leq \langle e', f'\rangle$ and $\langle e', f'\rangle \leq \langle e'', f''\rangle$. If $e' = e''$ and $f' = f''$, then clearly $\langle e, f\rangle \leq \langle e'', f''\rangle$. Otherwise, e' contains $e''.f''$. Also, e contains $e'.f'$. Therefore, e contains $e''.f''$.

□

Definition 3.5. *Define* $\mu(\cdot)$: $\text{STATFOOTPRINT} \rightarrow \text{FORMULA}$. μ *maps a static footprint* A *to a formula* $\phi' = \mu(A)$*. For each* $\langle e, f\rangle \in A$,$\phi'$ *should contain* $acc(e.f)$*. Then,* ϕ' *is sorted in ascending order with the partial order* $\leq$.

The following lemma describes a property of the minimum footprint of a formula, that A must contain all subchains of any chain of fields $e.f_1.f_2...f_n.f$.

Lemma 3.6. *Let* $\phi \in$ FORMULA*, and* $A = \min_{\subseteq}\{A' \in \text{STATFPRINT} | A' \vdash_{frm} \phi\}$*. Then, if* $\langle e.f_1.f_2...f_n, f\rangle \in A$*, then* $\forall 1 \leqslant m \leqslant n$*,* $\langle e.f_1...f_m - 1, f_m\rangle \in A$*.*

Proof. Proof for Lemma 3.6
Supppose for the sake of contradiction, that for some field access $e.f_1.f_2...f_n, f \in A$, there exists a maximum m, such that $e.f_1.f_2...f_{m-1}, f_m \notin A$. Then, since FFIELD is the only rule that concludes $A \vdash_{frm} e.f$, we can't conclude $A \vdash_{frm} e.f_1.f_2...f_{m-1}.f_m$. Since m is the maximum such that $e.f_1.f_2...f_{m-1}, f_m \notin A$, $e.f_1.f_2...f_m, f_{m+1} \in A$. Since FFIELD is the only rule that uses the fact $\langle e.f_1.f_2...f_m, f_{m+1}\rangle \in A$, but we can't conclude the $A \vdash_{frm} e.f_1.f_2...f_{m-1}.f_m$, removing $\langle e.f_1.f_2...f_m, f_{m+1}\rangle$ from A doesn't change any framing preperty of A. Contradicts that A is minimal. □

The following lemma claims the existence of a self-framed formula with a postfix of any given formula.

Lemma 3.7. *Let* $\phi \in$ ORDFORMULA*, and* $A = \min_{\subseteq}\{A' \in \text{STATFPRINT} | A' \vdash_{frm} \phi\}$*. Then,* $\mu(A) * \phi$ *is self-framed.*

Proof. Proof for Lemma 3.7 First, we prove that $\emptyset \vdash_{frm} \mu(A)$. We proceed by induction on the first k accesibility predicates separated by $*$ of $\mu(A)$.

- **Base case**: k = 0. Clearly, $\emptyset \vdash_{frm}$ true.
- **Inductive step**: Let the first k accessibility predicates of $\mu(A)$, separated by $*$, be $\mu(A)_{1-k}$. Suppose for the first k accessibility predicates of $\mu(A)$, $\emptyset \vdash_{frm} \mu(A)_{[k]}$ (Induction Hypothesis). Let the $k+1$th accessibility predicate be $acc(e.f)$. Need to show $\emptyset \vdash_{frm} mu(A)_{[k+1]}$. We proceed by a sub-induction:
  Sub-Claim: Let $e = e'.f_1.f_2...f_m$**. Then,** $\forall 0 \leqslant n \leqslant m$**,** $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e'.f_1.f_2...f_n$**.**
  - **Sub-Base case**: n = 0. Since $\emptyset \vdash_{frm} e'$ for arbitrary non-field access, $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e'$.
  - **Sub-Inductive case**: For simplification, let $e'' = e'.f_1...f_n$. Suppose for some n, $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e'.f_1.f_2...f_n = e''$. We need to show $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e'.f_1.f_2...f_{n+1} = e''.f_{n+1}$. Since $A \vdash_{frm} \phi$ and A is minimal, by Lemma 3.6, $\langle e'', f_{n+1}\rangle \in A$. By definition of μ (Definition 3.5), $acc(e''.f_{n+1})$ appears in $\mu(A)$. Since $\mu(A)$ is sorted by partial order $\leq$, and e contains $e''.f_{n+1}$, therefore, $acc(e''.f_{n+1})$ must appear before the kth term of $\mu(A)$, so $acc(e''.f_{n+1})$ is in $\mu(A)_{[k]}$. By FFIELD, since $\langle e'', f_{n+1}\rangle \in \lfloor \mu(A)_{[k]} \rfloor$ and $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e''$ by induction hypothesis, $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e''.f_{n+1}$

  Therefore, $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} e$. By SFACC, $A \vdash_{frm} acc(e.f)$. By SFSEPOP, $\emptyset \vdash_{frm} \mu(A)_{[k]}$ and $\lfloor \mu(A)_{[k]} \rfloor \vdash_{frm} acc(e.f)$, $\emptyset \vdash_{frm} \mu(A)_{[k]} * acc(e.f) = \mu(A)_{k+1}$

Therefore, by the induction, we proved that $\emptyset \vdash_{frm} \mu(A)$. By definition of μ, $\lfloor \mu(A) \rfloor = A$. $A \vdash_{frm} \phi$, so $\lfloor \mu(A) \rfloor \vdash_{frm} \phi$. By SFSEPOP, $\vdash_{frm} \mu(A)$, $\lfloor \mu(A) \rfloor \vdash_{frm} \phi$, we can finally conclude that $\emptyset \vdash_{frm} \mu(A) * \phi$.

Addtionally, we prove $\mu(A) * \phi$ is a well-formed formula with no repeated accessibility predicate. Suppose there $acc(e, f)$ appears twice in $\mu(A) * \phi$. Since ϕ is well formed and $\mu(A)$ forms from a set of footprints with no repeat, the 2 $acc(e.f)$ cannot be both in $\mu(A)$ or ϕ. Therefore, one of $acc(e.f)$ is in $\mu(A)$, which means that there must be some part of ϕ that uses $e.f$, and appears before $acc(e.f)$ in ϕ. This is a contradiction to $\phi \in$ OrdFormula. □

Lemma 3.8. *Let* $\widehat{\phi}, \phi' \in$ Formula *such that* $\widehat{\phi} \Rightarrow \phi'$, *and* $A = \min_{\subseteq}\{A' \in$ StatFootprint $\mid A' \vdash_{frm} \phi'\}$. *Then,* $A \subseteq \lfloor\widehat{\phi}\rfloor$

Proof. (Proof for lemma 3.8)
Let $\langle e, f\rangle \in A$ be arbitrary. Since A is the minimum footprint that frames phi', $\exists\phi'_0$ as a component of ϕ' separated by $*$ such that ϕ'_0 contains $e.f$. Let $E = \{\langle H, \rho, A\rangle \in$ env $\mid \rho \models \widehat{\phi}\}$ and $E' = \{\langle H', \rho', A'\rangle \in$ env $\mid \rho \models \phi'\}$. By definition of implication, since $\widehat{\phi} \Rightarrow \phi'$, $E \subseteq E'$. By the grammar of the verification language, ϕ'_0 can be either $e.f \odot k$ or $k \odot e.f$, in either case some possible value of $e.f$ is eliminated. $\exists o$ such that $\forall\langle H', \rho', A'\rangle \in E'$, $H, \rho \vdash e.f \Downarrow v$ and $v \neq o$. Since $E \subseteq E'$, $\forall\langle H, \rho, A\rangle \in E$, $H, \rho \vdash e.f \Downarrow v$ and $v \neq o$. Therefore, $\widehat{\phi}$ must contain a component ϕ_0 separated by $*$, such that ϕ_0 contains $e.f$. Since $\widehat{\phi}$ is self-framed, $\widehat{\phi}$ contains acc$(e.f)$, and therefore $\langle e, f\rangle \in \lfloor\widehat{\phi}\rfloor$.
□

Finally, we provide a proof for consistent formula evaluation:

Proof. (Proof for Lemma 3.2)

Case $\widetilde{\phi} = \phi'$

It follows from definition of γ(2.3) and static that $\gamma(\widehat{\phi}) = \{\phi\}$ and that $static(\widehat{\phi}) = \phi$. Also, $m \models static(\widehat{\phi})$ if and only if $m \widetilde{\models} \widehat{\phi}$(3.1). Therefore, the theorem trivially holds.

Case $\widetilde{\phi} =?_A * \phi$

Applying the definitions of γ and $\widetilde{\models}$, the goal becomes:

$$\exists\widehat{\phi'} \in \text{SatFormula}.(\widehat{\phi'_c} \Leftrightarrow \phi_c) \wedge (\widehat{\phi'_A} \Rightarrow \phi_A) \wedge (m \models \widehat{\phi'}) \iff m \models \phi, \exists A \vdash_{frm} \phi. \forall\langle e, f\rangle \in A. m \models \text{acc}(e.f)$$

- Case $\Rightarrow$: Since $\widehat{\phi'_C} \Rightarrow \phi_C$ and $\widehat{\phi'_A} \Rightarrow \phi_A$, we can conclude $\widehat{\phi'} \Rightarrow \phi$ (by Lemma 2.2). Since $m \models \widehat{\phi'}$, by definition of implication, $m \models \phi$.
  Let $A = \min_{\subseteq}\{A' \in$ StatFootprint $\mid A' \vdash_{frm} \phi\}$. $m \models \widehat{\phi'}$ implies that $\forall\langle e, f\rangle \in \lfloor\widehat{\phi'}\rfloor, m \models acc(e.f)$. We also have $\widehat{\phi'} \Rightarrow \phi$, so $A \subseteq \lfloor\widehat{\phi'}\rfloor$ by Lemma 3.8. Therefore, $\forall\langle e, f\rangle \in A, m \models acc(e.f)$
- Case $\Leftarrow$ Let $A = \min_{\subseteq}\{A' \in$ StatFootprint $\mid A' \vdash_{frm} \phi\}$. Let $\widehat{\phi'} = \mu(A) * \phi$. $\widehat{\phi}$ is self-framed by Lemma 3.7. Since we didn't add anything to the classical part of $\widehat{\phi'}$, $\widehat{\phi'_C} \Leftrightarrow \phi_C$ trivially holds. Since we only add more items to the accessibility part of the formula, $\widehat{\phi'_A} \Rightarrow \phi_A$

holds. Finally, since $\forall \langle e, f\rangle \in A.m \models \text{acc}(e.f)$, by Definition 3.5, we know that $m \models \mu(A)$. Since $m \models \phi$, therefore, $m \models \mu(A) * \phi = \widehat{\phi'}$.

Case $\widetilde{\phi} = \widehat{\phi} * ?_C$

Applying the definitions of γ, the goal becomes:

$$\exists \widehat{\phi'} \in \textsc{SatFormula}.\widehat{\phi'}_A \Leftrightarrow \widehat{\phi}_A \wedge \widehat{\phi'}_C \Rightarrow \widehat{\phi}_C \wedge m \models \widehat{\phi'} \iff m \models \widehat{\phi}$$

- Case $\Rightarrow$: Since $\widehat{\phi'}_C \Rightarrow \widehat{\phi}_C$ and $\widehat{\phi'}_A \Rightarrow \widehat{\phi}_A$, $\widehat{\phi'} \Rightarrow \widehat{\phi}$. Also, since $m \models \widehat{\phi'}$, by definition of implication, $m \models \widehat{\phi}$.
- Case $\Leftarrow$ substitute $\widehat{\phi}$ for $\widehat{\phi'}$. All the 3 clauses on the left hand side trivially hold.

Case $\widetilde{\phi} = ?_A * \phi * ?_C$

Applying the definitions of γ, the goal becomes:

$$\exists \widehat{\phi'} \in \textsc{SatFormula}.\widehat{\phi'} \Rightarrow \phi \wedge m \models \widehat{\phi'} \iff m \models \phi, A \vdash_{frm} \phi, \forall \langle e, f\rangle \in A.m \models \text{acc}(e.f)$$

- Case $\Rightarrow$: Since $\widehat{\phi'} \Rightarrow \phi$ and $m \models \widehat{\phi'}$, by definition of implication, $m \models \phi$.
  Let $A = \min_{\Rightarrow}\{A' \in \textsc{StatFootprint} \mid A' \vdash_{frm} \phi\}$. $m \models \widehat{\phi'}$ implies that $\forall \langle e, f\rangle \in \lfloor \widehat{\phi'} \rfloor, m \models acc(e.f)$. We also have $\widehat{\phi'} \Rightarrow \phi$, so $A \subseteq \lfloor \widehat{\phi'} \rfloor$ by Lemma 3.8. Therefore, $\forall \langle e, f\rangle \in A, m \models acc(e.f)$
- Case $\Leftarrow$: Let $A = \min_{\subseteq}\{A' \in \textsc{StatFootprint} \mid A' \vdash_{frm} \phi\}$. Let $\widehat{\phi'} = \mu(A) * \phi$. $\widehat{\phi}$ is self-framed by Lemma 3.7. Since we didn't add anything to the classical part of $\widehat{\phi'}$, $\widehat{\phi'}_C \Rightarrow \phi_C$ trivially holds. Since we only add more items to the accessibility part of the formula, $\widehat{\phi'}_A \Rightarrow \phi_A$ holds. By Lemma 2.1, $\widehat{\phi'} \Rightarrow \phi$. Finally, since $\forall \langle e, f\rangle \in A.m \models \text{acc}(e.f)$, by definition of μ (Definition 3.5), we know that $m \models \mu(A)$. Since $m \models \phi$, therefore, $m \models \mu(A) * \phi = \widehat{\phi'}$.

□

4 Implication

Definition 4.1. *Let* $\cdot \mathrel{\widetilde{\Rightarrow}} \cdot \subseteq \widetilde{\textsc{Formula}} \times \widetilde{\textsc{Formula}}$ *be defined inductively as:*

$$\frac{\widehat{\phi} \in \textsc{SatFormula} \qquad \widehat{\phi} \Rightarrow \text{static}(\widetilde{\phi'})}{\widehat{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'}}\ \widetilde{\textsc{ImplStatic}}$$

$$\frac{\widehat{\phi} \in \textsc{SatFormula} \qquad \widehat{\phi} \Rightarrow \phi \qquad \widehat{\phi} \Rightarrow \text{static}(\widetilde{\phi'})}{?_A * \phi * ?_C \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'}} \ \widetilde{\textsc{ImplGrad}}$$

$$\frac{\widehat{\phi} \in \textsc{SatFormula} \qquad \widehat{\phi}_C \Leftrightarrow \phi_C \qquad \widehat{\phi}_A \Rightarrow \phi_A \qquad \widehat{\phi} \Rightarrow \text{static}(\widetilde{\phi'})}{?_A * \phi \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'}} \ \widetilde{\textsc{ImplGrad}_A}$$

$$\frac{\widehat{\phi} \in \textsc{SatFormula} \qquad \widehat{\phi}_A \Leftrightarrow \phi_A \qquad \widehat{\phi}_C \Rightarrow \phi_C \qquad \widehat{\phi} \Rightarrow \text{static}(\widetilde{\phi'})}{\phi * ?_C \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'}} \ \widetilde{\textsc{ImplGrad}_C}$$

Lemma 4.2. *Gradual implication can be equivalently defined as the following: For* $\widetilde{\phi}, \widetilde{\phi'} \in \widetilde{\textsc{Formula}}$,

$$\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \widehat{\phi}.\widehat{\phi} \in \gamma(\widetilde{\phi}) \wedge \widehat{\phi} \Rightarrow \text{static}(\widetilde{\phi'})$$

Proof. (Proof for Lemma 4.2)

- Case $\widetilde{\phi'} = \widehat{\phi'}$: By IMPLSTATIC (Definition 4.1), $\exists \widehat{\phi} \in \textsc{SatFormula}.\widehat{\phi} \Rightarrow \mathit{static}(\widehat{\phi'}) \iff \widehat{\phi} \mathrel{\widetilde{\Rightarrow}} \widehat{\phi'}$. By definition of γ (Definition 2.3), $\widehat{\phi} \in \gamma(\widehat{\phi}) \iff \widehat{\phi} \Rightarrow \mathit{static}(\widehat{\phi'})$. Therefore, $\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \widehat{\phi}.\widehat{\phi} \in \gamma(\widetilde{\phi}) \wedge \widehat{\phi} \Rightarrow \mathit{static}(\widetilde{\phi'})$.
- Case $\widetilde{\phi'} = ?_A * \phi' * ?_C$: By IMPLGRAD (Definition 4.1), $\exists \widehat{\phi} \in \textsc{SatFormula}.\widehat{\phi} \Rightarrow \phi' \iff \widehat{\phi} \mathrel{\widetilde{\Rightarrow}} \widehat{\phi'}$. By definition of γ (Definition 2.3), $\widehat{\phi} \in \gamma(\widehat{\phi}) \iff \widehat{\phi} \Rightarrow \phi'$. Therefore, $\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \widehat{\phi}.\widehat{\phi} \in \gamma(\widetilde{\phi}) \wedge \widehat{\phi} \Rightarrow \mathit{static}(\widetilde{\phi'})$.
- Case $\widetilde{\phi'} = ?_A * \phi'$: By IMPLGRAD_A (Definition 4.1), $\exists \widehat{\phi} \in \textsc{SatFormula}.\widehat{\phi}_C \Leftrightarrow \widehat{\phi'}_C \wedge \widehat{\phi}_A \Rightarrow \widehat{\phi'}_A \iff \widehat{\phi} \mathrel{\widetilde{\Rightarrow}} \widehat{\phi'}$. By definition of γ (Definition 2.3), $\widehat{\phi} \in \gamma(\widehat{\phi}) \iff \widehat{\phi}_C \Leftrightarrow \widehat{\phi'}_C \wedge \widehat{\phi}_A \Rightarrow \widehat{\phi'}_A$. Therefore, $\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \widehat{\phi}.\widehat{\phi} \in \gamma(\widetilde{\phi}) \wedge \widehat{\phi} \Rightarrow \mathit{static}(\widetilde{\phi'})$.
- Case $\widetilde{\phi'} = \widehat{\phi'} * ?_C$: By IMPLGRAD_C (Definition 4.1), $\exists \widehat{\phi} \in \textsc{SatFormula}.\widehat{\phi}_A \Leftrightarrow \widehat{\phi'}_A \wedge \widehat{\phi}_C \Rightarrow \widehat{\phi'}_C \iff \widehat{\phi} \mathrel{\widetilde{\Rightarrow}} \widehat{\phi'}$. By definition of γ (Definition 2.3), $\widehat{\phi} \in \gamma(\widehat{\phi}) \iff \widehat{\phi}_A \Leftrightarrow \widehat{\phi'}_A \wedge \widehat{\phi}_C \Rightarrow \widehat{\phi'}_C$. Therefore, $\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \widehat{\phi}.\widehat{\phi} \in \gamma(\widetilde{\phi}) \wedge \widehat{\phi} \Rightarrow \mathit{static}(\widetilde{\phi'})$.

□

Lemma 4.3. *Consistent Implication Lifting.*

$$\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'} \iff \exists \phi \in \gamma(\widetilde{\phi}), \exists \phi' \in \gamma(\widetilde{\phi'}).\phi \Rightarrow \phi'$$

Proof. (Proof for Lemma 4.3)

- Case $\Rightarrow$: Let $\phi' = \mathit{static}(\widetilde{\phi'})$. By Lemma 4.2, $\exists \phi \in \gamma(\widetilde{\phi}).\phi \Rightarrow \mathit{static}(\widetilde{\phi'})$, where $\mathit{static}(\widetilde{\phi'}) = \phi'$.

- Case $\Leftarrow$: By definition of γ, in all 4 cases, $\forall \phi' \in \gamma(\widetilde{\phi'}).\phi' \Rightarrow \mathit{static}(\widetilde{\phi'})$. Since $\phi \Rightarrow \phi'$, we know $\phi \Rightarrow \mathit{static}(\widetilde{\phi'})$ by transitivity of implication. Therefore, the right hand side implies:

$$\exists \phi \in \gamma(\widetilde{\phi}).\phi \Rightarrow \mathit{static}(\widetilde{\phi'})$$

. By Lemma 4.2, $\widetilde{\phi} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'}$. Therefore, we can replace the inference rule by

□

5 Dynamic Semantics

Definition 5.1. *(Modified* $\widetilde{\text{sWLP}}^m$*) We redefine* $\widetilde{\text{sWLP}}^m$ *to adjust to the new definition of gradual formulas:*

$$\widetilde{sWLP}^m(\bar{s}, \widetilde{\phi}) = \begin{cases} \widehat{\phi'_n} \cdot \widetilde{\phi}_{n-1} \cdot ... \cdot \widetilde{\phi}_1 \cdot nil & \widetilde{\phi}_p \text{ and } \widetilde{\phi}_n \text{ precise} \\ ?_A * \widehat{\phi'_n} \cdot \widetilde{\phi}_{n-1} \cdot ... \cdot \widetilde{\phi}_1 \cdot nil & \widetilde{\phi}_p \text{ or } \widetilde{\phi}_n \text{ acc imprecise } \wedge \widetilde{\phi}_p \text{ and } \widetilde{\phi}_n \text{ classically precise} \\ \widehat{\phi'_n} *?_C \cdot \widetilde{\phi}_{n-1} \cdot ... \cdot \widetilde{\phi}_1 \cdot nil & \widetilde{\phi}_p \text{ and } \widetilde{\phi}_n \text{ acc precise } \wedge \widetilde{\phi}_p \text{ or } \widetilde{\phi}_n \text{ classically imprecise} \\ ?_A * \widehat{\phi'_n} *?_C \cdot \widetilde{\phi}_{n-1} \cdot ... \cdot \widetilde{\phi}_1 \cdot nil & \widetilde{\phi}_p \text{ or } \widetilde{\phi}_n \text{ acc imprecise } \wedge \widetilde{\phi}_p \text{ or } \widetilde{\phi}_n \text{ classically imprecise} \end{cases}$$

$$\text{where } \widetilde{\phi}_n \cdot \widetilde{\phi}_{n-1} \cdot ... \cdot \widetilde{\phi}_1 \cdot nil = \widetilde{\text{WLP}}(\bar{s}, \widetilde{\phi})$$

$$\widehat{\phi'_n} = \begin{cases} \min_{\Rightarrow}\{\widehat{\phi'_n} \mid \mathit{static}(\widetilde{\phi}_n) \Rightarrow \widehat{\phi'_n} * \widetilde{\phi'_p} \wedge \widehat{\phi'_n} * \widetilde{\phi}_p \in \textsc{SatFormula}\} & \text{if } \widetilde{\phi'_p} \text{ acc precise} \\ \min_{\Rightarrow}\{\widehat{\phi'_n} \mid \mathit{static}(\widetilde{\phi}_n) \Rightarrow \widehat{\phi'_n} \wedge \lfloor \widehat{\phi'_n} \rfloor = \emptyset\} & \text{otherwise} \end{cases}$$

$$\text{and } \widetilde{\phi'_p} = \mathit{mpre}(m)[z, x/\texttt{this}, \texttt{mparam(m)}]$$

Definition 5.2. *Naive dynanimc semantics.*
Let $\langle H, \langle \rho_n, A_n, s_n\rangle \cdot ... \cdot \langle \rho_1, A_1, s_1\rangle \cdot nil\rangle, \langle H, \langle \rho'_n, A'_n, s'_n\rangle \cdot ... \cdot \langle \rho'_1, A'_1, s'_1\rangle \cdot nil\rangle \in \textsc{State}$. *If*

$$\langle H, \langle \rho_n, A_n, s_n\rangle \cdot ... \cdot \langle \rho_1, A_1, s_1\rangle \cdot nil\rangle \longrightarrow \langle H, \langle \rho'_n, A'_n, s'_n\rangle \cdot ... \cdot \langle \rho'_1, A'_1, s'_1\rangle \cdot nil\rangle$$

holds, and

$$\bar{\phi} = \widetilde{\text{sWLP}}(s_n \cdot ... \cdot s_1, \mathit{true})$$

then

$$\langle H, \langle \rho_n, A_n, s_n\rangle \cdot ... \cdot \langle \rho_1, A_1, s_1\rangle\rangle \widetilde{\longrightarrow} \begin{cases} \langle H, \langle \rho'_n, A'_n, s'_n\rangle \cdot ... \cdot \langle \rho'_1, A'_1, s'_1\rangle\rangle & (\forall i \le n\ \langle H, \rho'_i, A'_i\rangle \mathrel{\widetilde{\models}} \bar{\phi}_i) \\ \mathit{error} & (\mathit{otherwise}) \end{cases}$$

Note: in the naive dynamic semantics, we modified the following cases of $\longrightarrow$ in SVL_{IDF} to gradual formulas:

$$\mathit{method}(m) = T_r \quad m(Tx')\ \texttt{requires}\ \widetilde{\phi}_p\ \texttt{ensures}\ \widetilde{\phi}_q\{r\} \quad H, \rho \vdash z \Downarrow o \quad H, \rho \vdash x \Downarrow v$$

$$A' = \begin{cases} \lfloor \widetilde{\phi}_p \rfloor_{H,\rho'} & \text{if } \widetilde{\phi}_p \text{ acc precise} \\ A & \text{otherwise} \end{cases}$$

$$\frac{\rho' = [\text{this} \mapsto o, x' \mapsto v] \qquad \langle H, \rho', A' \rangle \mathrel{\widetilde{\models}} \widetilde{\phi}_p}{\langle H, \langle \rho, A, (y = z.m(x); s) \rangle \cdot ... \rangle \longrightarrow \langle H, \langle \rho', A', r \rangle \cdot \langle \rho, A \setminus A', (y = z.m(x); s) \rangle \cdot ... \rangle} \text{ SsCall}$$

$$\frac{\langle H, \rho', A' \rangle \mathrel{\widetilde{\models}} \widetilde{\phi}_q \qquad \rho'' = \rho[y \mapsto \rho'(result)]}{\langle H, \langle \rho', A', \texttt{skip} \rangle \cdot \langle \rho, A \setminus A', (y = z.m(x); s_n) \rangle \cdot S \rangle \widetilde{\longrightarrow} \langle H, \langle \rho'', A \cup A', s_n) \rangle \cdot S \rangle} \text{ SsCallFinish}$$

Definition 5.3. *Dynamic semantics with residual checks.*

$$\frac{\langle H, \langle \rho_n, A_n, (s; s_n) \rangle \cdot ... \rangle \longrightarrow \langle H, \langle \rho'_n, A'_n, s_n \rangle \cdot ... \rangle}{\langle H, \langle \rho_n, A_n, (s; s_n) \rangle \cdot ... \rangle \widetilde{\longrightarrow} \langle H, \langle \rho'_n, A'_n, s_n \rangle \cdot ... \rangle} \text{ SsLocal}$$

$$method(m) = T_r \quad m(Tx') \texttt{ requires } \widetilde{\phi}_p \texttt{ ensures } \widetilde{\phi}_q\{r\} \quad H, \rho \vdash z \Downarrow o \quad H, \rho \vdash x \Downarrow v$$
$$\rho' = [\texttt{this} \mapsto o, x' \mapsto v] \quad A' = \begin{cases} \lfloor \widetilde{\phi}_p \rfloor_{H,\rho'} & \text{if } \widetilde{\phi}_p \text{ acc precise} \\ A & \text{otherwise} \end{cases}$$

$$\frac{\langle H, \rho', A' \rangle \mathrel{\widetilde{\models}} \widetilde{\phi}_p \qquad \langle H, \rho', A' \rangle \mathrel{\widetilde{\models}} \widetilde{\text{diff}}(\widetilde{\text{WLP}}(r, \widetilde{\phi}_q), \widetilde{\phi}_p)}{\langle H, \langle \rho, A, (y = z.m(x); s) \rangle \cdot ... \rangle \widetilde{\longrightarrow} \langle H, \langle \rho', A', r \rangle \cdot \langle \rho, A \setminus A', (y = z.m(x); s) \rangle \cdot ... \rangle} \text{ SsCall}$$

$$mpost(m) = \widetilde{\phi}_q \quad \rho'' = \rho[y \mapsto \rho'(result)] \quad \widetilde{\phi}'_q = \widetilde{\phi}_q[z, x, y/this, old(mparam(m), result)]$$
$$\widetilde{\phi}'_p = mpre(m)[z, x/\texttt{this,mparam(m)}] \quad \widetilde{\phi} = \widetilde{\text{sWLP}}_n(s_n \cdot ..., true)$$

$$\frac{\langle H, \rho'', A \cup A' \rangle \mathrel{\widetilde{\models}} \widetilde{\text{diff}}(\widetilde{\phi}, \widetilde{\text{SP}}(y := z.m(x), \widetilde{\phi}))}{\langle H, \langle \rho', A', \texttt{skip} \rangle \cdot \langle \rho, A \setminus A', (y = z.m(x); s_n) \rangle \cdot S \rangle \widetilde{\longrightarrow} \langle H, \langle \rho'', A \cup A', s_n) \rangle \cdot S \rangle} \text{ SsCallFinish}$$

Definition 5.4. *Strongest postconditions.*
Let $\text{SP} : \text{Stmt} \times \text{Formula} \to \text{Formula}$ *be defined as:*

$$\text{SP}(s, \phi) = \min_{\Rightarrow}\{\phi' \in \text{Formula} \mid \phi \Rightarrow \text{WLP}(s, \phi')\}$$

Let $\widetilde{\text{SP}} : \text{Stmt} \times \widetilde{\text{Formula}} \to \widetilde{\text{Formula}}$ *be defined as the Consistent Function Lifting of* SP*:*

$$\widetilde{\text{SP}}(s, \widetilde{\phi}) = \alpha(\{\text{SP}(s, \phi) \mid \phi \in \gamma(\widetilde{\phi})\})$$

Definition 5.5. *Reducing formulas.*
Let $\text{diff} : \text{Formula} \times \text{Formula} \to \text{Formula}$ *be defined as:*

$$\text{diff}(\phi_j, \phi_k) = \max_{\Rightarrow}\{\phi \in \text{Formula} \mid (\phi * \phi_k \Rightarrow \phi_j) \wedge (\phi * \phi_k \in \text{SatFormula})\}$$

(Note:** $\phi * \phi_k$ **is not well ordered, but it should not affect evaluation of formula or formula implication.)
Let $\widetilde{diff} : \widetilde{\text{Formula}} \times \widetilde{\text{Formula}} \to \widetilde{\text{Formula}}$ *be defined as:*

$$\widetilde{\text{diff}}(\widetilde{\phi}_j, \widetilde{\phi}_k) = \begin{cases} \text{diff}(\widetilde{\phi}_j, \text{static}(\widetilde{\phi}_k)) & (\widetilde{\phi}_j \ \textit{precise}) \\ ?_A * \text{diff}(\phi_j, \text{static}(\widetilde{\phi}_k)) & (\widetilde{\phi}_j = ?_A * \phi_j) \\ \text{diff}(\phi_a, \text{static}(\widetilde{\phi}_j)) * ?_C & (\widetilde{\phi}_j = \phi_j * ?_C) \\ ?_A * \text{diff}(\phi_j, \text{static}(\widetilde{\phi}_k)) * ?_C & (\widetilde{\phi}_j = ?_A * \phi_j * ?_C) \end{cases}$$

Lemma 5.6. *Dynamic semantics with residual checks is equivalent to dynamic semantics with naive full checks.*

Proof. Proof for Lemma 5.6. We prove the lemma by cases on the state $\langle H, S\rangle$. We assume $\langle H, S\rangle$ is valid. We will show after one step of evaluation $\langle H, S\rangle \longrightarrow \langle H, S'\rangle$, the state $\langle H, S'\rangle$ satisfies residual checks if the state H, S' satisfies the residual checks. The other direction (residual checks satisfied if full checks satisfied) is trivial since residual check is a subset of the full check.

Case $\widetilde{\textbf{SsLocal}}$

In this case, $S = \langle \rho_n, A_n, s; s_n\rangle \cdot ...$, where s does not involve a method call. After one step of evaluation, assume $\langle H, S\rangle \longrightarrow \langle H, S'\rangle$, where $S' = \langle \rho'_n, A'_n, s_n\rangle \cdot ....$ The naive semantics proposes to perform the following check:

$$\langle H, \rho', A'_n\rangle \widetilde{\models} \widetilde{\phi}$$

where $\widetilde{\phi} = \widetilde{\text{sWLP}}_n(s_n \cdot ... \cdot s_1, true)$. By assumption, the state we leave must be valid:

$$\langle H, \rho_n, A_n\rangle \widetilde{\models} \widetilde{\text{sWLP}}_n(s; s_n \cdot ... \cdot s_1 \cdot nil, true) = \widetilde{\text{WLP}}(s, \widetilde{\phi})$$

By definition of $\widetilde{\text{SP}}$,

$$\langle H, \rho'_n, A'_n\rangle \widetilde{\models} \widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \widetilde{\phi}))$$

By definition of $\widetilde{SP}$, we also know that for arbitrary precise formula ϕ,

$$\widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \phi)) \Rightarrow \phi$$

Therefore, by setting $\phi = \text{static}(\widetilde{\phi})$,

$$\begin{aligned} \text{static}(\widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \widetilde{\phi}))) &= \widetilde{\text{SP}}(s, \text{static}(\widetilde{\text{WLP}}(s, \widetilde{\phi}))) \\ &= \widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \text{static}(\widetilde{\phi}))) \\ &\Rightarrow \text{static}(\widetilde{\phi}) \end{aligned}$$

Therefore, $\langle H, \rho', A'_n\rangle \widetilde{\models} \text{static}(\widetilde{\phi})$, and therefore $\langle H, \rho', A'_n\rangle \widetilde{\models} \widetilde{\phi}$ by definition of consistent formula lifting.

Case $\widetilde{\text{SsCall}}$

In this case, $S = \langle \rho, A, (y = z.m(x); s)\rangle \cdot ....$ Recalling the definitions in SsCall,

$$m(x) \text{ requires } \widetilde{\phi}_p \text{ ensures } \widetilde{\phi}_q\{r\}$$

Let $\bar{\phi} = \widetilde{\text{sWLP}}(r \cdot (y = z.m(x); s) \cdot ..., \text{true})$. Let $\widetilde{\phi} = \bar{\phi}_{|\bar{\phi}|}$ and $\widetilde{\phi}' = \bar{\phi}_{|\bar{\phi}|-1}$.
After one step of evaluation, $\langle H, S\rangle \widetilde{\longrightarrow} \langle H, S'\rangle$, where

$$S' = \langle \rho', A', r)\rangle \cdot \langle \rho, A \setminus A', y = z.m(x); s\rangle \cdot ...$$

The naive semantics proposes to perform the following 3 checks:

1. $\langle H, \rho', A'\rangle \widetilde{\models} \widetilde{\phi}_p$: according to $\widetilde{\text{SsCall}}$, this check is performed explicitly.

2. $\langle H, \rho', A'\rangle \widetilde{\models} \widetilde{\phi} = \widetilde{\text{WLP}}(r, \widetilde{\phi}_q)$: by definition of $\widetilde{\text{diff}}$, since we checked for $\langle H, \rho', A'\rangle \widetilde{\models} \widetilde{\phi}_p$ explicitly, the naive check is reduced to:

   $$\langle H, \rho', A'\rangle \widetilde{\models} \widetilde{\text{diff}}(\widetilde{\text{WLP}}(r, \widetilde{\phi}_q), \widetilde{\phi}_p)$$

   which is checked explicitly in the residual checks.

3. $\langle H, \rho, A \setminus A'\rangle \widetilde{\models} \widetilde{\phi}'$: let $\widetilde{\phi}_n = \widetilde{\text{sWLP}}_n((y = z.m(x); s) \cdot ..., true)$.

   - Case $\widetilde{\phi}'_p$ acc precise: by definition of $\widetilde{\text{sWLP}}^m$, $static(\widetilde{\phi}') = \widehat{\phi}'$, where

     $$\widehat{\phi}' = min_{\Rightarrow}\{\widehat{\phi}'' \mid static(\widetilde{\phi}_n) \Rightarrow \widehat{\phi}'' * \widetilde{\phi}'_p \wedge \widehat{\phi}'' * \widetilde{\phi}'_p \in \text{SatFormula}\}$$

     By the dynamic semantics, we can assume the state we left must passed the check of the naive semantics, therefore $\langle H, \rho, A\rangle \widetilde{\models} \widetilde{\phi}_n$. Since $\widetilde{\phi}_n \Rightarrow \widehat{\phi}'$, we know that $\langle H, \rho, A\rangle \widetilde{\models} \widehat{\phi}'$. We also know that $\lfloor \widehat{\phi}' \rfloor \cap \lfloor \widetilde{\phi}_p \rfloor = \emptyset$ since $\widehat{\phi}' * \widetilde{\phi}_p \in \text{SatFormula}$. Since $A' = \lfloor \widetilde{\phi}_p \rfloor$ by SsCall, we can conclude that $\langle H, \rho, A \setminus A'\rangle \widetilde{\models} \widehat{\phi}'$, since removing each field in A' must not appear in $\widehat{\phi}'$. Therefore,

     $$\langle H, \rho, A \setminus A'\rangle \widetilde{\models} static(\widetilde{\phi}')$$

     so we can conclude

     $$\langle H, \rho, A \setminus A'\rangle \widetilde{\models} \widetilde{\phi}'$$

   - Case $\widetilde{\phi}'_p$ acc imprecise:by definition of $\widetilde{\text{sWLP}}^m$, $\widetilde{\phi}' = \widehat{\phi}'$, where

     $$\widehat{\phi}' = min_{\Rightarrow}\{\widehat{\phi}'' \mid static(\widetilde{\phi}_n) \Rightarrow \widehat{\phi}'' \wedge \lfloor \widehat{\phi}'' \rfloor = \emptyset\}$$

     By the dynamic semantics, we can assume the state we left must passed the check of the naive semantics, therefore $\langle H, \rho, A\rangle \widetilde{\models} \widetilde{\phi}_n$. Since $\widetilde{\phi}_n \Rightarrow \widehat{\phi}'$, we know that $\langle H, \rho, A\rangle \widetilde{\models} \widehat{\phi}'$.

We also know that $A' = A$, and therefore $A \setminus A' = \emptyset$. Since $\lfloor \widehat{\phi'} \rfloor = \emptyset$, we can conclude that

$$\langle H, \rho, A \setminus A' \rangle \widetilde{\models} \widehat{\phi'}$$

since the formula has empty footprint, and $A \setminus A'$ is also empty. Since $\widetilde{\phi'} = \widehat{\phi'}$, we finally conclude that

$$\langle H, \rho, A \setminus A' \rangle \widetilde{\models} \widetilde{\phi'}$$

Case SsC$\widetilde{\text{allFinish}}$

In this case, $s = y := z.m(x)$ and $S = \langle \rho', A', \texttt{skip} \cdot \langle \rho, A, s; s_n \rangle \cdot ...$ The naive semantics proposes to perform the following check:

$$\langle H, \rho'', A \cup A' \rangle \widetilde{\models} \widetilde{\phi}$$

where $\widetilde{\phi} = \widetilde{\text{sWLP}}_n(s_n \cdot ... \cdot s_1, true)$. By assumption, the state before the method call must be valid:

$$\langle H, \rho'', A \cup A' \rangle \widetilde{\models} \widetilde{\text{sWLP}}_n(s; s_n \cdot ... \cdot s_1 \cdot nil, true) = \widetilde{\text{WLP}}(s, \widetilde{\phi})$$

By definition of $\widetilde{\text{SP}}$, since state $\langle H, \rho', A' \rangle$ is reached by execution of s,

$$\langle H, \rho', A' \rangle \widetilde{\models} \widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \widetilde{\phi}))$$

Therefore, the check proposed in the naive semantics is reduced to

$$\langle H, \rho', A' \rangle \widetilde{\models} \widetilde{\text{diff}}(\widetilde{\phi}, \widetilde{\text{SP}}(s, \widetilde{\text{WLP}}(s, \widetilde{\phi})))$$

□

6 Soundness

We now establish the soundness with full checks. Since in the previous section we proved the dynmaic semantics with residual checks is equivalent to full checks, we proceed the proof of soundness only with full checks, and therefore extensible to residual checks.

Definition 6.1. *We call the state* $\langle H, \langle \rho_n, A_n, s_n \rangle \cdot ... \cdot \langle \rho_1, A_1, s_1 \rangle \cdot nil \rangle \in$ State *valid if* $\langle H, \rho_i, A_i \rangle \models sWLP_i(s_n \cdot ... \cdot s_1 \cdot nil, true)$ *for all* $1 \leq i \leq n$.

Lemma 6.2. *(Progress)*
If $\langle H, S \rangle \in$ State *is a valid state, then for some* $\langle H', S' \rangle \in$ State, $\langle H, S \rangle \widetilde{\longrightarrow} \langle H', S' \rangle$ *or* $\langle H, S \rangle \widetilde{\longrightarrow}$ ***error***.

Proof. Proof for Progress (Lemma 6.2).
We are given a valid state $\langle H, S \rangle \in$ State. Soundness of SVL_{IDF} implies that $\longrightarrow$ will step. By definition of the naive dynamic semantics, $\widetilde{\longrightarrow}$ either steps as in SVL_{IDF} if the checks succeed, or go to the error state. □

Lemma 6.3. *(Preservation)*
If $\langle H, S\rangle$ is a valid state and for some $\langle H', S'\rangle \in$ State, $\langle H, S\rangle \widetilde{\longrightarrow} \langle H', S'\rangle$ then $\langle H', S'\rangle$ is a valid state.

Proof. Proof for Progress (Lemma 6.3).
We are given a valid state $\langle H, S\rangle \in$ State. Soundness of SVL_{IDF} guarantees that $\longrightarrow$ will step to another valid state.

- If the explicit check succeeds, By definition of the naive dynamic semantics, $\widetilde{\longrightarrow}$ either steps as in SVL_{IDF} if the checks succeed, or go to the error state.
- If the explicit check fails, then the program steps to an error state, which trivially satisfies preservation.

□

7 Gradual guarantee

The following lemma is trivial by the definition of $\widetilde{\text{WLP}}$.
Lemma 7.1. *Let $p \in$ Stmt $\widetilde{\phi}, \widetilde{\phi'} \in \widetilde{\text{Formula}}$ such that $\widetilde{\phi} \sqsubseteq \widetilde{\phi'}$, then $\widetilde{\text{WLP}}(s, \widetilde{\phi}) \sqsubseteq \widetilde{\text{WLP}}(s, \widetilde{\phi'})$.*
Lemma 7.2. *Let $\widetilde{\phi}, \widetilde{\phi'} \in \widetilde{\text{Formula}}$ such that $\widetilde{\phi} \sqsubseteq \widetilde{\phi'}$. Then, $static(\widetilde{\phi}) \Rightarrow static(\widetilde{\phi'})$.*

Proof. Since $\widetilde{\phi} \sqsubseteq \widetilde{\phi'}$, we know

$$\gamma(\widetilde{\phi}) \subseteq \gamma(\widetilde{\phi'})$$

Since $\forall \widehat{\phi'} \in \gamma(\widetilde{\phi'})$, $\widehat{\phi'} \Rightarrow static(\widetilde{\phi'})$ by the definition of γ, therefore, $\forall \widehat{\phi} \in \gamma(\widetilde{\phi})$, $\widehat{\phi'} \Rightarrow static(\widetilde{\phi'})$. Since static$(()\widetilde{\phi}) \in \gamma(\widetilde{\phi})$, we can conclude that $static(\widetilde{\phi}) \Rightarrow static(\widetilde{\phi'})$.

□

Lemma 7.3. *Let $\widetilde{\phi}_1, \widetilde{\phi}_2, \widetilde{\phi'_1}, \widetilde{\phi'_2} \in \widetilde{\text{Formula}}$ satisfy: $\widetilde{\phi}_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2$, $\widetilde{\phi}_1 \sqsubseteq \widetilde{\phi'_1}$ and $\widetilde{\phi}_2 \sqsubseteq \widetilde{\phi'_2}$. Then, $\widetilde{\phi'_1} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'_2}$.*

Proof. (Proof for Lemma 7.3) By definition, $\gamma(\widetilde{\phi'_1}) \subseteq \gamma(\widetilde{\phi}_1)$ and $\gamma(\widetilde{\phi'_2}) \subseteq \gamma(\widetilde{\phi}_2)$. By Lemma 4.2, since $\widetilde{\phi}_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2$, $\exists \widehat{\phi}_1 \in \gamma(\widetilde{\phi}_1).\widehat{\phi}_1 \Rightarrow static(\widetilde{\phi}_2)$. Therefore, $\exists \widehat{\phi} \in \gamma(\widetilde{\phi'_1}).\widehat{\phi} \Rightarrow static(\widetilde{\phi}_2)$. By Lemma 7.2, since $\widetilde{\phi}_2 \sqsubseteq \widetilde{\phi'_2}$, $static(\widetilde{\phi}_2) \Rightarrow static(\widetilde{\phi'_2})$. Therefore,

$$\exists \widehat{\phi} \in \gamma(\widetilde{\phi'_1}).\widehat{\phi} \Rightarrow static(\widetilde{\phi'_2})$$

which is the definition of $\widetilde{\phi'_1} \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi'_2}$. □

Lemma 7.4. *(Static Gradual Guarantee) Let $p_1, p_2 \in$ Program such that $p_1 \sqsubseteq p_2$, then if p_1 is valid, then p_2 is valid.*

Proof. (Proof for Lemma 7.4)
Validity of a program relies on validity of each individual method. Consider the following methods

$$method(m) = T_r \quad m(T\ x')\ \texttt{requires}\ \widetilde{\phi}_p\ \texttt{ensures}\ \widetilde{\phi}_q\{r\}$$

and

$$method(m') = T_r \quad m(T\ x')\ \texttt{requires}\ \widetilde{\phi}'_p\ \texttt{ensures}\ \widetilde{\phi}'_q\{r\}$$

where $\widetilde{\phi}_p \sqsubseteq \widetilde{\phi}'_p$ and $\widetilde{\phi}_q \sqsubseteq \widetilde{\phi}'_q$. We assume m is a valid. By Lemma 7.1, $\widetilde{\text{WLP}}(r, \widetilde{\phi}_q) \sqsubseteq \widetilde{\text{WLP}}(r, \widetilde{\phi}'_q)$. Since m is valid, $\widetilde{\phi}_p \mathrel{\widetilde{\Rightarrow}} \widetilde{\text{WLP}}(r, \widetilde{\phi}_q)$. By Lemma 7.3, we can conclude $\widetilde{\phi}'_p \mathrel{\widetilde{\Rightarrow}} \widetilde{\text{WLP}}(r, \widetilde{\phi}'_q)$, therefore m' is valid. □

Definition 7.5. *(State Precision) Let $\pi_1, \pi_2 \in$ State. Then π_1 is more precise than π_2, written $\pi_1 \lesssim \pi_2$, if and only if all of the following applies:*

1. *π_1 and π_2 have identical heap and stacks of size n.*
2. *The stack of variable environments and stack of statements is identical.*
3. *Let $A^1_{1\ldots n}$ and $A^2_{1\ldots n}$ be the stack of footprints of π_1 and π_2, respectively. Then, the following holds for $1 \le m \le n$:*

$$\bigcup_{i=m}^{n} A^1_i \subseteq \bigcup_{i=m}^{n} A^2_i$$

Lemma 7.6. *Let $\bar{s} \in$ Stack, $\widetilde{\phi}, \widetilde{\phi}' \in$ Formula such that $\widetilde{\phi} \sqsubseteq \widetilde{\phi}'$, then $\forall 1 \le i \le n, \widetilde{\text{sWLP}}_i(s, \widetilde{\phi}) \sqsubseteq \widetilde{\text{sWLP}}_i(s, \widetilde{\phi}')$.*
Lemma 7.7. *(Dyanmic Gradual Guarantee) Let $p_1, p_2 \in$ Program such that $p1 \sqsubseteq p2$, and $pi \in$ State such that $\pi_1 \lesssim \pi_2$. If $\pi_1 \widetilde{\rightarrow}_{p_1} \pi'_1$, then $\pi_2 \widetilde{\rightarrow}_{p_2} \pi'_2$ for some $\pi'_2 \in$ State and $\pi'_1 \lesssim \pi'_2$.*

Proof. (Proof for Lemma 7.7)
We analyze the definition of $\widetilde{\rightarrow}$. Increasing imprecision of contracts will increase the imprecision of $\widetilde{\text{sWLP}}$ by Lemma 7.6 and hence increase the chances that that the non-error case applies. Hence, if $\pi_1 \widetilde{\Rightarrow}_{p_1} \pi'_1$, then $\pi_2 \widetilde{\Rightarrow}_{p_2} \pi'_2$. Now all we need to prove is $\pi'_1 \lesssim \pi'_2$. π'_1 and π'_2 have the same heap, stack size, variable environments and stack of statements, since the programs are identical, and difference in contracts doesn't affect the environmnet except the dynamic footprint, so the first 2 properties of $\lesssim$ are trivial. We finally prove the third property of $\lesssim$:

$$\bigcup_{i=m}^{n-1} A'^1_i \subseteq \bigcup_{i=m}^{n-1} A'^2_i$$

The dynamic footprints satisfies the requiresment. Let stack size before executing the next statement be n. We assume the state before executing the statement satisfies for $1 \le m \le n$:

$$\bigcup_{i=m}^{n} A^1_i \subseteq \bigcup_{i=m}^{n} A^2_i$$

We perform the analysis based on the naive dynamic sementics. Since the naive dynamic semantics changes the dynamic footprint in the same way as in SVL_{IDF}, we only need to analyze the 3 cases in SVL_{IDF} that changes the dynamic footprint: SsAlloc, SsCall and SsCallFinish

- Case SsAlloc: the only change to the dynamic footprint is to add $\langle o, f_i \rangle$ to both A_n^1 and A_n^2. Therefore, the subset relation still holds after the statement is executed.
- Case SsCall: recalling from SsCall, let $\widetilde{\phi}^1$ represent the precondition in p_1 and $\widetilde{\phi}^2$ represents the precondition in p_2. Consider the following cases:
  1. $\widetilde{\phi}^2$ acc precise. Then, since $\widetilde{\phi}^1 \sqsubseteq \widetilde{\phi}^2$, $\widetilde{\phi}^1$ must also be acc precise, which implies $\widetilde{\phi}^1_a = \widetilde{\phi}^2_a$. Therefore, $\lfloor \widetilde{\phi}^1 \rfloor = \lfloor \widetilde{\phi}^2 \rfloor$. Therefore, $A'^1_{n+1} = A'^2_{n+1}$, which implies $A'^1_{n+1} \subseteq A'^2_{n+1}$.
  2. $\widetilde{\phi}^2$ acc imprecise, then by SsCall, $A'^2_{n+1} = A^2_n$. Since we know that $A^1_n \subseteq A^2_n$ by plugging $m = n$ into assumption, and $A'^1_n \subseteq A^1_n$ by framing rule, we can conclude that $A'^1_{n+1} \subseteq A'^2_{n+1}$.

  Since we know
  $$\bigcup_{i=m}^{n} A_i^1 \subseteq \bigcup_{i=m}^{n} A_i^2$$
  and clearly A_n^1 is partitioned into A'^1_n and A'^1_{n+1} and A_n^2 is partitioned into A'^2_n and A'^2_{n+1}, so we can conclude that when $1 \leq m \leq n$
  $$\bigcup_{i=m}^{n+1} A_i'^1 = \bigcup_{i=m}^{n} A_i^1 \subseteq \bigcup_{i=m}^{n} A_i^2 = \bigcup_{i=m}^{n+1} A_i'^2$$
  When, $m = n + 1$,
  $$\bigcup_{i=m}^{n+1} A_i'^1 = A'^1_{n+1} \subseteq A'^2_{n+1} = \bigcup_{i=m}^{n+1} A_i'^2$$
  Therefore, $\forall 1 \leq m \leq n + 1$:
  $$\bigcup_{i=m}^{n+1} A_i'^1 \subseteq \bigcup_{i=m}^{n+1} A_i'^2$$
- Case SsCallFinish: By SsCallFinish, $A'^1_{n-1} = A^1_n \cup A^1_{n-1}$ and $A'^2_{n-1} = A^2_n \cup A^2_{n-1}$. Since we can assume that $\forall 1 \leq m \leq n$:
  $$\bigcup_{i=m}^{n} A_i^1 \subseteq \bigcup_{i=m}^{n} A_i^2$$
  Therefore, $\forall 1 \leq m \leq n - 1$:
  $$\bigcup_{i=m}^{n-1} A_i'^1 = \bigcup_{i=m}^{n} A_i^1 \subseteq \bigcup_{i=m}^{n} A_i^2 = \bigcup_{i=m}^{n-1} A_i'^2$$

□

8 Conclusion

References

[1] Johannes Bader, Jonathan Aldrich, and Eric Tanter. Gradual Program Verification. Proc. VMCAI, January 2018.

[2] Smans, J., Jacobs, B., Piessens F.: Implicit dynamic frames. FTfJP 2008, Technical Report ICIS-R08013, Radboud University, pp. 1-12 (2008)

[3] Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM 12(10), 576580 (1969)

[4] Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on. pp. 5574. IEEE (2002)

[5] Smans, J., Jacobs, B., Piessens, F.: Implicit dynamic frames: Combining dynamic frames and separation logic. In: European Conference on Object-Oriented Pro- gramming. pp. 148172. Springer (2009)

[6] Meyer, B.: Object-Oriented Software Construction. Prentice Hall (1988)