Verifier WLP Definitions

Jenna Wise, Johannes Bader, Jonathan Aldrich, Éric Tanter

February 15, 2019

1 Weakest liberal precondition calculus definitions over self-framed non-gradual formulas

$$\text{WLP}(\mathit{skip}, \widehat{\phi}) = \widehat{\phi}$$

$$\text{WLP}(s_1; s_2, \widehat{\phi}) = \text{WLP}(s_1, \text{WLP}(s_2, \widehat{\phi}))$$

$$\text{WLP}(T\ x, \widehat{\phi}) = \widehat{\phi}$$
$$\text{WLP}(x := e, \widehat{\phi}) = \max_{\Rightarrow} \left\{ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi}[e/x] \quad \wedge \quad \widehat{\phi}' \Rightarrow \text{acc}(e) \right\}$$

$$\text{WLP}(\mathit{if}\ (x \odot y)\ \{s_1\}\ \mathit{else}\ \{s_2\}, \widehat{\phi}) =$$

$$\text{WLP}(x.f := y, \widehat{\phi}) = \text{acc}(x.f) * \max_{\Rightarrow} \left\{ \widehat{\phi}' \mid \widehat{\phi}' * \text{acc}(x.f) * (x.f = y) \Rightarrow \widehat{\phi} \ \wedge\ \widehat{\phi}' * \text{acc}(x.f) \in \textsc{SatFormula} \right\}$$

$$\text{WLP}(x := \mathit{new}\ C, \widehat{\phi}) = \max_{\Rightarrow} \left\{ \widehat{\phi}' \mid \widehat{\phi}' * \overline{\text{acc}(x.f_i)} \Rightarrow \widehat{\phi} \right\}$$
$$\text{where fields}(C) = \overline{T_i\ f_i}$$

$$\text{WLP}(y := z.m(\overline{x}), \widehat{\phi}) = \mathit{undefined}$$

$$\text{WLP}(y := z.m_C(\overline{x}), \widehat{\phi}) = \max_{\Rightarrow} \Big\{ \widehat{\phi}' \mid y \notin \text{FV}(\widehat{\phi}') \quad \wedge \quad \widehat{\phi}' \Rightarrow (z \neq \mathit{null}) * \text{pre}(C, m) \left[z/\mathit{this}, \overline{x_i/\text{params}(C, m)_i} \right]$$
$$\wedge \quad \widehat{\phi}' * \text{post}(C, m) \left[z/\mathit{this}, \overline{x_i/\text{old}(\text{params}(C, m)_i)}, y/\mathit{result} \right] \Rightarrow \widehat{\phi} \Big\}$$

$$\text{WLP}(\mathit{assert}\ \phi_a, \widehat{\phi}) = \max_{\Rightarrow} \left\{ \widehat{\phi}' \mid \widehat{\phi}' \Rightarrow \widehat{\phi} \quad \wedge \quad \widehat{\phi}' \Rightarrow \phi_a \right\}$$

$$\text{WLP}(\mathit{release}\ \phi_a, \widehat{\phi}) =$$

$$\text{WLP}(\mathit{hold}\ \phi_a\ \{s\}, \widehat{\phi}) =$$

Note:

Dynamic method calls. Dynamic method calls are left undefined, because we are not verifying programs with dynamic dispatch at this time (all method calls should be static method calls). They are included in the grammar for future implementation.

If & Release & hold. Definitions coming soon.

Predicates in the logic. Although the grammar allows for abstract predicate families, we do not support them yet. Therefore, we assume formulas look like:

$$\phi ::= \text{true} \mid e \odot e \mid acc(e.f) \mid \phi * \phi$$

2 Algorithmic WLP calculus definitions over self-framed non-gradual formulas

Note:

It may be helpful to check that the $\text{WLP}(s, \widehat{\phi})$ is well-formed and/or self-framed for some of the more complicated rules, which may be buggy in implementation.

Note:

We do not always calculate the correct WLP due to a limitation in the expressiveness of our specification language. Concrete formulas which do not contain enough information to determine whether two or more objects alias, wrongly assume that those two or more objects do not alias. This is because our current specification language is missing a logical OR or a conditional construct, which would allow us to write self-framed concrete formulas that capture the unknown aliasing between two or more objects. A conditional construct will be added and the definitions below adjusted in the near future.

$\text{WLP}(skip, \widehat{\phi}) = \widehat{\phi}$

$\text{WLP}(s_1; s_2, \widehat{\phi}) = \text{WLP}(s_1, \text{WLP}(s_2, \widehat{\phi}))$

$$\text{WLP}(T\ x, \widehat{\phi}) = \begin{cases} \widehat{\phi} & if\ x \notin \text{FV}(\widehat{\phi}) \\ undefined & otherwise \end{cases}$$

$$\text{WLP}(x := e, \widehat{\phi}) = \begin{cases} \widehat{\phi}[e/x] & if\ \widehat{\phi}[e/x] \Rightarrow \widehat{\phi}_{acc(e)} \\ \widehat{\phi}_{acc} * |\widehat{\phi}[e/x]| & otherwise \end{cases}$$

where e_i is an object dereferenced in e (ex. $y \neq null$ and $y.f \neq null$ if $e = y.f.f$), $\mid \phi \mid$ means the formula ϕ without accessibility predicates, $\widehat{\phi}_{acc(e)}$ is the formula of conjoined accessibility predicates necessary to frame $e = e$, and $\widehat{\phi}_{acc}$ is the formula of conjoined accessibility predicates necessary to frame $|\widehat{\phi}[e/x]|$ and $e = e$ ($\widehat{\phi}_{acc}$ can be computed in a similar fashion as in the assert rule just with ϕ_a replaced by $e = e$ and $\widehat{\phi}$ replaced by $\widehat{\phi}[e/x]$).

Check that $\text{WLP}(x := e, \widehat{\phi}) * x = e \Rightarrow \widehat{\phi}$ and that $\text{WLP}(x := e, \widehat{\phi})$ is satisfiable.

Important cases to consider:

$\widehat{\phi} = acc(y.f) * acc(y.f.f) * x = y.f.f,\ e = y.f.f$

$\widehat{\phi} = acc(y.f) * y.f = z * x > 0,\ e = y.f.f$

$\text{WLP}(if\ (x \odot y)\ \{s_1\}\ else\ \{s_2\}, \widehat{\phi}) =$

$$\text{WLP}(x.f := y, \widehat{\phi}) = \begin{cases} \widehat{\phi}[y/x.f] * x & if\ \widehat{\phi}[y/x.f] \Rightarrow acc(x.f) \\ acc(x.f) * \widehat{\phi}[y/x.f] & otherwise \end{cases}$$

Check that $\text{WLP}(x.f := y, \widehat{\phi}) * x.f = y \Rightarrow \widehat{\phi}$ and that $\text{WLP}(x.f := y, \widehat{\phi})$ is satisfiable.

Important cases to consider:

$\widehat{\phi} = acc(x.f) * x.f = p * x.f = q * a = b$

$\widehat{\phi} = acc(x.f) * acc(x.f.f) * x = y$

$$\text{WLP}(x := new\ C, \widehat{\phi}) = \begin{cases} \widehat{\phi} \div x & if\ (\widehat{\phi} \div x) * \overline{x \neq e_i} * \overline{acc(x.f_i)} \Rightarrow \widehat{\phi} \\ undefined & otherwise \end{cases}$$

where $\text{fields}(C) = \overline{T_i\ f_i}$, $\widehat{\phi} \div x$ means to transitively expand (in-)equalities ($\odot$) and then remove conjunctive terms containing x, and $\overline{x \neq e_i}$ are conjunctive terms in $\widehat{\phi}$.

Check $\text{WLP}(x := new\ C, \widehat{\phi})$ is satisfiable.

Important cases to consider:

$\widehat{\phi} = x \neq null * acc(x.f)$

$\widehat{\phi} = x \neq null * acc(x.f) * x.f = 1 * x.f = y$ — should fail, bad postcondition

$\widehat{\phi} = x \neq null * acc(x.f) * x = y * x = z$ — should fail, bad postcondition

$\widehat{\phi} = x \neq null * acc(x.f) * x = y * y = z$ — should fail, bad postcondition

$\widehat{\phi} = x \neq null * acc(x.f) * x \neq y * y = z$

$\widehat{\phi} = x \neq null * acc(x.f) * acc(x.f.f) * x.f.f \neq y$ — should fail, bad postcondition

$\widehat{\phi} = x \neq null * acc(y.f) * x = y$ — should fail, bad postcondition

$\widehat{\phi} = x \neq null * acc(x.f) * y > x.f * x.f > z * r \geq x.f * x.f \geq s$ — should fail, bad postcondition

Note:

$x := new\ C$ creates a fresh object and assigns it to x without setting default values to the object's fields; therefore, postconditions cannot say anything about the value of x other than it does not equal other values (no aliasing with x) and they cannot say anything about the values of the fields of x.

$\text{WLP}(y := z.m(\overline{x}), \widehat{\phi}) = undefined$

$$\text{WLP}(y := z.m_C(\overline{x}), \widehat{\phi}) = \widehat{\phi}\ \div\ y\ \overline{\div\ (e.f)_i} * z \neq null * \text{pre}(C, m)\ \left[z/this, \overline{x_i/\text{params}(C, m)_i}\right]$$

where $\phi \div x$ is defined differently than for the allocation rule above. $\phi \div x$ means to transitively expand (in-)equalities and comparisons and then remove conjunctive terms containing x, but removal of conjuctive terms containing x as an object dereference should replace x with an alias, if an alias exists. If an alias does not exist, then the term is removed as usual.

$(e.f)_i$ and $\phi\ \overline{\div\ (e.f)_i}$ are defined as follows:

$(e.f)_i$ is a field access used in the substituted postcondition of m. It comes from a list of field accesses used in the substituted postcondition of m that is determined in the following way:

1. Determine the list of field accesses occurring in $\text{post}(C, m)\left[z/this, \overline{x_i/\text{old}(\text{params}(C, m)_i)}, y/result\right]$

   - This list should be determined by starting with the ones in the accessibility predicates and then adding their duplicates due to aliasing to the list. If the static part of $\text{post}(C, m)\left[z/this, \overline{x_i/\text{old}(\text{params}(C, m)_i)}, y/result\right]$ is not self-framed, as will be the case in the WLP definition of method calls over gradual formulas, then start with the ones in the accessibility predicates and add the ones which occur in the non-accessibility part of $\text{post}(C, m)\left[z/this, \overline{x_i/\text{old}(\text{params}(C, m)_i)}, y/result\right]$ that are not already in the list. Finally, their duplicates due to aliasing should be added to the list.

$\phi \overline{\div (e.f)_i}$ removes dependencies on the field accesses in the removal list computed above in the following way:

1. Transitively expand (in-)equalities and comparisons in ϕ, so ϕ is now modified in this way
2. Determine the list of aliases to each variable or field access dereferenced to a final field value in ϕ using ϕ
3. Use the list of aliases to each variable or field access dereferenced to a final field value in ϕ to replace any variable or field access in ϕ that aliases to a top-level variable being dereferenced in a field access (ie. x in $x.f$ and $x.f.g$) in the list of removal that the $(e.f)_i$s come from. When substituting in multi-level field accesses (ie. $x.f.g.h$), the top-level object dereferences take precedence (ie. x should be substituted if possible; if not possible for x, then $x.f$ should be next and so forth).
4. Remove conjunctive terms containing the $(e.f)_i$s from ϕ

   - Removal for all conjunctive terms is straightforward (remove the term containing it) except for conjunctive terms which contain the $(e.f)_i$s as an object being dereferenced. In this case, each $(e.f)_i$ should be substituted with an available alias in ϕ; if no alias exists, then remove the term completely. The available alias should not be used if it exists in the list for removal that the $(e.f)_i$s come from or if the substitution of the alias results in a field access which contains or may be itself a field access that exists in the list for removal that the $(e.f)_i$s come from.
   - The order in which the $(e.f)_i$s are removed from ϕ should be from the smallest to largest in length of dereferencing, ie. $x.f$ is removed before $x.f.h$ which is removed before $x.f.h.g$ and so forth; $(e.f)_i$s with lengths of two are removed before $(e.f)_i$s with lengths of three and so forth.

Also, check that
$\text{WLP}(y := z.m_C(\overline{x}), \widehat{\phi}) \overline{\div (e.f)_i} \div y * \text{post}(C, m)\left[z/this, \overline{x_i/\text{old}(\text{params}(C, m)_i)}, y/result\right]$
$\Rightarrow \widehat{\phi}$,
and
$\text{WLP}(y := z.m_C(\overline{x}), \widehat{\phi})$ is satisfiable.
where $\phi \div x$ is defined as above, $(e.f)_i$ is defined as a field access used in the substituted precondition of m and coming from a list of field accesses used in the substituted precondition of m computed

in the same way as above except for replacing $\text{post}(C,m)\left[z/this, \overline{x_i/\text{old}(\text{params}(C,m)_i)}, y/result\right]$ with $\text{pre}(C,m)\left[z/this, \overline{x_i/\text{params}(C,m)_i}\right]$, and $\phi \overline{\div\ (e.f)_i}$ is defined as below:

1. Transitively expand (in-)equalities and comparisons in ϕ, so ϕ is now modified in this way
2. Remove conjunctive terms containing the $(e.f)_i$s as a whole from ϕ. Do not remove conjunctive terms that contain an $(e.f)_i$ as an object being dereferenced

Important cases to consider:
TBD – have them just need to type them up
Note:
This rule assumes only $old(id)$ can show up in the postcondition of a method. $old(.)$ of anything else is not allowed, ie. $old(e.f)$ (for now).

$\text{WLP}(assert\ \phi_a, \widehat{\phi}) = \widehat{\phi}_{acc} * \mid \phi_a \mid * \mid \widehat{\phi} \mid$
where $\mid \phi \mid$ means the formula ϕ without accessibility predicates and $\widehat{\phi}_{acc}$ is the self-framed formula which contains the accessibility predicates that frame $\mid \phi_a \mid * \mid \widehat{\phi} \mid$ and, in general, the accessibility predicates in ϕ_a and $\widehat{\phi}$ that are not duplicated (even with respect to aliasing).
Also, check $\text{WLP}(assert\ \phi_a, \widehat{\phi}) \Rightarrow \phi_a$, $\text{WLP}(assert\ \phi_a, \widehat{\phi}) \Rightarrow \widehat{\phi}$, and $\text{WLP}(assert\ \phi_a, \widehat{\phi})$ is satisfiable.
Advice on how to compute $\widehat{\phi}_{acc}$:

1. Start with a list of the accessibility predicates in ϕ_a and $\widehat{\phi}$
2. Add accessibility predicates to this list for self-framing ϕ_a, as some may be missing since ϕ_a is not self-framed
   (a) Duplicates (including those due to aliasing) are fine, because they will be removed later
   (b) But, not including obvious duplicates will improve performance
3. Add aliasing information from the actual conjoined accessibility predicates in ϕ_a and $\widehat{\phi}$ to the non-accessibility parts of ϕ_a and $\widehat{\phi}$ respectively. In other words, the accessibility predicates in ϕ_a ($\widehat{\phi}$) separated by the separating conjunction indicate that if the fields ultimately being accessed within two or more of them are the same, then the objects being dereferenced in those field accesses do not alias, so add this information to the non-accessibility part of ϕ_a ($\widehat{\phi}$) to not lose information
   (a) $acc(x.f) * acc(y.f) * x.f = 2$ implies $acc(x.f) * acc(y.f) * x.f = 2 * x \neq y$
   (b) $acc(x.f) * acc(x.f.g) * acc(y.g) * acc(y.g.f) * x.f = 2$ implies
   $acc(x.f) * acc(x.f.g) * acc(y.g) * acc(y.g.f) * x.f = 2 * x \neq y.g * y \neq x.f$
4. Determine the list of aliases and the list of non-aliases to each variable or field access dereferenced to a final field value in the current list of accessibility predicates using $\mid \phi_a \mid * \mid \widehat{\phi} \mid$
   (a) Keep in mind that if $x.f \neq y.f$ then $x \neq y$ and that $x.f \neq y.f$ can be determined in many different ways, including through $x.f = 3 * y.f = 4$, so the SMT solver should be helpful and necessary for this

5. Remove duplicate accessibility predicates, including accessibility predicates which are duplicate due to aliasing (the list of aliases and list of non-aliases for each variable and field access dereferenced to a final field value in an acc predicate should help)
   (a) If it is unknown whether two objects alias due to missing or conflicting information in ϕ_a and $\widehat{\phi}$, then assume (for now) that they do not alias
6. $\widehat{\phi}_{acc}$ is the list of non-duplicated accessibility predicates conjoined with the separating conjunction in an appropriate order (ie. $acc(x.f)$ comes before $acc(x.f.g)$)

Important cases to consider:
$| \phi_a | * | \widehat{\phi} | = x \neq null * x.f = 1 * x.f = y.f * y.f \neq p * y.f.f > 1$
$| \phi_a | * | \widehat{\phi} | = x = y * x = z * x.f = 8 * y.f > 4$
$| \phi_a | * | \widehat{\phi} | = x = y * y = z * z.f + 1 \leq 10 * \text{true}$
$| \phi_a | * | \widehat{\phi} | = x.f.f \neq y * x \neq p * p = q * x.f > y * y > z$
$| \phi_a | * | \widehat{\phi} | = y > x.f * x.f > z * r \geq x.f * x.f \geq s$
$\phi_a = acc(x.f) * y = 2$ and $\widehat{\phi} = acc(z.f) * z.f = 4$
$\phi_a = acc(x.f) * x.f = 4$ and $\widehat{\phi} = acc(z.f) * z.f = 4$
Assumptions:
We assume that objects are only referred to in formulas through variables or as field accesses, variables or field accesses which refer to objects are not used in binary operations ($\oplus$), and variables or field accesses which refer to objects are not used in comparison operators other than $\neq$ and $=$.

$\text{WLP}(release\ \phi_a, \widehat{\phi}) =$

$\text{WLP}(hold\ \phi_a\ \{s\}, \widehat{\phi}) =$

3 Algorithmic WLP calculus definitions over gradual formulas

$\widetilde{\text{WLP}}(s, \widehat{\phi}) = \text{WLP}(s, \widehat{\phi})$ where s is not a call statement

$\widetilde{\text{WLP}}(skip, ? * \phi) = ? * \phi$

$\widetilde{\text{WLP}}(s_1; s_2, ? * \phi) = \widetilde{\text{WLP}}(s_1, \widetilde{\text{WLP}}(s_2, ? * \phi))$

$\widetilde{\text{WLP}}(T\ x := e, ? * \phi) = ? * \text{WLP}(T\ x := e, \phi)$
Check that $\widetilde{\text{WLP}}(T\ x := e, ? * \phi) * x = e \mathrel{\widetilde{\Rightarrow}} ? * \phi$ and that $\widetilde{\text{WLP}}(T\ x := e, ? * \phi)$ is satisfiable.
Note: The use of WLP here is fine even though ϕ is not self-framed; it will still compute the correct static part for $\widetilde{\text{WLP}}(T\ x := e, ? * \phi)$.
Important cases to consider:
Similar to the ones for the non-gradual version; replacing $\widehat{\phi}$ with ϕ and appending ? to the front of ϕ on either side of the = sign.

$$\widetilde{\text{WLP}}(x.f := y, ? * \phi) = \begin{cases} ? * \phi[y/x.f] & if\ \phi[y/x.f] \Rightarrow acc(x.f) \\ ? * acc(x.f) * \phi[y/x.f] & otherwise \end{cases}$$

Check that $\widetilde{\text{WLP}}(x.f := y, ? * \phi) * x.f = y \mathrel{\widetilde{\Rightarrow}} ? * \phi$ and that $\widetilde{\text{WLP}}(x.f := y, ? * \phi)$ is satisfiable.

Important cases to consider:

$? * \phi = ? * acc(x.f) * acc(x.f.f) * x = y$

$$\widetilde{\text{WLP}}(x := new\ C, ? * \phi) = \begin{cases} ? * \phi \div x & if\ (\phi \div x) * \overline{x \neq e_i} * \overline{acc(x.f_i)} \Rightarrow \phi \\ undefined & otherwise \end{cases}$$

where fields$(C) = \overline{T_i\ f_i}$, $\phi \div x$ means to transitively expand (in-)equalities ($\odot$) and then remove conjunctive terms containing x, and $\overline{x \neq e_i}$ are conjunctive terms in ϕ.

Check that $\widetilde{\text{WLP}}(x := new\ C, ? * \phi) * x \neq null * \overline{x \neq e_i} * \overline{acc(x.f_i)} \mathrel{\widetilde{\Rightarrow}} ? * \phi$ and that $\widetilde{\text{WLP}}(x := new\ C, ? * \phi)$ is satisfiable.

Important cases to consider:

Similar to the ones for the non-gradual version; replacing $\widehat{\phi}$ with ϕ and appending ? to the front of ϕ on either side of the = sign.

$\widetilde{\text{WLP}}(y := z.m(\overline{x}), \widetilde{\phi}) = undefined$

$$\widetilde{\text{WLP}}(y := z.m_C(\overline{x}), \widetilde{\phi}) = \begin{cases} if\ \widetilde{\phi}, \text{ pre, post are all concrete formulas} \\ if\ \widetilde{\phi} \text{ concrete and pre or post gradual} \quad \text{— IN PROGRESS} \\ if\ \widetilde{\phi} \text{ is gradual and pre or post gradual} \end{cases}$$

$\widetilde{\text{WLP}}(assert\ \phi_a, ? * \phi) = ? * \text{WLP}(assert\ \phi_a, \phi)$

Check that $\widetilde{\text{WLP}}(assert\ \phi_a, ? * \phi) \mathrel{\widetilde{\Rightarrow}} ? * \phi$, $\widetilde{\text{WLP}}(assert\ \phi_a, ? * \phi) \mathrel{\widetilde{\Rightarrow}} \phi_a$, and that $\widetilde{\text{WLP}}(assert\ \phi_a, ? * \phi)$ is satisfiable.

Note:

Since ϕ is not self-framed, then the resulting WLP$(assert\ \phi_a, \phi)$ will not necessarily be self-framed. This is okay due to ? accounting for the lack of self-framing and since WLP$(assert\ \phi_a, \phi)$ does not lose any information from ϕ_a or ϕ (in fact, it gains information with respect to ϕ_a). So, allow this call to WLP$(assert\ \phi_a, \widehat{\phi})$ even though ϕ is not self-framed.

Important cases to consider:

Similar to the ones for the non-gradual version; replacing $\widehat{\phi}$ with ϕ.

Note: The static parts of gradual formulas do not need to be self-framed unless the gradual formula is completely precise (completely static). The imprecision can account for the framing.

4 Gradual formula implication and satisfiability

4.1 Gradual formula implication implementation advice

$$\frac{\widehat{\phi}_1 \Rightarrow static(\widetilde{\phi}_2)}{\widehat{\phi}_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2}\ \textsc{ImplStatic}$$

$$\frac{\widehat{\phi} \in \textsc{SatFormula} \qquad \widehat{\phi} \Rightarrow \phi_1 \qquad \widehat{\phi} \Rightarrow \mathit{static}(\widetilde{\phi}_2)}{? * \phi_1 \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}_2} \textsc{ImplGrad}$$

ImplStatic should be implemented directly for determining gradual formula implication. ImplGrad should be implemented by checking $\widehat{\phi}$ is satisfiable, is self-framed, implies ϕ_1, and implies $\mathit{static}(\widetilde{\phi}_2)$ and following this advice for determining $\widehat{\phi}$ (determining $\widehat{\phi}$ is similar to computing WLP($\mathit{assert}\ \phi_a, \widehat{\phi}$), so some of the detailed advice is left out since it is mentioned above):

1. Start with a list of the accessibility predicates in ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$
2. Add accessibility predicates to this list for self-framing ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$, as some may be missing since ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$ is/may not be self-framed
   (a) Duplicates (including those due to aliasing) are fine, because they will be removed later
   (b) But, not including obvious duplicates will improve performance
3. Add aliasing information from the actual conjoined accessibility predicates in ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$ to the non-accessibility parts of ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$ respectively. In other words, the accessibility predicates in ϕ_1 ($\mathit{static}(\widetilde{\phi}_2)$) separated by the separating conjunction indicate that if the fields ultimately being accessed within two or more of them are the same, then the objects being dereferenced in those field accesses do not alias, so add this information to the non-accessibility part of ϕ_1 ($\mathit{static}(\widetilde{\phi}_2)$) to not lose information
4. Determine the list of aliases and the list of non-aliases to each variable or field access dereferenced to a final field value in the current list of accessibility predicates using $\mid \phi_1 \mid * \mid \mathit{static}(\widetilde{\phi}_2) \mid$
5. Remove duplicate accessibility predicates, including accessibility predicates which are duplicate due to aliasing (the list of aliases and list of non-aliases for each variable and field access dereferenced to a final field value in an acc predicate should help)
   (a) If it is unknown whether two objects alias due to missing or conflicting information in ϕ_1 and $\mathit{static}(\widetilde{\phi}_2)$, then assume (for now) that they do not alias
6. Then $\widehat{\phi} = \widehat{\phi}_{acc} * \mid \phi_1 \mid * \mid \mathit{static}(\widetilde{\phi}_2) \mid$, where $\widehat{\phi}_{acc}$ is the list of non-duplicated accessibility predicates conjoined with the separating conjunction in an appropriate order (ie. $\mathit{acc}(x.f)$ comes before $\mathit{acc}(x.f.g)$)

4.2 Gradual formula satisfiability implementation advice

For $\widetilde{\phi}$, if $\widetilde{\phi} = \widehat{\phi}$, then check satisfiability as usually for fully-precise formulas. If $\widetilde{\phi} = ? * \phi$, then $? * \phi$ is satisfiable iff ϕ is satisfiable. Also, there is no need to modify ϕ to be self-framed before checking satisfiability. Any attempt to modify ϕ to be self-framed would not affect satisfiability, as this modification should ensure the resulting formula is satisfiable if ϕ is satisfiable by relying on existing information in ϕ. If ϕ was unsatisfiable, then this modification should not be made, since it relies on information in ϕ to be correct. But, if it was made it would not make the resulting formula satisfiable.

In general, it is recommended to check self-framing and satisfiability separately, ie. a formula can be satisfiable and not self-framed.