A Featherweight Java Correctness

We state again our main result.

Theorem 3.1: If $\vdash P \ \mid\ \sigma$ and $\varphi \models \sigma$, then $\exists \sigma'$ such that $\vdash \mathit{reduce}(P, \varphi) \ \mid\ \sigma'$.

Proof. Let $P = (\overline{R}\ e)$. We have the assumption 1) $\vdash P \ \mid\ \sigma$, which was derived from the assumptions 2) $\overline{R}$ OK in $P \ \mid\ \overline{\pi}$ and 3) $P, \emptyset \vdash e : T \ \mid\ \pi$, where $\sigma = \overline{\pi} \wedge \pi$. For each R_i in $\mathit{reduce}(P, \varphi)$, we have $\varphi \models [\mathit{Rid}(R)]$, so from (1), (2), and Lemma A.1 we have $\exists \pi_i'$ such that 4) $\mathit{reduceR}(R_i, \varphi)$ OK in $\mathit{reduce}(P, \varphi) \ \mid\ \pi_i'$. We have from (1), (3), and Lemma A.4 that $\exists \pi'$ such that 5) $\mathit{reduce}(P, \varphi), \emptyset \vdash e : T \ \mid\ \pi'$. Define $\sigma' = \overline{\pi}' \wedge \pi'$. From (4), (5), the definition $\mathit{reduce}(P, \varphi) = \mathit{reduceR}(\overline{R}, \varphi)\ e$, and the rule for program typing, we have $\vdash \mathit{reduce}(P, \varphi) \ \mid\ \sigma'$. □

Lemma A.1. *If* $\vdash P \ \mid\ \sigma$ *and* R *OK in* $P \ \mid\ \pi$ *and* $\varphi \models \sigma \wedge [\mathit{Rid}(R)] \wedge \pi$, *then* $\exists \pi'$ *such that* $\mathit{reduceR}(R, \varphi)$ *OK in* $\mathit{reduce}(P, \varphi) \ \mid\ \pi'$.

We define the notation $\mathit{Rid}(R)$ *as follows. If* $R = \text{class } C \text{ extends } D \text{ implements } I\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}$, *then* $\mathit{Rid}(R) = C$. *If* $R = \text{interface } I\ \{\overline{S}\}$, *then* $\mathit{Rid}(R) = I$.

Proof. We have two main cases.

In the first main case, if $R = \text{class } C \text{ extends } D \text{ implements } I\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}$, then we have the assumption R OK in $P \ \mid\ \pi$, which was derived from the assumptions 1) $\mathit{fields}(P, D) = \overline{U}\ \overline{g}$ and 2) $K = C(\overline{U}\ \overline{g}, \overline{T}\ \overline{f})\ \{\ \text{super}(\overline{g});\ \text{this}.\overline{f} = \overline{f};\ \}$ and 3) $P \vdash \overline{M}$ OK in $C \ \mid\ \overline{\pi}$ and 4) $P(I) = \text{interface } I\ \{\overline{S}\}$ and 5) $P \vdash \overline{S}$ OK in I for $C \ \mid\ \overline{\tau}$, and where $\pi = (([C] \Rightarrow ([D] \wedge [\overline{U}] \wedge [\overline{T}])) \wedge ([C \lhd I] \Rightarrow ([C] \wedge [I])) \wedge \overline{\pi} \wedge \overline{\tau})$. From the assumption that $\varphi \models [C] \wedge \pi$, we have 6) $\varphi \models [D]$. From (6), (1) and Lemma A.5, we have 7) $\mathit{fields}(\mathit{reduce}(P, \varphi), D) = \overline{U}\ \overline{g}$. For each M with name m in $\mathit{reduceM}(\overline{M}, \varphi)$, we have that σ as a conjunct contains $[C.m()!\mathit{code}] \Rightarrow [C.m()]$, so we have 8) $\varphi \models [C.m()]$. From the assumption that $\varphi \models [C]$ and (8) and Lemma A.2, we have that $\exists \pi'$ such that 9) $\mathit{reduce}(P, \varphi) \vdash \mathit{reduceM}(M, \varphi)$ OK in $C \ \mid\ \pi'$.

Now we have two subcases.

In the first subcase, $\varphi([C \lhd I]) = 1$. By definition we have $\mathit{reduceI}(C, I, \varphi) = I$. From the assumption that $\varphi \models \pi$ and (5), we have 10) $\varphi \models ([C] \wedge [I])$. From (10) we have $\mathit{reduce}(P, \varphi)$ is defined on I and that 11) $(\mathit{reduce}(P, \varphi))(I) = \text{interface } I\ \{\mathit{reduceS}(I, \overline{S}, \varphi)\}$. For any $T\ m(\overline{T}\ \overline{x})$ in $\mathit{reduceS}(I, S, \varphi)$ we have 12) $\varphi([I.m()]) = 1$. From (5), $\varphi([C \lhd I]) = 1$, (10), (12), and Lemma A.3, we have that $\exists \tau'$ such that 13) $\mathit{reduce}(P, \varphi) \vdash T\ m(\overline{T}\ \overline{x})$ OK in I for $C \ \mid\ \tau'$. From (7), (2), (9), (11), (13), and the rule for class typing, we have that $\exists \pi'$ such that $\mathit{reduce}(R, \varphi)$ OK in $\mathit{reduce}(P, \varphi) \ \mid\ \pi'$.

In the second subcase, $\varphi([C \lhd I]) = 0$. By definition we have $\mathit{reduceI}(C, I, \varphi) = \text{EmptyInterface}$. Also by definition, we have that 14) $(\mathit{reduce}(P, \varphi))(\text{EmptyInterface}) = \text{interface EmptyInterface}\ \{\ \}$. Notice that EmptyInterface contains no signatures, which wipes out the condition
$P \vdash \overline{S}$ OK in EmptyInterface for $C \ \mid\ \overline{\tau}$. So, from (7), (2), (9), (14), we have that $\exists \pi'$ such that $\mathit{reduce}(R, \varphi)$ OK in $\mathit{reduce}(P, \varphi) \ \mid\ \pi'$.

In the second main case, if $R = \text{interface } I\ \{\overline{S}\}$, then we have the assumption R OK in $P \ \mid\ \overline{\pi}$, which was derived from the assumption 15) $\overline{S}$ OK in $I \ \mid\ \overline{\pi}$. For each $T\ m(\overline{T}\ \overline{x})$ in $\mathit{reduceS}(I, \overline{S}, \varphi)$, we have 16) $\varphi([I.m()]) = 1$. From the rule for signature typing, we have that we can derive 17) $T\ m(\overline{T}\ \overline{x})$ OK in $I \ \mid\ \pi$. So, from (17) we have that $\exists \overline{\pi}'$ such that $\mathit{reduceR}(R, \varphi)$ OK in $\mathit{reduce}(P, \varphi) \ \mid\ \overline{\pi}'$. □

Lemma A.2. *Let* $M = T\ m(\overline{T}\ \overline{x})\ \{\ \text{return } e;\ \}$. *If* $\vdash P \ \mid\ \sigma$ *and* $P \vdash M$ *OK in* $C \ \mid\ \pi$ *and* $\varphi \models \sigma \wedge [C] \wedge [C.m()] \wedge \pi$ *then* $\exists \pi'$ *such that* $\mathit{reduce}(P, \varphi) \vdash \mathit{reduceM}(M, \varphi)$ *OK in* $C \ \mid\ \pi'$.

Proof. We have the assumption $P \vdash M$ OK in $C \ \mid\ \pi$, which was derived from the assumptions
2) $P(C) = \text{class } C \text{ extends } D \text{ implements } I\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}$ and 3) $\mathit{override}(P, m, D, \overline{T} \to T)$ and 4) $P, (\overline{x} : \overline{T}, \text{this} : C) \vdash e : U \ \mid\ \pi_1$ and 5) $P \vdash U \leq T \ \mid\ \pi_2$, where 6) $\pi = ([C.m()] \Rightarrow ([C] \wedge [T] \wedge [\overline{T}])) \wedge ([C.m()!\mathit{code}] \Rightarrow ([C.m()] \wedge \pi_1 \wedge \pi_2))$. From (6) and the assumption that $\varphi \models [C.m()] \wedge \pi$, we have 7) $\varphi \models [T] \wedge [\overline{T}]$. From (2), (3), $\varphi \models \sigma$, and Lemma A.7, we have 8) $\mathit{override}(\mathit{reduce}(P, \varphi), m, D, \overline{T} \to T)$.

Now we have two cases.

In the first case, 12) $\varphi \models [C.m().\mathit{code}]$. In this case, 13) $\mathit{reduceM}(M) = M$. From (12) and (6), we have 14) $\varphi \models \pi_1 \wedge \pi_2$. From (4), (7), the assumption that $\varphi \models [C]$, (14), and Lemma A.4, we have $\exists \pi_1'$ such that 15) $\mathit{reduce}(P, \varphi), (\overline{x} : \overline{T}, \text{this} : C) \vdash e : U \ \mid\ \pi_1'$ and 16) $\varphi \models [U]$. From (5), (16), (14), and Lemma A.9, we have $\exists \pi_2'$ such that 17) $\mathit{reduce}(P, \varphi) \vdash U \leq T \ \mid\ \pi_2'$. From (2), (8), (15), (17), and the rule for method typing, we derive $\mathit{reduce}(P, \varphi) \vdash \mathit{reduceM}(M)$ OK in $C \ \mid\ \pi'$, where $\pi' = ([C.m()] \Rightarrow ([C] \wedge [T] \wedge [\overline{T}])) \wedge ([C.m()!\mathit{code}] \Rightarrow ([C.m()] \wedge \pi_1' \wedge \pi_2'))$.

In the second case, 18) $\varphi([C.m().code]) = 0$. In this case, 19) $reduceM(M) = T\ m(\overline{T}\ \overline{x})$ { return this.$m(\overline{x})$; }. From the expression typing rules for variables and method call, we can derive 20) $P, (\overline{x} : \overline{T}, \text{this} : C) \vdash \text{this}.m(\overline{x}) : T \mid (1 \wedge ([C.m()] \vee mAny(P, m, D)) \wedge \overline{1} \wedge 1)$, and from the assumption that $\varphi \models [C.m()]$, we have that 21) $\varphi \models 1 \wedge ([C.m()] \vee mAny(P, m, D)) \wedge \overline{1} \wedge 1)$. From (20), (7), the assumption that $\varphi \models [C]$, (21), and Lemma A.4, we have $\exists \pi_1'$ such that 22) $reduce(P, \varphi), (\overline{x} : \overline{T}, \text{this} : C) \vdash \text{this}.m(\overline{x}) : T \mid \pi_1'$. From the first rule for subtyping, we have 23) $reduce(P, \varphi) \vdash T \leq T \mid 1$. From (2), (8), (22), (23), and the rule for method typing, we derive $reduce(P, \varphi) \vdash reduceM(M)$ OK in $C \mid \pi'$, where $\pi' = ([C.m()] \Rightarrow ([C] \wedge [T] \wedge [\overline{T}])) \wedge ([C.m()!code] \Rightarrow ([C.m()] \wedge \pi_1' \wedge 1))$. □

Lemma A.3. *If $\vdash P \mid \sigma$ and $P \vdash T\ m(\overline{T}\ \overline{x})$ OK in I for $C \mid \tau$ and $\varphi \models \sigma \wedge [I] \wedge [I.m()] \wedge [C] \wedge [C \lhd I] \wedge \tau$, then $\exists \tau' : reduce(P, \varphi) \vdash T\ m(\overline{T}\ \overline{x})$ OK in I for $C \mid \tau'$.*

Proof. From the rule for signature typing relative to a class, we have that 1) $\tau = ([C \lhd I] \wedge [I.m()]) \Rightarrow mAny(P, m, C)$, which was derived from 2) $mtype(P, m, C) = \overline{T} \rightarrow T$. From (1) and the assumption that $\varphi \models \sigma \wedge [I] \wedge [I.m()] \wedge [C] \wedge [C \lhd I] \wedge \tau$, we have that 3) $\varphi \models mAny(P, m, C)$. From (2), (3), $\varphi \models [C]$, and Lemma A.6, we have 4) $mtype(reduce(P, \varphi), m, C) = (\overline{T} \rightarrow T)$. From (4) and the rule for signature typing relative to a class, we can derive $reduce(P, \varphi) \vdash T\ m(\overline{T}\ \overline{x})$ OK in I for $C \mid mAny(P, m, C)$. □

Lemma A.4. *If $\vdash P \mid \sigma$ and $P, \Gamma \vdash e : T \mid \pi$ and $\varphi \models \sigma \wedge conjRan(\Gamma) \wedge \pi$, then $\exists \pi' : reduce(P, \varphi), \Gamma \vdash e : T \mid \pi'$ and $\varphi \models [T]$. We use the notation $conjRan(\Gamma)$ to denote $\bigwedge_{x \in domain(\Gamma)} [\Gamma(x)]$.*

Proof. We proceed by induction on e.

In the base case of a variable x, we have by assumption that (1) $P, \Gamma \vdash x : \Gamma(x) \mid 1$ and (2) $\varphi \models [\Gamma(x)]$. So $\Gamma(x)$ is defined and we can use the rule for variables to derive $reduce(P, \varphi), \Gamma \vdash x : \Gamma(x) \mid 1$, and we have from (2) that $\varphi \models [\Gamma(x)]$.

In the induction step, we have four cases.

In the case of a field access $e.f_i$, we have the assumption that $P, \Gamma \vdash e.f_i : T_i \mid \pi$, which was derived from the assumptions (1) $P, \Gamma \vdash e : C \mid \pi$ and (2) $fields(P, C) = \overline{T}\ \overline{f}$. From (1) and the induction hypothesis, we have π' such that (3) $reduce(P, \varphi), \Gamma \vdash e : C \mid \pi'$ and (4) $\varphi \models [C]$. From (2), (4), and Lemma A.5, we have (5) $fields(reduce(P, \varphi), C) = \overline{T}\ \overline{f}$. From (3), (5), and the rule for field access, we can derive $reduce(P, \varphi), \Gamma \vdash e.f_i : T_i \mid \pi'$. From the rule for class typing, we have that (6) σ as a conjunct contains $([C] \Rightarrow [T_i])$, so from (4), (6), and $\varphi \models \sigma$, we have $\varphi \models [T_i]$.

In the case of a method call $e.m(\overline{e})$, we have the assumption that $P, \Gamma \vdash e.m(\overline{e}) : U \mid [T] \wedge \pi_1 \wedge mAny(P, m, T) \wedge \overline{\pi} \wedge \pi_2$, which was derived from the assumptions (1) $P, \Gamma \vdash e : T \mid \pi_1$ and (2) $mtype(P, m, T) = \overline{U} \rightarrow U$, (3) $P, \Gamma \vdash \overline{e} : \overline{T} \mid \overline{\pi}$ and (4) $P \vdash \overline{T} \leq \overline{U} \mid \pi_2$. From (1) and the induction hypothesis, we have π_1' such that (5) $reduce(P, \varphi), \Gamma \vdash e : T \mid \pi_1'$ and (6) $\varphi \models [T]$. From (2), (6), the assumption that $\varphi \models \pi$, where π contains $mAny(P, m, T)$ as a conjunct, and Lemma A.6, we have (7) $mtype(reduce(P, \varphi), m, T) = \overline{U} \rightarrow U$ and (8) $\varphi \models [\overline{U}] \wedge [U]$. From (3) and the induction hypothesis, we have $\overline{\pi}'$ such that (9) $reduce(P, \varphi), \Gamma \vdash \overline{e} : \overline{T} \mid \overline{\pi}'$ and (10) $\varphi \models [\overline{T}]$. From (4), (10), and Lemma A.9, we have (11) $reduce(P, \varphi) \vdash \overline{T} \leq \overline{U} \mid \pi_2$. From (5), (7), (9), (11), and the rule for method call, we can derive $reduce(P, \varphi), \Gamma \vdash e.m(\overline{e}) : U \mid [T] \wedge \pi_1' \wedge mAny(reduce(P, \varphi), m, T) \wedge \overline{\pi}' \wedge \pi_2$. From (8), we have $\varphi \models [U]$.

In the case of an object creation new $C(\overline{e})$, we have the assumption $P, \Gamma \vdash \text{new } C(\overline{e}) : C \mid [C] \wedge \overline{\pi} \wedge \pi$, which was derived from the assumptions (1) $fields(P, C) = \overline{T}\ \overline{f}$ and (2) $P, \Gamma \vdash \overline{e} : \overline{U} \mid \overline{\pi}$ and (3) $P \vdash \overline{U} \leq \overline{T} \mid \pi$. From (1) and the assumption that $\varphi \models [C]$ and Lemma A.5, we have (4) $fields(reduce(P, \varphi), C) = \overline{T}\ \overline{f}$. From (2) and the induction hypothesis, we have $\overline{\pi}'$ such that (5) $reduce(P, \varphi), \Gamma \vdash \overline{e} : \overline{U} \mid \overline{\pi}'$ and (6) $\varphi \models [\overline{U}]$. From (3), (6), and Lemma A.9, we have (7) $reduce(P, \varphi) \vdash \overline{U} \leq \overline{T} \mid \pi$. From (4), (5), (7), and the rule for object creation, we have $reduce(P, \varphi), \Gamma \vdash \text{new } C(\overline{e}) : C \mid [C] \wedge \overline{\pi}' \wedge \pi$. From the assumption $\varphi \models [C] \wedge \overline{\pi} \wedge \pi$, we have $\varphi \models [C]$.

In the case of a cast $(T)\ e$, we have the assumption $P, \Gamma \vdash (T)\ e : T \mid [T] \wedge \pi$, which was derived from the assumption (1) $P, \Gamma \vdash e : U \mid \pi$. From (1) and the induction hypothesis, we have π' such that (2) $reduce(P, \varphi), \Gamma \vdash e : U \mid \pi'$ and (3) $\varphi \models [U]$. From (2) and the rule for cast, we have $reduce(P, \varphi), \Gamma \vdash (T)\ e : T \mid [T] \wedge \pi'$. From the assumption $\varphi \models [T] \wedge \pi$, we have $\varphi \models [T]$. □

Lemma A.5. *If $\vdash P \mid \sigma$ and $\varphi \models \sigma \wedge [C]$ and $fields(P, C) = \overline{T}\ \overline{f}$, then $fields(reduce(P, \varphi), C) = \overline{T}\ \overline{f}$.*

Proof. We proceed by induction on the derivation of $fields(P, C) = \overline{T}\ \overline{f}$.

If the last rule used in the derivation is the rule for Object, then we that $fields(P, \text{Object}) = \bullet$ and we can use the rule for Object to derive $fields(reduce(P, \varphi), \text{Object}) = \bullet$.

If the last rule used in the derivation is the rule for a class C, then $fields(P, C) = \overline{U}\ \overline{g}\ ,\ \overline{T}\ \overline{f}$, which was derived from (1) $P(C) =$ class C extends D implements I $\{\overline{T}\ \overline{f};\ K\ \overline{M}\}$ and (2) $fields(P, D) = \overline{U}\ \overline{g}$. From (1) and $\varphi \models [C]$ and the definition of $reduceR$, we have that $(reduce(P, \varphi))(C)$ is defined and that (3) the fields of $(reduce(P, \varphi))(C)$ are $\overline{T}\ \overline{f}$. From the rule for class typing applied to class C, we get that σ as a conjunct has the constraint $([C] \Rightarrow [D])$, so $\varphi \models (\sigma \wedge [C])$ gives that (4) $\varphi \models [D]$. From (2), (4), and the induction hypothesis, we have (5) $fields(reduce(P, \varphi), D) = \overline{U}\ \overline{g}$. From (3) and (5) and the rule for a class C, we conclude $fields(reduce(P, \varphi), C) = \overline{U}\ \overline{g}\ ,\ \overline{T}\ \overline{f}$. □

Lemma A.6. *If* $\vdash P \mid \sigma$ *and* $\varphi \models \sigma \wedge [T] \wedge mAny(P, m, T)$ *and* $mtype(P, m, T) = (\overline{U} \to U)$*, then* $mtype(reduce(P, \varphi), m, T) = (\overline{U} \to U)$ *and* $\varphi \models [\overline{U}] \wedge [U]$.

Proof. We proceed by induction on the derivation of $mtype(P, m, T) = (\overline{U} \to U)$.

Suppose the last rule used to derive $mtype(P, m, T) = (\overline{U} \to U)$ is the first rule for method type lookup. Thus, $T = C$ and we have 1) $P(C) =$ class C extends D implements I $\{\overline{T}\ \overline{f};\ K\ \overline{M}\}$ and 2) $U\ m(\overline{U}\ \overline{x})$ { return e; } $\in \overline{M}$. From (1), (2), and the first rule for method choice, we have 3) $mAny(P, M, C) = [C.m()] \vee mAny(P, m, D)$. Now we have two subcases.

In the first subcase, $\varphi([C.m()]) = 1$. We have that $(reduce(P, \varphi))(C)$ is defined that that is has a method m. So, $mtype(reduce(P, \varphi), m, C) = (\overline{U} \to U)$. From the rule for class typing, we have that σ as a conjunct contains (4) $[C] \Rightarrow ([U] \wedge [\overline{U}])$. From (4) and $\varphi \models [T]$, we have $\varphi \models [U] \wedge [\overline{U}]$.

In the second subcase, $\varphi([C.m()]) = 0$. From (3) and $\varphi([C.m()]) = 0$ we have (4) $mAny(P, M, C) = mAny(P, m, D)$ From (1) and the rule for class typing, we have that σ as a conjunct contains $[C] \Rightarrow [D]$. From $\varphi \models \sigma \wedge [T]$ we have 5) $\varphi \models [D]$. From (4), (5), and the induction hypothesis, we have 6) $mtype(reduce(P, \varphi), m, D) = (\overline{U} \to U)$ and 7) $\varphi \models [\overline{U}] \wedge [U]$. From (6) and the second rule for method choice, we have $mtype(reduce(P, \varphi), m, C) = mtype(reduce(P, \varphi), m, D) = (\overline{U} \to U)$.

Suppose the last rule used to derive $mtype(P, m, T) = (\overline{U} \to U)$ is the second rule for method type lookup. Thus, $T = C$ and we have 1) $P(C) =$ class C extends D implements I $\{\overline{T}\ \overline{f};\ K\ \overline{M}\}$ and 2) $U\ m(\overline{U}\ \overline{x})$ { return e; } $\in \overline{M}$. From (1), (2), and the second rule for method choice, we have 3) $mAny(P, M, C) = mAny(P, m, D)$. From (1) and the rule for class typing, we have that σ as a conjunct contains $[C] \Rightarrow [D]$. From $\varphi \models \sigma \wedge [T]$ we have 4) $\varphi \models [D]$. From (3), (4), and the induction hypothesis, we have 5) $mtype(reduce(P, \varphi), m, D) = (\overline{U} \to U)$ and 6) $\varphi \models [\overline{U}] \wedge [U]$. From (5) and the second rule for method choice, we have $mtype(reduce(P, \varphi), m, C) = mtype(reduce(P, \varphi), m, D) = (\overline{U} \to U)$.

Suppose the last rule used to derive $mtype(P, m, T) = (\overline{U} \to U)$ is the third rule for method type lookup. Thus, $T = I$ and we have 1) $P(I) =$ interface I $\{\overline{S}\}$ and 2) $U\ m(\overline{U}\ \overline{x}) \in \overline{S}$. From (1), (2), and the third rule for method choice, we have 3) $mAny(P, M, I) = [I.m()]$. From the assumption $\varphi \models [T] \wedge mAny(P, m, T)$, we have that $(reduce(P, \varphi))(I)$ is defined and that it has a signature $U\ m(\overline{U}\ \overline{x})$. So, $mtype(reduce(P, \varphi), m, I) = (\overline{U} \to U)$. From the rule for interface typing, we have that σ as a conjunct contains (4) $[I.m()] \Rightarrow ([I] \wedge [U] \wedge [\overline{U}])$. From (3), (4), and $\varphi \models \sigma \wedge mAny(P, m, I)$, we have $\varphi \models [U] \wedge [\overline{U}]$. □

Lemma A.7. *If* $\vdash P \mid \sigma$ *and* $P(C) =$ class C extends D implements I $\{\overline{T}\ \overline{f};\ K\ \overline{M}\}$ *and* $\varphi \models \sigma$ *and* $override(P, m, D, \overline{U} \to U)$*, then* $override(reduce(P, \varphi), m, D, \overline{U} \to U)$.

Proof. Our goal is to prove $override(reduce(P, \varphi), m, D, \overline{U} \to U)$, which by the rule for valid method overriding means that we must prove (1) $mtype(reduce(P, \varphi), m, D) = \overline{U'} \to U'$ implies $\overline{U'} = \overline{U}$ and $U' = U$.
If $mtype(reduce(P, \varphi), m, D) = \overline{U'} \to U'$ is not derivable, then (1) is vacuously true. If we can derive $mtype(reduce(P, \varphi), m, D) = \overline{U'} \to U'$, then from Lemma A.8 we have (2) $mtype(P, m, D) = \overline{U'} \to U'$. By assumption we have that we can derive $override(P, m, D, \overline{U} \to U)$ so by (2) and the rule for valid method overriding, we have $\overline{U'} = \overline{U}$ and $U' = U$, as required. □

Lemma A.8. *If* $\vdash P \mid \sigma$ *and* $P(C) =$ class C extends D implements I $\{\overline{T}\ \overline{f};\ K\ \overline{M}\}$ *and* $\varphi \models \sigma$ *and* $mtype(reduce(P, \varphi), m, C) = \overline{U} \to U$*, then* $mtype(P, m, C) = \overline{U} \to U$.

Proof. We proceed by induction on the derivation of $mtype(reduce(P, \varphi), m, C) = \overline{U} \to U$.

If the last rule used to derive $mtype(reduce(P, \varphi), m, C) = \overline{U} \to U$ was the first rule for method type lookup, then $(reduce(P, \varphi))(C)$ has a method m that also $P(C)$ has, so $mtype(P, m, C) = \overline{U} \to U$.

If the last rule used to derive $mtype(reduce(P, \varphi), m, C) = \overline{U} \to U$ was the second rule for method type lookup, then D has no method m and we have $mtype(reduce(P, \varphi), m, C) = mtype(reduce(P, \varphi), m, D)$. From the induction hypothesis, we have $mtype(reduce(P, \varphi), m, D) = mtype(P, m, D)$. Now we have two subcases.

In the first subcase, $P(C)$ has a method m. From the type rule for method typing applied to that method, we have $override(P, m, C, \overline{T} \to T)$, where $\overline{T}$ and T are the declared parameter and result types in the method. From the type rule for valid method overriding, we have that $mtype(P, m, C) = \overline{U} \to U$ implies $\overline{U} = \overline{T}$ and $U = T$. Thus, $mtype(P, m, C) = \overline{U} \to U$.

In the second subcase, $P(C)$ has no method m. In this case, we have from the second rule for method type lookup that $mtype(P, m, C) = mtype(P, m, D)$. Now we have that

$$mtype(P, m, C) = mtype(P, m, D) = mtype(reduce(P, \varphi), m, D) = mtype(reduce(P, \varphi), m, C) = \overline{U} \to U.$$

□

Lemma A.9. *If $\vdash P \mid \sigma$ and $P \vdash T \leq U \mid \pi$ and $\varphi \models \sigma \wedge [T] \wedge \pi$, then $reduce(P, \varphi) \vdash T \leq U \mid \pi$ and $\varphi \models [U]$.*

Proof. We proceed by induction on the derivation of $P \vdash T \leq U \mid \pi$.

If the last rule used in the derivation is the rule for reflexivity, then $P \vdash T \leq T \mid 1$ and we can use the rule for reflexivity to derive $reduce(P, \varphi) \vdash T \leq T \mid 1$. Additionally, we have from an assumption that $\varphi \models [T]$.

If the last rule used in the derivation is the rule for transitivity, then $P \vdash T \leq T'' \mid \pi_1 \wedge \pi_2$, which was derived from (1) $P \vdash T \leq T' \mid \pi_1$ and (2) $P \vdash T' \leq T'' \mid \pi_2$. From (1), the assumption that $\varphi \models [T]$, and the induction hypothesis, we have (3) $reduce(P, \varphi) \vdash T \leq T' \mid \pi_1$ and (4) $\varphi \models [T']$. From (2), (4), and the induction hypothesis, we have (5) $reduce(P, \varphi) \vdash T' \leq T'' \mid \pi_2$ and (6) $\varphi \models [T'']$. From (3), (5), and the rule for transitivity, we can derive $reduce(P, \varphi) \vdash T \leq T'' \mid \pi_1 \wedge \pi_2$. Additionally, we have $\varphi \models [T'']$ from (6).

If the last rule used in the derivation is the rule for $C \leq D$, then $P \vdash C \leq D \mid 1$, which was derived from (1) $P(C) = \text{class } C \text{ extends } D \text{ implements } I \; \{\overline{T}\;\overline{f};\; K\; \overline{M}\}$. By assumption, we have (2) $\varphi \models [C]$. We can use (1), (2), and the rule for $C \leq D$ to derive $reduce(P, \varphi) \vdash C \leq D \mid 1$. From the rule for class typing, we have that (3) σ as a conjunct contains $([C] \Rightarrow [D])$, so from (2) and (3), we have $\varphi \models [D]$.

If the last rule used in the derivation is the rule for $C \leq I$, then $P \vdash C \leq I \mid [C \triangleleft I]$, which was derived from (1) $P(C) = \text{class } C \text{ extends } D \text{ implements } I \; \{\overline{T}\;\overline{f};\; K\; \overline{M}\}$. By assumption, we have (2) $\varphi \models [C \triangleleft I]$. From the rule for class typing, we have that (3) σ as a conjunct contains $([C \triangleleft I] \Rightarrow [C])$. From (2), (3), and the assumption $\varphi \models \sigma$, we have that (4) $\varphi \models [C]$. We can use (1), (4), and the rule for $C \leq I$ to derive $reduce(P, \varphi) \vdash C \leq I \mid [C \triangleleft I]$. From the rule for class typing, we have that (5) σ as a conjunct contains $([C \triangleleft I] \Rightarrow [I])$. so from (2), (5), and the assumption $\varphi \models \sigma$, we have $\varphi \models [I]$. □

B Details of The Generalized Binary Reduction

In this section we go in the details of how we compute the variable order and the minimal satisfying assignment, furthermore we describe the properties of the progression. First, we will introduce a theoretical logical form which we use to guarantee correctness of the algorithms.

B.1 Implicative Positive Form

The correctness of our algorithm builds on that our logical expression has a special form which we call *implicative positive form.*

Definition B.1 (Implicative positive form)**.** The implicative positive form (IPF) is a conjunction of clauses with distinct variables (CNF), where at least one of the literals in each clause are positive.

This structure have some very nice properties that we use:

Lemma B.2. *A CNF R is an IPF if and only if $R(\text{VARS}(R))$.*

Lemma B.3. *If R is an IPF, then if we condition R by setting variables to true, we still have an IPF.*

There also exist a dual-IPF, where there must exist at least one negative literal in every clause.

Lemma B.4. *A CNF R can be formulated as an dual-IPF if and only if $R(\emptyset)$.*

Lemma B.5. *If R is an dual-IPF, then conditioning R by setting variables to false, we still have an dual-IPF.*

B.1.1 Proofs

Proof of Lemma B.2. First, let's assume that $R(\text{VARS}(R))$. Since R, is CNF, and therefore a conjunction of clauses, we can see that every clause have to be statisfied. A clause is only statisifed by settting all variables true, if at least one of the literals in the clause is positive. This proves the first direction.

Then, lets' assume that R is IPF. In this case all clauses have atleast one positive literal. For each such clause we can see that if all variables in the clause is true then the clause is also true. This logic extends to the conjunction of the clauses and that concludes the proof. □

Proof of Lemma B.3. Since conditioning an CNF returns a CNF, and if we condition R with a set of variables $X = \mathbf{1}$ then we know that $\text{VARS}(R) \setminus X$ is still a model to $(R \mid X = \mathbf{1})$. We can use Lemma B.2, with the fact that $(R \mid x = \mathbf{1})$ is a CNF, $\text{VARS}(R \mid x = \mathbf{1}) = \text{VARS}(R) \setminus \{X\}$, and $(R \mid X = \mathbf{1})(\text{VARS}(R) \setminus \{X\})$ we see that $(R \mid x = \mathbf{1})$ is IPF. □

The proofs of Lemmas B.4 and B.5 are like the proofs of Lemmas B.2 and B.3.

B.2 The Variable Order

We compute the variable order by mapping R_I to a directed graph in which each variable in R_I is a node and each clause yields edges from all positive variables to all negative variables. Specifically, $X^\wedge \Rightarrow Y^\vee$ yields the edges $y \to x$ for $(x, y) \in X \times Y$. Then, we get the variable order as the reverse post-order of the graph.

If all clauses are graph constraints, then X and Y are singletons, and this heuristics, represents the first step in the Kosaraju Sharir algorithm for calculating strongly connected components (SCC) [21].

Since $\text{MSA}_\leq$ is effectively a DFS from the smallest variables in all the clauses with only positive variables (Lemma B.9), then each element in the progression after the first will be a SCC in the graph. This means we will only add unions of SCC's to $\mathcal{L}$. Because $\text{MSA}_\leq$ chooses the smallest variable in each set in $\mathcal{L}$, The first element in the progression will always be the union of the SCC represented by $\mathcal{L}$. Since at least one variable in each set in $\mathcal{L}$ is required to satisfy P, and since each set in $\mathcal{L}$ is a SCC in R_I, the local minima is exactly the union of the sets in $\mathcal{L}$. This fact is used in Lemma B.14.

B.2.1 Example

The variable order in Section 4.5, was computed like this. We build the transposed graph in Figure 9, by drawing edges from the positive variables to the negated variables in each clause in Figure 2, effectively reversing each implication. The red arrows and numbers (top) are the result of a depth first search, and the black numbers are the post-order of that search. We extract the variable order, by reversing the post-order of the variables:

$$\begin{aligned}[\text{M}]_{20} \preceq [\text{M.main()}]_{19} \preceq [\text{I}]_{18} \preceq [\text{M.x()}]_{17} \preceq [\text{B}]_{16} \preceq [\text{B.n()}]_{15} \preceq [\text{B.n()!code}]_{14} \\ \preceq [\text{B.m()}]_{13} \preceq [\text{B} \lhd \text{I}]_{12} \preceq [\text{B.m()!code}]_{11} \preceq [\text{A}]_{10} \preceq [\text{A.n()}]_{9} \preceq [\text{I.n()}]_{8} \preceq [\text{A.n()!code}]_{7} \\ \preceq [\text{A.m()}]_{6} \preceq [\text{I.m()}]_{5} \preceq [\text{M.x()!code}]_{4} \preceq [\text{A} \lhd \text{I}]_{3} \preceq [\text{A.m()!code}]_{2} \preceq [\text{M.main()!code}]_{1}\end{aligned}$$

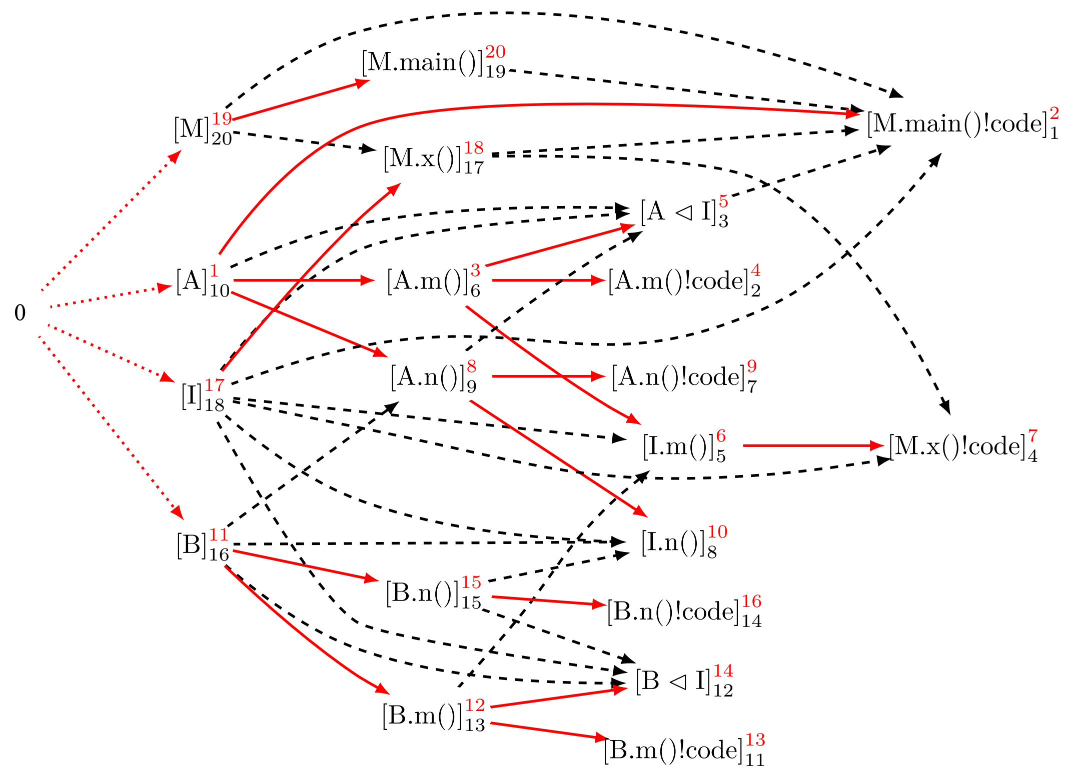

Figure 9. The transposed graph used to generate the variable order (red and dashed arrows). The 0 and the dotted red lines represent the root of the forest. The red arrows and numbers show the depth-first order, while the black numbers show the post-order. The reverse of the post-order is the variable order.

B.3 The Minimal Satisfying Assignment ($\mathrm{MSA}_{\leq}$)

The approximate minimal satisfying assignment computes a satisfying assignment, where we try to minimize the number of variables that has to be true.

Definition B.6 (Approx. Minimal Satisfying Assignment ($\mathrm{MSA}_{\leq}$)). Under the assumption that R is IPF, then

$$\mathrm{MSA}_{\leq}(R) = \begin{cases} \{x\} \cup \mathrm{MSA}_{\leq}(R \mid x = \mathbf{1}) & x \in \min_{\leq} O^{\cup}_{R} \\ \emptyset & o/w \end{cases}$$

where, O_R is the set of all clauses in R that only contain positive variables.

We compute $\mathrm{MSA}_{\leq}$ by choosing the smallest variable in the clauses that only contain positive variables until there are no such clauses left. At this point we have found a satisfying solution, because if there is no clauses with only positive variables then R is a dual-ipf, and the empty set is a solution (Lemma B.4).

If R is an IPF, we can gurantee that $\mathrm{MSA}_{\leq}$ will return a solution that only contain variables in R:

Lemma B.7. *Given any variable order* ($\leq$)*, the logical closure* $\mathrm{MSA}_{\leq}(R)$ *output a solution to* R*, only containing variables in* R*, given that* R *is IPF.*

$$R(\mathrm{MSA}_{\leq}(R)) \quad \wedge \quad \mathrm{MSA}_{\leq}(R) \subseteq \mathrm{VARS}(R)$$

Furthermore, we can prove that $\mathrm{MSA}_{\leq}(R)$ will return a set that contain smallest variable of all clauses with only positive literals in R. We use this fact to prove that our algorithm runs in polynomial time.

Lemma B.8. *Given any variable order* ($\leq$)*, and a* R *which is an IPF, then*

$$\{\min_{\leq} O \mid O \in O_R\} \subseteq \mathrm{MSA}_{\leq}(R)$$

R	O_R	$x \in \min_{\leq} O_R^{\cup}$
R	$\{[\text{M.main()!code}]\}$	$[\text{M.main()!code}]$
$(R \mid [\text{M.main()!code}] = \mathbf{1})$	$\{\{[\text{M.main()}]\}, \{[\text{M.x()}]\}, \{[\text{A}]\}, \{[\text{A} \lhd \text{I}]\}\}$	$[\text{M.main()}]$
$(R \mid [\text{M.main()}] = \mathbf{1})$	$\{\{[\text{M}]\}, \{[\text{M.x()}]\}, \{[\text{A}]\}, \{[\text{A} \lhd \text{I}]\}\}$	$[\text{M}]$
$(R \mid [\text{M}] = \mathbf{1})$	$\{\{[\text{M.x()}]\}, \{[\text{A}]\}, \{[\text{A} \lhd \text{I}]\}\}$	$[\text{M.x()}]$
$(R \mid [\text{M.x()}] = \mathbf{1})$	$\{\{[\text{I}]\}, \{[\text{A}]\}, \{[\text{A} \lhd \text{I}]\}\}$	$[\text{I}]$
$(R \mid [\text{I}] = \mathbf{1})$	$\{\{[\text{A}]\}, \{[\text{A} \lhd \text{I}]\}\}$	$[\text{A}]$
$(R \mid [\text{A}] = \mathbf{1})$	$\{\{[\text{A} \lhd \text{I}]\}\}$	$[\text{A} \lhd \text{I}]$
$(R \mid [\text{A} \lhd \text{I}] = \mathbf{1})$	$\emptyset$	$\bullet$

Figure 10. The initial run of $D_0 = \text{MSA}_{\leq}(R) = \{[\text{A}], [\text{A} \lhd \text{I}], [\text{I}], [\text{M}], [\text{M.x()}], [\text{M.main()}], [\text{M.main()!code}]\}$. Each row corresponds to a recursive call to $\text{MSA}_{\leq}$. The column R represents the value of R at that call, O_R is the set of positive closures in R, and x is the smallest variable in the union of the sets.

Finally, we can prove that $\text{MSA}_{\leq}(R)$ esentially is a DFS from all the smallest variables in all the clauses that only contain positive variables, if R can be represented using graph constraints and clauses with only positive variables. We use this to prove that GBR is locally optimal for graphs.

Lemma B.9. *Given any variable order* $(\leq)$*, and* R *where all clauses are graph constraints* $(x \Rightarrow y)$ *or clauses with only positive variables. Then* $\text{MSA}_{\leq}$ *produces minimal closure that contains the smallest variable from each clause in* O_R.

B.3.1 Example

We calculate D_0 in Section 4.5, by finding the minimal satisfying sassingment on the unmodified constraints. We illustrate this step by step in Figure 10.

B.3.2 Proofs

Proof of lemma B.7. Proof by induction on size of R. If R is empty, then O_R is empty, which means that $\text{MSA}_{\leq}(R) = \emptyset$. When R contains no clauses then all sets are solutions, including the empty one $R(\text{MSA}_{\leq}(R))$. And since $\text{MSA}_{\leq}(R) = \emptyset$ we know that $\text{MSA}_{\leq}(R) \subseteq \text{VARS}(R)$.

For all R's we get the induction hypothesis, that for any R', which is in IPF and contains fewer variables than R:

$$R'(\text{MSA}_{\leq}(R')) \wedge \text{MSA}_{\leq}(R') \subseteq \text{VARS}(R')$$

There are now two cases, either $x \in O_R$ or O_R is empty. In the case where O_R is empty, then $\text{MSA}_{\leq} R = \emptyset$ and we know that all clauses in R have at least one positive variable which means that it is a dual-ipf, which in turn means that the empty set is a solution Lemma B.4, which is also clearly a subset of the variables of R.

In the second case, there exist an x which is a positive variable in $\text{VARS}(R)$, where $\text{MSA}_{\leq}(R) = \{x\} \cup \text{MSA}_{\leq}(R \mid x = \mathbf{1})$. Conditioning with $x = \mathbf{1}$ in R result in a IPF (Lemma B.3), with fever variables because $x \in \text{VARS}(R)$. We can now use the induction hypothesis with $R' = (R \mid x = \mathbf{1})$, from which we get that $(R \mid x = \mathbf{1})(\text{MSA}_{\leq}(R \mid x = \mathbf{1}))$ and $\text{MSA}_{\leq}(R \mid x = \mathbf{1}) \subseteq \text{VARS}(R \mid x = \mathbf{1})$. From this we get

$$\begin{aligned}\text{MSA}_{\leq}(R) &\subseteq \{x\} \cup \text{MSA}_{\leq}(R \mid x = \mathbf{1})\\ &\subseteq \{x\} \cup \text{VARS}(R \mid x = \mathbf{1})\\ &\subseteq \text{VARS}(R).\end{aligned}$$

And from $(R \mid x = \mathbf{1})(\text{MSA}_{\leq}(R \mid x = \mathbf{1}))$ we can see that $R(\{x\} \cup \text{MSA}_{\leq}(R \mid x = \mathbf{1})$ which by substitution concludes the proof. □

Proof of lemma B.8. Proof by induction on the number of variables in R.

The base case, where there are no variables in R, it is also the case that O_R is empty, which means that proof is trivial.

The inductive case we know that for all R' smaller than R:

$$\{\min_{\leq} O \mid O \in O_{R'}\} \subseteq \text{MSA}_{\leq}(R')$$

We can see there are two cases. Either there are O is empty, in which case the lemma is trivially true or there exist a $x \in \min_{\leq} O^{\cup}$, s.t. $\text{MSA}_{\leq}(R) = \{x\} \cup \text{MSA}_{\leq}(R \mid x = \mathbf{1})$. Because x is the smallest element in all the sets, we know that it is all so the smallest element in each clause it is an element of. We can split O_R in two sets O_R^x and $O_R^{!x}$, where x is contained in O_R^x. It is clear that x is the smallest element in O_R^x, so we only have to prove:

$$\{\min_{\leq} O \mid O \in O_R^{!x}\} \subseteq \{x\} \cup \text{MSA}_{\leq}(R \mid x = \mathbf{1})$$

Because x is not an element in $O_R^{!x}$, we know that $O_R^{!x} \subseteq O_{(R \mid x=\mathbf{1})}$. And since $(R \mid x = \mathbf{1})$ is smaller than R ($x \in \text{VARS}(R)$) we can use the induction hypothesis, to see that

$$\{\min_{\leq} O \mid O \in O_R^{!x}\} \subseteq \{\min_{\leq} O \mid O \in O_{(R \mid x=\mathbf{1})}\} \subseteq \text{MSA}_{\leq}(R \mid x = \mathbf{1}),$$

which concludes the proof. □

Proof of lemma B.9. By inspection of $\text{MSA}_{\leq}$ we can see that it acts precisely as a best-first search. Where the minimal variable from O_R is chosen as the next element each time. By conditioning on R with x we essentially mark x as visited in the graph and add all its neighbors to O_R. Following the same argument as in lemma B.8, we will add the smallest set from each clause ever added to O_R. Since a best-first search finds a minimal closure from the set in its queue, and essentially the queue consist of the first element in each set in O_R, we have proved the lemma. □

B.4 Progression

We have disccused the the progression in the main text of the paper. We have identified the main properties of the progression. Given an CNF R_I, a set of learned sets $\mathcal{L}$, and an input space J, then for a progression $\mathcal{D} = \text{PROGRESSION}_{R_I}(\mathcal{L}, J)$ Assuming $R_I(J)$ and that $\forall L \in \mathcal{L}.\ J \cap L \neq \emptyset$:

- The progression is a non-empty split of the input space J.
$$|\mathcal{D}| > 0 \quad \wedge \quad \mathcal{D}^{\cup} = J \quad \wedge \quad \forall i, j.\ i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset. \tag{SPLIT}$$
- The progression is *correct*, if all non-empty prefixes of the progression is a solution to R_I and intersects with all elements in $\mathcal{L}$.
$$\forall r \geq 0. \quad R_I(\mathcal{D}^{\cup}_{\leq r}) \quad \wedge \quad \forall L \in \mathcal{L}.\ \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset \tag{CORRECT}$$
- The progression is *locally optimal* if the first set D_0 in the progression is minimal:
$$\forall T \subseteq \mathcal{D}_0. \quad R_I(T) \wedge (\forall L \in \mathcal{L}. T \cap L \neq \emptyset) \Rightarrow T = \mathcal{D}_0 \tag{LOC-OPT}$$

And we can prove that our progression have these properties

Lemma B.10. *Our progression is (SPLIT) and (CORRECT).*

Lemma B.11. *Given that R_I is represented using graph constraints then our progression is (LOC-OPT).*

Furthermore, we can prove that our progression run in polynomial time.

Lemma B.12. *Assuming that R_I is a CNF, $R_I(J)$ and $\forall L \in \mathcal{L}.\ J \cap L \neq \emptyset$, our progression runs in polynomial time.*

B.4.1 Proofs

Proof of Lemma B.10. Initially we see that $\mathcal{D} = \text{PROGRESSION}_{R_I}(\mathcal{L}, J)$, and we get to assume that $R_I(J)$ and that $\forall L \in \mathcal{L}.\ J \cap L \neq \emptyset$. From these two assumptions we can see that the progression algorithm will terminate and all calls to $\text{MSA}_{\leq}$ are IPFs Lemmas B.12 and B.13.

We start by a proof by induction over the prefixes of the progression. Our IH is for all $0 \leq r < |\mathcal{D}|$:

$$R^{+}(\mathcal{D}^{\cup}_{\leq r}) \quad \wedge \quad \forall i, j \leq r.\ i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$$

Let's start with the base case $r = 0$. We see that $R^{+}(\mathcal{D}^{\cup}_{\leq r})$, from $\mathcal{D}^{\cup}_{\leq r} = \mathcal{D}_0 = \text{MSA}_{\leq}(R^{+})$, lemma B.7. The second conjunct is trivially true.

Now let's prove it for the case where $r \geq 1$. We get the induction hypothesis for all $r' < r$. We know that $\mathcal{D}_r = \text{MSA}_{\leq}(R^{+} \wedge x \mid \mathcal{D}^{\cup}_{\leq k} = \mathbf{1})$ and that $x \in J \setminus \mathcal{D}^{\cup}_{\leq k}$, where $k = r - 1$. From Lemma B.7 we know that $(R^{+} \wedge x \mid \mathcal{D}^{\cup}_{\leq k} = \mathbf{1})(\mathcal{D}_r)$, which also means that $R^{+}(\mathcal{D}^{\cup}_{\leq k} \cup \mathcal{D}_r)$ which is $R^{+}(\mathcal{D}^{\cup}_{\leq r})$, we can also see that $\mathcal{D}_r \subseteq \text{VARS}(R^{+} \wedge x \mid \mathcal{D}^{\cup}_{\leq k} = \mathbf{1})$ which means that $\mathcal{D}_r$ does not intersect with any elements in $\mathcal{D}_{\leq k}$, and because we know $\forall i, j \leq k.\ i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ from the induction hypothesis, we get $\forall i, j \leq r.\ i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$.

Using this IH we can satisfy each of the requirements in (SPLIT): $|\mathcal{D}| > 0$ is trivial. $\mathcal{D}^{\cup} = J$, follows from that on exit with $i = r$, we know know that $\nexists x \in J \setminus \mathcal{D}^{\cup}_{\leq r}$, because otherwise there would exist yet another element in $\mathcal{D}$. $\forall i, j.\ i \neq j \Rightarrow$

$\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$. Trivial from the IH. And now (CORRECT), for all $r \geq 0$, which is only 0. Using basic logical operations we hat for all X, $R^{+}(X) \Rightarrow R_I(X) \wedge (\bigvee_{L \in \mathcal{L}} L^{\vee})(X)$, which correspond directly to $R_I(\mathcal{D}^{\cup}_{\leq r})$ and $(\forall L \in \mathcal{L}.\ \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset)$. Which concludes the proof. □

Proof of Lemma B.11. From Lemma B.14, we can see that every call to our progression $(\mathcal{L}', J')$ produces strongly connected components (SCC). The GBR only add elements from the progression $\mathcal{D}$ (not including the first) to $\mathcal{L}$, so we know that we only add SCC's to $\mathcal{L}$.

Because we know that $\mathcal{L}$ only contain SCCs, then O_R will only be SCCs. The property of SCCs is that the closure from any element is equal to the closure from all. This means that when we use that if R_I is a graph $\text{MSA}_{\leq}$ finds a closure from smallest elements in each set in O_R (Lemma B.9), We can conclude that $\mathcal{D}_0 = \text{MSA}_{\leq}(R)$ will be the closure that contain $O_R^{\cup}$. This closure is the smallest which contain a single element from each element in $\mathcal{L}$. This concludes the proof. □

Proof of Lemma B.12. Because each call to $\text{MSA}_{\leq}$ is an IPF (Lemma B.13), we can can see that each element after the first must contain at least one element x (Lemma B.7). This means that we can call $\text{MSA}_{\leq}$ no more than $|I|$ times. Each call to $\text{MSA}_{\leq}$ can only do one conditioning for each clause, which takes $|R|$ time. All other operations are also polynomial in time. □

B.4.2 Auxiliary lemmas

Lemma B.13. *Assuming that R_I is a CNF, $R_I(J)$ and $\forall L \in \mathcal{L}.\ J \cap L \neq \emptyset$, then calls to $\text{MSA}_{\leq}$ in our progression are IPF's.*

Proof. We can first see that R is an IPF. We can see because R_I is CNF, and we only add clauses with at least one positive variable in J to R_I, and limit it to only having variables in J. From this fact we can see that all calls to $\text{MSA}_{\leq}$ is IPF. For $i = 0$ it is trivial, for $i = k + 1$ we can clearly see that $R \wedge x$ is an IPF since x is a positive variable. And, since we are allowed to condition with positive variables Lemma B.3 $(R \wedge X \mid \mathcal{D}^{\cup}_{\leq k} = \mathbf{1})$ is IPF. □

Lemma B.14. *Given a R_I that contains only graph constraints, then all elements in each progression produced by our progression algorithm ($\mathcal{D} = \text{PROGRESSION}_{R_I}(\mathcal{L}, J)$) after the first, are strongly connected components in R_I*

Proof. Our variable order is the reverse post-order of the transposed over-approximated graph of R_I. Since the solution of a logical graph correspond directly with the closures in the graph, we know that the first element $\mathcal{D}_0 = \text{MSA}_{\leq}(R)$ contain a closure of the smallest elements in clauses (Lemma B.8).

For all $k > 0$, $(R \mid D^{\cup}_{\leq k} = \mathbf{1})$ is a logical graph with only edges. From this point, each element in the progression $\mathcal{D}_i$ correspond to a closure of x in $(R \mid D^{\cup}_{\leq i} = \mathbf{1})$. Because x is the smallest in the reverse post-order of the transposed graph, then we know that there exist no edges from x to any other element than x, without there also existing an transitive edge in the opposite direction. This means that any closure in the graph at this point containing x will be a SCC in $(R \mid D^{\cup}_{\leq k} = \mathbf{1})$. And because any SCC in a subgraph is also a SCC in the full graph R. We have, therefore, proved the lemma. □

C Proof of the Generalized Binary Reduction

C.1 Proof that Input Reduction Problem is NP-complete

Proof of Theorem 4.2. The Input Reduction Problem is in NP because given S, we can check in polynomial time that $P(S) \wedge R_I(S) \wedge |S| < k$ because both P and R_I can be evaluated in polynomial time.

The Input Reduction Problem is NP-hard via a reduction from the Hitting Set Problem. The Hitting Set Problem is: given (I, Z, k), where $Z \subseteq 2^I$ is a set of sets, decide $\exists S \in 2^I : (\forall X \in Z : S \cap X \neq \emptyset) \wedge (|S| < k)$. The reduction works as follows. Define $P(S) = (\forall X \in Z : S \cap X \neq \emptyset)$ and $R_I(S) = \mathbf{1}$. Notice that these definitions satisfy all the assumptions of Input Reduction Problem. In particular, P can be evaluated in polynomial time and both $P(I)$ and $R_I(I)$ are true. Additionally, since R_I is always true, we see easily from the definition of P that P is monotonic.

If (I, P, R_I, k) has solution S, then $P(S) = (\forall X \in Z : S \cap X \neq \emptyset)$ is true, and $|S| < k$. This means that S is also a solution to (I, Z, k).

Conversely, if (I, Z, k) has solution S, then $(\forall X \in Z : S \cap X \neq \emptyset) \wedge (|S| < k)$. This means that $P(S)$ is true, and since $R_I(S)$ is true and $|S| < k$, we have that S is also a solution to (I, P, R_I, k). □

C.2 Proof of Main Theorems

Proof of Theorem 4.4 (Polynomial Time). From Lemma B.8 we can see that initial set of the progression will contain the smallest variables in $\mathcal{L}$ in respect to the variable order: $\{\min_{\leq} L \mid L \in \mathcal{L}\} \subseteq \mathcal{D}_0$. We know that $\mathcal{D}_r \cap \mathcal{D}_0 = \emptyset$ for any $r \neq 0$ (SPLIT), and therefore also

$$\{\min_{\leq} L \mid L \in \mathcal{L}\} \subset \{\min_{\leq} L \mid L \in \mathcal{L} \cup \{\mathcal{D}_r\}\}.$$

This means that for every iteration the lower bound of $\mathcal{D}_0$ increases. Since we can only increase the lower bound $|I|$ times, we can only add $\mathcal{D}_r$ to $\mathcal{L}$ $O(|I|)$ times. This, in turn, means we can only run the loop $O(|I|)$ times. Since all operations, including calculating the progression (Lemma B.12), is polynomial in time, we know that GBR runs in polynomial time. □

Proof of Theorem 4.5 (Local Minima). First we can see that GBR will terminate (Theorem 4.4), so now we just have to prove that GBR on (I, P, R_I) will return an $S = \mathcal{D}_0$ st.

$$\forall T \subseteq S. \quad P(T) \wedge R_I(T) \iff T = S$$

In the first direction ($\Rightarrow$), we get to assume $T \subseteq \mathcal{D}_0 \subseteq \mathcal{D}^{\cup}$, that $P(T)$ and $R_I(T)$, and we know that the algorithm must have terminated with $P(\mathcal{D}_0)$. We can now use

$$\left(\forall T \subseteq \mathcal{D}^{\cup}. \quad P(T) \wedge R_I(T) \Rightarrow \forall L \in \mathcal{L}.\, T \cap L \neq \emptyset\right)$$

from Lemma 4.3, to see that $\forall L \in \mathcal{L}.\, T \cap L \neq \emptyset$ which we can use with $R_I(T)$ in (LOC-OPT) to see that $T = \mathcal{D}_0$. We get (LOC-OPT) from the fact that R_I is described using graph constraints and Lemma B.11.

In the other direction ($\Leftarrow$). Since the loop condition is false we know that $P(\mathcal{D}_0)$, furthermore from the invariant (Lemma 4.3) we know that $\mathcal{D}_0 \subseteq \mathcal{D}^{\cup} \subseteq I$ and $R_I(\mathcal{D}^{\cup}_{\leq 0}) = R_I(\mathcal{D}_0)$. We have therefore proven that S is a local minimal solution to the input reduction problem. □

C.3 Proof of Invariant

Proof of Lemma 4.3 (Invariant). We know that our progression is (SPLIT) and (CORRECT) from Lemma B.10. We can use this fact to, for every step of the generalized binary reduction on (I, P, R_I), show that the following invariant holds:

$$\begin{aligned} \mathrm{Inv}(\mathcal{L}, \mathcal{D}) \equiv \;& P(\mathcal{D}^{\cup}) \;\wedge\; |\mathcal{D}| > 0 \;\wedge\; \mathcal{D}^{\cup} \subseteq I \;\wedge\; (\forall i, j.\, i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset) \\ & \wedge\; \left(\forall r \geq 0. \quad R_I(\mathcal{D}^{\cup}_{\leq r}) \;\wedge\; \forall L \in \mathcal{L}.\, \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset\right) \\ & \wedge\; \left(\forall T \subseteq \mathcal{D}^{\cup}. \quad P(T) \wedge R_I(T) \Rightarrow \forall L \in \mathcal{L}.\, T \cap L \neq \emptyset\right) \end{aligned}$$

Initially, with $\mathcal{L} = \emptyset$ and $R_I(I)$, we can see that $\forall L \in \mathcal{L}.\, \mathcal{D}^{\cup} \cap L \neq \emptyset$ is trivially true. This means that we can use that $\mathcal{D} = \mathrm{PROGRESSION}_{R_I}(\mathcal{L}, I)$ is a correct progression which means that we can use (SPLIT) and (CORRECT). We can now prove each conjunct in the invariant:

- $P(\mathcal{D}^{\cup})$ from the input requirement $P(I)$ and $\mathcal{D}^{\cup} = I$ (SPLIT).
- $|\mathcal{D}| > 0$ from (SPLIT).
- $\mathcal{D}^{\cup} \subseteq I$ from $\mathcal{D}^{\cup} = I$ (SPLIT).
- $\forall i, j.\, i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ from (SPLIT).
- $(\forall r \geq 0.\, R_I(\mathcal{D}^{\cup}_{\leq r}) \wedge \forall L \in \mathcal{L}.\, \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset)$ from (CORRECT).
- $(\forall T \subseteq \mathcal{D}^{\cup}.\, P(T) \wedge R_I(T) \Rightarrow \forall L \in \mathcal{L}.\, T \cap L \neq \emptyset)$ is trivially true because $\mathcal{L} = \emptyset$.

For each iteration of the while loop, we get assume the invariant. First we see that there exist a $\mathcal{D}_0$ in $\mathcal{D}$, which we get from the invariant ($|\mathcal{D}| > 0$). Then for entering the loop body we get that $\neg P(\mathcal{D}_0)$. In the loop we make the following updates:

$$\begin{aligned} r &= \min r \text{ st. } r > 0 \wedge P(\mathcal{D}^{\cup}_{\leq r}) \\ \mathcal{L}' &= \mathcal{L} \cup \{\mathcal{D}_r\} \\ \mathcal{D}' &= \text{PROGRESSION}_{R_I}\left(\mathcal{L}',\ \mathcal{D}^{\cup}_{\leq r}\right) \end{aligned}$$

First we need to prove that there exist a minimal $r > 0$, where $P(\mathcal{D}^{\cup}_{\leq r})$. We know that $P(\mathcal{D}^{\cup})$ and because $\mathcal{D}^{\cup}$ is equal to the longest prefix $\mathcal{D}^{\cup}_{\leq |D|-1}$, we know that there exist an $r = |D| - 1$, st $P(\mathcal{D}^{\cup}_{\leq r})$. From the loop check we know that $\neg P(\mathcal{D}_0)$, which means that we can conclude that $|D| > 1$, because otherwise $\mathcal{D}_0 = \mathcal{D}^{\cup}$, and $P(\mathcal{D}_0)$ which is a contradiction.

The only way $\mathcal{D}^{\cup}_{\leq r} \cap \mathcal{D}_r = \emptyset$ is if $\mathcal{D}_r = \emptyset$, but if that is the case we also know that $P(\mathcal{D}^{\cup}_{\leq r-1})$, which is a contradiction, because r is the smallest st $P(\mathcal{D}^{\cup}_{\leq r-1})$. So we know $\mathcal{D}^{\cup}_{\leq r} \cap \mathcal{D}_r \neq \emptyset$. From the invariant we can see that $\forall L \in \mathcal{L}.\ \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset$, which means that $\forall L \in \mathcal{L}'.\ \mathcal{D}^{\cup}_{\leq r} \cap L \neq \emptyset$. The invariant also gives us that $R_I(\mathcal{D}^{\cup}_{\leq r})$ and that means we have both preconditions to allow us to use that $\mathcal{D}'$ is a correct progression, so we know (SPLIT) and (CORRECT).

We can now prove each conjunct in the invariant $\text{Inv}(\mathcal{L}', \mathcal{D}')$, except the last:

- $P(\mathcal{D}'^{\cup})$ from the input requirement $P(\mathcal{D}^{\cup}_{\leq r})$ and $\mathcal{D}'^{\cup} = \mathcal{D}^{\cup}_{\leq r}$ (SPLIT).
- $|\mathcal{D}'| > 0$ from (SPLIT).
- $\forall i, j.\ i \neq j \Rightarrow \mathcal{D}'_i \cap \mathcal{D}'_j = \emptyset$ from (SPLIT).
- $(\forall r \geq 0.$
- $\mathcal{D}'^{\cup} \subseteq I$ from $\mathcal{D}'^{\cup} = \mathcal{D}^{\cup}_{\leq r}$ and $\mathcal{D}^{\cup}_{\leq r} \subseteq \mathcal{D}^{\cup} \subseteq I$ (SPLIT).
- $(\forall r \geq 0.\ R_I(\mathcal{D}'^{\cup}_{\leq r}) \wedge \forall L \in \mathcal{L}'.\ \mathcal{D}'^{\cup}_{\leq r} \cap L \neq \emptyset)$ from (CORRECT).

We now only have to prove:

$$\forall T \subseteq \mathcal{D}'^{\cup}.\ P(T) \wedge R_I(T) \Rightarrow \forall L \in \mathcal{L}'.\ T \cap L \neq \emptyset$$

We get to assume that $T \subseteq \mathcal{D}'^{\cup}$, and $P(T)$ and $R_I(T)$ and we have to prove that

$$\begin{aligned} \forall L \in \mathcal{L}'.T \cap L \neq \emptyset &\impliedby \forall L \in \mathcal{L} \cup \{\mathcal{D}_r\}.T \cap L \neq \emptyset && (1) \\ &\impliedby T \cap \mathcal{D}_r \neq \emptyset \wedge \forall L \in \mathcal{L}.T \cap L \neq \emptyset && (2) \\ &\impliedby T \cap \mathcal{D}_r \neq \emptyset && (3) \end{aligned}$$

We get (1) from $\mathcal{L}' = \mathcal{L} \cup \{\mathcal{D}_r\}$, and (2) by expanding $\{\mathcal{D}_r\}$ out of the for all statement, and finally we get $\forall L \in \mathcal{L}.T \cap L \neq \emptyset$ from the invariant, because $T \subseteq \mathcal{D}'^{\cup} \subseteq \mathcal{D}^{\cup}_{\leq r} \subseteq \mathcal{D}^{\cup}$ (SPLIT), and $P(T)$ and $R_I(T)$ which leaves us to prove $T \cap \mathcal{D}_r \neq \emptyset$ (3).

Proving $T \cap \mathcal{D}_r \neq \emptyset$ is straight forward. If $T \cap \mathcal{D}_r = \emptyset$ then we know that $T \subseteq \mathcal{D}^{\cup}_{\leq r-1}$, and because P is monotone over solutions to R_I, $R_I(\mathcal{D}^{\cup}_{\leq r-1})$ and $R_I(T)$, then we know that $P(T) \sqsubseteq P(\mathcal{D}^{\cup}_{\leq r-1})$. Since we know that $P(T)$, we also know that $P(\mathcal{D}^{\cup}_{\leq r-1})$, this a contradiction because r is the smallest r st $P(\mathcal{D}^{\cup}_{\leq r})$. Thus we have proven that $\text{Inv}(\mathcal{L}, \mathcal{D})$ is true for all steps of the algorithm. □