Supplementary Material for LEGOS-A: Legal Compliance Verifier via Satisfiability Checking

In this document, we provide the supplementary material for our submission: "LEGOS-A: Legal Compliance Verifier via Satisfiability Checking". Specifically, Sec. 1 provides the correctness proof for the global lower bound (GLB), local upper bound (LUB) and the grounding algorithm G_A (Alg. 1). Sec. 2 prove the correctness, termination and solution optimality of SEARCH-A. Sec. 3 present SEARCH-A's support for aggregation function *Count*, *Max* and *Min*. Sec. 4 illustrates the over- and under-approximation of the summation. Sec. 5 illustrates the algorithm IBSC.

1 Correctness Proof for GLB and LUB

In this section, we prove the correctness of global lower bound (GLB) and local upper bound (LUB). To make the document self-contained, we first present the necessary background (from [1]) for understanding LUB and GLB, including the definition for LUB, GLB, stratified sum (inspired by stratified recurrence [8]), over/ under-approximation for the summation, and the grounding algorithm G_A(Sec. 1.1). Then we state the main correctness lemmas for GLB and LUB (Lemma 2 and Lemma 3)(Sec. 1.2). Finally, we provide the full proof of Lemma 2 and Lemma 3. We prove the lemmas together by induction (in the layer where the target sum is stratified). We prove the base case in Sec. 1.3, the inductive step in Sec. 1.4 and conclude the proof in Sec. 1.5.

1.1 Background

Definition 1 (Stratified Sum). *A summation* $Sum(S, p, val)$ *is* stratified at layer 0 *if for every* $s \in S$, $p(s)$ *and* $val(s)$ *do not contain summations.* $Sum(S, p, val)$ *is* stratified at layer n *if for every* $s \in S$, $p(s)$ *and* $val(s)$ *only contain summation that are stratified at layer* $n-1$ *or lower. Given an* FOL^{*+} *formula* ϕ *with* N *unique functions, if* $Sum(S, p, val)$ *is an expression in* ϕ*, then* $Sum(S, p, val)$ *is* stratified *if and only if it is stratified at layer* N *or below.*

Definition 2 (Global lower-bound). *Let* $sum = Sum(S, p, val)$ *be a summation, and* $D_\downarrow$ *be a domain.* $\mathrm{GLB}^{sum}_{D_\downarrow}$ *is a* global lower bound *of sum in* $D_\downarrow$ *if and only if for every domain* $D \supseteq D_\downarrow$, $\mathrm{GLB}^{sum}_{D_\downarrow} \leq sum_D$*, where* $sum_D = \sum_{s \in S \subseteq D} ite(s.ext \wedge p(s), val(s), 0)$ *is the under-approximated summation in* D.

Definition 3 (Local upper-bound). *Let sum be a summation in the form of* $Sum(S, p, val)$*, and* $D_\downarrow$ *be a domain.* $\mathrm{LUB}^{sum}_{D_\downarrow}$ *is a* local upper-bound *sum in domain* $D_\downarrow$ *if and only if* $\mathrm{LUB}^{sum}_{D_\downarrow} \geq sum_{D_\downarrow}$*, where* $sum_{D_\downarrow} = \sum_{s \in S \subseteq D} ite(s.ext \wedge p(s), val(s), 0)$ *is the under-approximated summation in* $D_\downarrow$.

Definition 4 (Global lower-bound function). GLB *is a function that receives a numerical term t and a domain $D_\downarrow$ and computes* $\mathrm{GLB}(t, D_\downarrow)$ *as:*

$$\begin{cases} t & \textit{if } t = v \mid c \\ -\mathrm{LUB}(t_1, D_\downarrow) & \textit{if } t = -t_1 \\ GLB(t_1, D_\downarrow) + GLB(t_2, D_\downarrow) & \textit{if } t = t_1 + t_2 \\ c \times ite(c \geq 0,\ \mathrm{GLB}(t_1, D_\downarrow),\ \mathrm{LUB}(t_1, D_\downarrow)) & \textit{if } t = c \times t_1 \\ \sum_{s \in S \subseteq D_\downarrow} ite(\neg s.ext \vee \mathrm{G\_A}(\neg p(s), D_\downarrow), 0, \mathrm{GLB}(val(s), D_\downarrow)) & \textit{if } t = Sum(S, p, val) \end{cases}$$

where G_A *is the extended version of* G *(Alg. 1) for computing the over-approximation of an FOL^{*+} formula, and* LUB *is a function that computes a numerical term's local upper bound (Def. 5).*

Definition 5 (Local upper-bound function). LUB *is a function that receives a numerical term s and a domain $D_\downarrow$ and computes* $\mathrm{LUB}(t, D_\downarrow)$ *as:*

$$\begin{cases} t & \textit{if } t = v \mid c \\ -\mathrm{GLB}(t_1, D_\downarrow) & \textit{if } s = -t_1 \\ \mathrm{LUB}(t_1, D_\downarrow) + \mathrm{LUB}(t_2, D_\downarrow) & \textit{if } s = t_1 + t_2 \\ c \times ite(c \geq 0,\ \mathrm{LUB}(t_1, D_\downarrow),\ \mathrm{GLB}(t_1, D_\downarrow)) & \textit{if } t = c \times t_1 \\ \sum_{s \in S \subseteq D_\downarrow} ite(s.ext \wedge \mathrm{G\_A}(p(s), D_\downarrow), \mathrm{LUB}(val(s), D_\downarrow), 0) & \textit{if } t = Sum(S, p, val) \end{cases}$$

where GLB *is a function that computes a term's global lower bound (Def. 4).*

Definition 6 (Over-approximation). *Given a summation sum in the form of $Sum(S, p, val)$ and a domain $D_\downarrow$, let $\mathrm{GLB}^{sum}_{D_\downarrow}$ and $\mathrm{LUB}^{sum}_{D_\downarrow}$ be the sum's global lower-bound and local upper-bound at $D_\downarrow$, respectively. The* over-approximation of *sum is a new integer variable i that satisfies the following constraints, denoted as req_{sum}:*

(a) $i \geq \mathrm{GLB}^{sum}_{D_\downarrow}$
(b) if $i > \mathrm{LUB}^{sum}_{D_\downarrow}$ *then* $\exists s' \cdot p(s') \ \wedge \ val(s') + Sum(S, \lambda s : s \neq s' \wedge p(s), val) = i$

Lemma 1 (Under-approximation soundness). *Given an FOL^{*+} formula ϕ and a domain $D_\downarrow$, let ϕ_g be the over-approximation computed by $\mathrm{G\_A}(\phi, D_\downarrow)$, and let $\phi_g^\perp = \phi_g \wedge Inc(\phi_g, D_\downarrow) \wedge \bigwedge_{sum \in \phi_g}(sum = sum_{D_\downarrow})$ where every sum in ϕ_g is under-approximated by $sum_{D_\downarrow}$. Then $\phi_g^\perp$ is an under-approximation of ϕ (i.e., if $\phi_g^\perp$ has a solution, then it must be a solution to ϕ).*

Proof. If ϕ does not contain summations, then $\phi_g^\perp = \phi_g \wedge Inc(\phi_g, D_\downarrow)$, and it is an under-approximation of ϕ (Lemma 3 of [6]). If ϕ contains summations, then for every summation sum in the form of $Sum(S, p, val)$, its under-approximation $sum_{D_\downarrow} = \sum_{s \in S \subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)$ matches the interpretation of sum in the domain $D_\downarrow$. Therefore, if σ is a solution to $\phi_g^\perp$ then σ is a solution to ϕ in the domain $D_\downarrow$.

Algorithm 1 G_A: ground a FOL*+ formula in a domain $D_\downarrow$.

Input an FOL*+ formula ϕ and a domain $D_\downarrow$.
Input b for optimization boundary case reduction.
Output a grounded quantifier-free formula ϕ_g over relational objects.

if $\textsc{Match}(\phi,\ \exists o : r \cdot \phi')$ **then**
  $o' \leftarrow \textsc{NewAct}(r)$
  $\phi'_g \leftarrow o'.ext \wedge \mathrm{G\_}A(\phi'[o \leftarrow o'], D_\downarrow, b)$
  if b **then**
    $\phi_g \leftarrow \phi_g \wedge \mathrm{G\_A}(\mathrm{BCR}(\phi'), D_\downarrow, b)$
  return ϕ_g
if $\textsc{Match}(\phi,\ \forall o : r \cdot \phi')$ **then**
  return $\bigwedge_{o' : r \in D_\downarrow} o'.ext \Rightarrow \mathrm{G\_}A(\phi'[o \leftarrow o'],\ D_\downarrow, b)$
if $\textsc{Match}(\phi,\ f(t_1, \ldots, t_n)$ **then**)
  return $f(\mathrm{G\_}A(t_1, D_\downarrow, b), \ldots, \mathrm{G\_}A(t_n, D_\downarrow, b))$
if $\textsc{Match}(\phi,\ Sum(r, p, val))$ **then**
  $i \leftarrow \textsc{Nat}()$; **reg**$(Req_{sum}, i)$; **return** i
return ϕ ▷ The case if ϕ is atomic.

1.2 Correctness Lemmas for GLB and LUB

Lemma 2 (GLB and LUB correctness). *For every domain $D_\downarrow$ and sum in the form of $Sum(S, p, val)$, $\mathrm{GLB}(sum, D_\downarrow)$ computes a global lower-bound, and $\mathrm{LUB}(sum, D_\downarrow)$ computes a local upper-bound.*

Lemma 3 (Over-approximation soundness). *Given an FOL^{*+} formula ϕ and a domain $D_\downarrow$, $\mathrm{G\_A}(\phi, D_\downarrow)$ is an over-approximation of ϕ (i.e., if there exists a domain D where ϕ is satisfiable, then $\mathrm{G\_A}(\phi, D_\downarrow)$ is also satisfiable.*

The following corollary is a direct consequence of Lemma 2.

Corollary 1. *Let sum be a summation in the form $Sum(S, p, val)$. If for every $s \in S$, $p(s)$ and $val(s)$ do not contain summations or quantifiers, then $\mathrm{GLB}(sum, D) = \mathrm{LUB}(sum, D) = Sum_D(S, p, val)$ for every domain D.*

1.3 The base case proof for Lemma 2 and 3

First, we consider the base case where the target summation is stratified at layer 0. The base cases are stated as Lemmas 4, 5 and 6.

Lemma 4 (Local correctness of GLB and LUB at layer 0). *Suppose $Sum(S, p, val)$, denoted sum^0, is a stratified summation at level 0 (see Def. 1) and $D_\downarrow$ is a domain. Let $\mathrm{LUB}(sum, D_\downarrow)$ and $\mathrm{GLB}(sum, D_\downarrow)$ be the local upper-bound and global lower-bound of sum^0, respectively. Let $sum^0_{D_\downarrow}$ be the under-approximation of sum^0 in $D_\downarrow$ ($sum^0_{D_\downarrow} = \sum^s_{S \subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)$). The following statement is true:*

$$\mathrm{GLB}(sum^0, D_\downarrow) \leq sum^0_{D_\downarrow} \leq \mathrm{LUB}(sum^0, D_\downarrow)$$

Proof. First, we prove the inequality $\mathrm{GLB}(sum^0, D_\downarrow) \leq sum^0_{D_\downarrow}$ by contradiction. Suppose $\mathrm{GLB}(sum^0, D_\downarrow) > sum^0_{D_\downarrow}$, then it is the case that

$$\begin{aligned}\sum\nolimits_{s \in S \subseteq D_\downarrow} ite(\neg s.ext \vee \mathrm{G\_A}(\neg p(s), D_\downarrow), 0, \mathrm{GLB}(val(s), D_\downarrow)) >\\ \sum\nolimits^s_{S \subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)\end{aligned}$$

Since sum^0 is stratified at level 0, $p(s)$ and $val(s)$ do not have summation in them (by Def. 1). Therefore, $\mathrm{G\_A}(\neg p(s), D_\downarrow) = \mathrm{G}(\neg p(s), D_\downarrow)$. Feng et al. [6] proved (Lemma 3) that $\mathrm{G}(\neg p(s), D_\downarrow)$ is an over-approximation of $\neg p(s)$ (i.e., $\neg p(s) \Rightarrow \mathrm{G}(\neg p(s), D_\downarrow)$), hence $\neg\mathrm{G}(\neg p(s), D_\downarrow) \Rightarrow p(s)$. Therefore, for every $s \in S$, if s contributes $\mathrm{GLB}(val(s), D_\downarrow)$ to $\mathrm{GLB}(sum^0, D_\downarrow)$, then it must contribute $val(s)$ to $sum^0_{D_\downarrow}$. Since $val(s)$ does not contain any summation (sum^0 is stratified at layer 0), it is easy to see that $\mathrm{GLB}(val(s), D_\downarrow) = val(s)$. Therefore,

$$\begin{aligned}\textstyle\sum_{s\in S\subseteq D_\downarrow} ite(\neg s.ext \vee \mathrm{G\_A}(\neg p(s), D_\downarrow), 0, \mathrm{GLB}(val(s), D_\downarrow)) \leq \\ \textstyle\sum^s_{S\subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)\end{aligned}$$

This reaches a contradiction.

Second, we prove the inequality $sum^0_{D_\downarrow} \leq \mathrm{LUB}(sum^0, D_\downarrow)$ by contradiction: Suppose $sum^0_{D_\downarrow} > \mathrm{LUB}(sum^0, D_\downarrow)$, then it is the case that

$$\begin{aligned}\textstyle\sum_{s\in S\subseteq D_\downarrow} ite(s.ext \wedge G(p(s), D_\downarrow), \mathrm{LUB}(val(s), D_\downarrow), 0) \leq \\ \textstyle\sum^s_{S\subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)\end{aligned}$$

Since sum^0 is stratified at level 0, the result of $p(s)$ and $val(s)$ will not have summation in them. Therefore, $\mathrm{G\_A}(\neg p(s), D_\downarrow) = \mathrm{G}(\neg p(s), D_\downarrow)$. Feng et al. proved (Lemma 3 in [6]) that $\mathrm{G}(p(s), D_\downarrow)$ is an over-approximation of $p(s)$ (i.e., $p(s) \Rightarrow \mathrm{G}(p(s), D_\downarrow)$). Therefore, for every $s \in S$, if s contributes $val(s)$ to $sum^0_{D_\downarrow}$, then it must contribute $\mathrm{LUB}(val(s), D_\downarrow)$ to $\mathrm{LUB}(sum^0, D_\downarrow)$. Since $val(s)$ does not contain any summation (sum^0 is stratified at layer 0), it is easy to see that $\mathrm{GLB}(val(s), D_\downarrow) = val(s)$. Therefore,

$$\begin{aligned}\textstyle\sum_{s\in S\subseteq D_\downarrow} ite(s.ext \wedge G(p(s), D_\downarrow), \mathrm{LUB}(val(s), D_\downarrow), 0) > \\ \textstyle\sum^s_{S\subseteq D_\downarrow} ite(s.ext \wedge p(s), val(s), 0)\end{aligned}$$

This is a contradiction. Therefore, both inequalities are proven.

Lemma 5 (Global correctness GLB at layer 0). *Suppose $Sum(S, p, val)$, denoted as sum^0, is a stratified summation at level 0 (see Def. 1) and $D_\downarrow$ is a domain. Let $\mathrm{GLB}(sum^0, D_\downarrow)$ be the global lower-bound of sum^0. For every domain such that $D \supseteq D_\downarrow$, let $sum_D = \sum^s_{S\supseteq D} ite(s.ext \wedge p(s), val(s), 0)$ be the under-approximation of sum^0 in D. The following relation always holds: $\mathrm{GLB}(sum^0, D_\downarrow) \leq sum^0_D$.*

Proof. Proof by contradiction: suppose that there exists a domain $D \supseteq D_\downarrow$ such that $\mathrm{GLB}(sum^0, D_\downarrow) > sum^{i+1}_D$. Since $D \supseteq D_\downarrow$, $sum^0_D \geq sum^0_{D_\downarrow}$. By Lemma 4, $\mathrm{GLB}(sum^0, D_\downarrow) \leq sum^0_{D_\downarrow}$, hence $\mathrm{GLB}(sum^0, D_\downarrow) \leq sum^0_D$. Contradiction.

Before moving on to the inductive step, we establish an important lemma for the function G_A since it is used in the definition of LUB and GLB, and behaves differently from G when the input formula contains summations.

Lemma 6. *Suppose ϕ^0 is a FOL^{*+} formula where every summation in ϕ^0 is stratified at layer 0 (see Def. 1), and $D_\downarrow$ is a domain. The grounded formula*

$\text{G\_A}(sum^0, D_\downarrow)$ *(Alg. 1) is an over-approximation of* ϕ^0 *(i.e., if* ϕ^0 *is satisfiable, then* $\text{G\_A}(\phi^0, D_\downarrow)$ *is satisfiable).*

Proof. If ϕ^0 does not contain any summation, then $\text{G\_A}(\phi^0, D_\downarrow) = \text{G}(\phi^0, D_\downarrow)$, and Feng et al. [6] proved that $\text{G}(\phi^0, D_\downarrow)$ is an over-approximation of ϕ^0. If ϕ^0 contains a summation sum^0, then it is encoded as a fresh integer variable i (L: 12 of Alg. 1) subject to the constraint req_{sum} (Def. 6). It suffices to show that the range of i, $[\text{GLB}(sum^0, D_\downarrow), \infty)$ includes the possible value of sum^0_D for all $D \supseteq D_\downarrow$. By Lemma 5, $sum^0_D \geq \text{GLB}(sum^0, D)$, and thus sum^0_D is in $[\text{GLB}(sum^0, D_\downarrow),)$. Therefore, $\text{G\_A}(\phi^0, D_\downarrow)$ is an over-approximation of ϕ^0.

1.4 The inductive step for Lemma 2 and 3

Now we prove the inductive step. First, we establish the inductive hypothesis.

Hypothesis 1 (Inductive hypothesis of GLB) *Let a domain* $D_\downarrow$ *and a summation* sum^i *stratified at layer* i *be given. Then for every domain* $D \supseteq D_\downarrow$, $\text{GLB}(sum^i, D_\downarrow) \leq sum^i_D$*, where* sum^i_D *is the under-approximation of* sum^i *in domain* D*.*

Hypothesis 2 (Inductive hypothesis of LUB) *Let a domain* $D_\downarrow$*, and a summation* sum^i *stratified at layer* i *be given. Then* $\text{LUB}(sum^i, D_\downarrow) \geq sum^i_{D_\downarrow}$ *where* $sum^i_{D_\downarrow}$ *is the under-approximation of* sum^i *in* $D_\downarrow$*.*

Hypothesis 3 (Inductive hypothesis of G_A) *Given a domain* $D_\downarrow$*, and an* FOL^{*+} *formula* ϕ^i *whose summations are stratified at layer* i*,* $\text{G\_A}(\phi^i, D_\downarrow)$ *is an over-approximation of* ϕ^i*.*

We now prove the inductive lemmas by assuming the above inductive hypotheses.

Lemma 7 (Inductive local correctness of GLB and LUB). *Suppose* $Sum(S, p, val)$*, denoted as* sum^{i+1}*, is a stratified summation at level* $i+1$ *(see Def.1) and* $D_\downarrow$ *is a domain. Let* $sum^{i+1}_{D_\downarrow}$ *be the under-approximation of* sum^{i+1} *in* $D_\downarrow$*. If Hypotheses 1, 2 and 3 holds, then*

$$\text{GLB}(sum^{i+1}, D_\downarrow) \leq sum^{i+1}_{D_\downarrow} \leq \text{LUB}(sum^{i+1}, D_\downarrow)$$

Proof. First, we prove the inequality $\text{GLB}(sum^{i+1}, D_\downarrow) \leq sum^{i+1}_{D_\downarrow}$. By Def. 4, $\text{GLB}(sum^{i+1}, D_\downarrow) = \sum_{s \in S \subseteq D_\downarrow} ite(\neg s.ext \vee \text{G\_A}(\neg p(s), D_\downarrow), 0, \text{GLB}(val(s), D_\downarrow))$. Since sum^{i+1} is stratified at layer $i+1$, by Def. 1, $\neg p(s)$ only contains summations that are stratified at layer i or below. Therefore, by Hypothesis 3, $\text{G\_A}(\neg p(s), D_\downarrow)$ is an over-approximation of $\neg p(a)$, and hence if s contributes $\text{GLB}(val(s), D_\downarrow)$ to $\text{GLB}(sum^{i+1}, D_\downarrow)$, then it must also contribute $val(s)$ to $sum^{i+1}_{D_\downarrow}$. Therefore, we can show $\text{GLB}(val(s), D_\downarrow) \leq val(s)$ to prove $\text{GLB}(sum^{i+1}, D_\downarrow) \leq sum^{i+1}_{D_\downarrow}$. Now consider $\text{GLB}(val(s), D_\downarrow)$, By Def. 4, there are five cases:

(1) if $val(s)$ is a constant or a variable, then $\mathrm{GLB}(val(s), D_\downarrow) = val(s)$. $\square$
(2) if $val(s) = -t$ then we create an obligation showing $\mathrm{LUB}(val(s), D_\downarrow) \geq val(s)$. Without loss of generality (WLOG), we can assume that t does not contain any other negation operator, '$-$', since otherwise the negation on t can be pushed in.
(3) if $val(s) = t_1 + t_2$, then we create an obligation to show $\mathrm{GLB}(val(t_1), D_\downarrow) \leq t_1 \wedge \mathrm{GLB}(val(t_2), D_\downarrow) \leq t_2$.
(4) if $val(s) = c \times t$, WLOG, we can assume $c > 0$ (if $c < 0$, can rewrite it as $-(-c \times t)$), then we create an obligation to show $\mathrm{GLB}(t, D_\downarrow) \leq t$.
(5) if $val(s)$ is a summation, then by Hypothesis 1, $\mathrm{GLB}(val(s), D_\downarrow) \leq val(s)$.

Cases (1) and (5) are terminal and have already been proven. Case (2) is a special terminal case where we need to prove $\mathrm{LUB}(t, D_\downarrow) \geq t$ for some negation-free term t. Cases (3) and (4) are non-terminal cases which generate a set of new proof obligations. Since $val(s)$ is a finite expression, Cases (3) and (4) will reach one of the terminal cases (1), (2) or (5).

To prove Case (2): $\mathrm{LUB}(t, D_\downarrow) \geq t$ for some negation-free term t, we consider the definition of LUB (Def. 5) which consists of five cases.

(i) t is a constant or a variable. Then $\mathrm{GLB}(t, D_\downarrow) = t$. $\square$
(ii) $t = -t'$. However, since we assumed that t does not contain negation, this case is unreachable.
(iii) $val(s) = t_1 + t_2$. Then we create an obligation to show $\mathrm{LUB}(val(t_1), D_\downarrow) \geq t_1 \wedge \mathrm{LUB}(val(t_2), D_\downarrow) \geq t_2$.
(iv) if $val(s) = c \times t$, WLOG, we can assume $c > 0$ (if $c < 0$, can rewrite it as $-(-c \times t)$). Then we create an obligation to show $\mathrm{LUB}(t, D_\downarrow) \geq t$.
(v) if $val(s)$ is a summation, then by Hypothesis 2, $\mathrm{LUB}(val(s), D_\downarrow) \geq val(s)$. $\square$

Cases (i) and (v) are terminal and Case (ii) is unreachable. Cases (iii) and (iv) are non-terminal, which generate more proof obligations. Given that t is a finite expression, by recursively analyzing the proof obligations, these cases will eventually reach either Case (i) or Case (v). This proves Case (2). $\square$

Combining Case (2) with Cases (1) and (3)-(5), we now have proven $\mathrm{GLB}(sum^{i+1}, D_\downarrow) \leq sum^{i+1}_{D_\downarrow}$. Combining this with the proven fact that $\mathrm{G\_A}(\neg p(s), D_\downarrow) \Rightarrow \neg p(s)$, we obtain the first inequality $\mathrm{GLB}(sum^{i+1}, D_\downarrow) \leq sum^{i+1}_{D_\downarrow}$. $\square$

The proof for the second inequality, $sum^{i+1}_{D_\downarrow} \leq \mathrm{LUB}(sum^{i+1}, D_\downarrow)$, is identical to the proof of the first inequality with a few exceptions: (1) we prove that $\mathrm{G\_A}(p(s), D_\downarrow) \Rightarrow p(s)$ given Hypothesis 3; (2) we prove $\mathrm{LUB}(val(s), D_\downarrow) \geq val(s)$ by case analysis following the definition of LUB (Def. 5), and (3) we prove $\mathrm{GLB}(t, D_\downarrow) \leq t$ for any negation-free term t. Due to the similarity, the detailed proof is omitted.

Lemma 8 (Inductive global correctness GLB). *Suppose $Sum(S, p, val)$, denoted as sum^{i+1}, is a stratified summation at level $i+1$ (see Def. 1) and $D_\downarrow$ is a domain. Let $\mathrm{GLB}(sum^0, D_\downarrow)$ be the global lower-bound of sum^{i+1}. For every domain $D \supseteq D_\downarrow$, let $sum_D = \sum^s_{S \supseteq D} ite(s.ext \wedge p(s), val(s), 0)$ be the* *under-approximation of sum^{i+1} in D. If Hypotheses 1, 2 and 3 hold, then the following relation always holds:* $\mathrm{GLB}(sum^{i+1}, D_{\downarrow}) \leq sum_D^{i+1}$.

Proof. Proof by contradiction: suppose there exists a domain $D \supseteq D_{\downarrow}$ such that $\mathrm{GLB}(sum^{i+1}, D_{\downarrow}) > sum_D^{i+1}$. Since $D \supseteq D_{\downarrow}$, $sum_D^{i+1} \geq sum_{D_{\downarrow}}^{i+1}$. By Lemma 7, $\mathrm{GLB}(sum^{i+1}, D_{\downarrow}) \leq sum_{D_{\downarrow}}^{i+1}$, hence $\mathrm{GLB}(sum^0, D_{\downarrow}) \leq sum_D^{i+1}$. □

The following lemma is an inductive generalization to Lemma 6, which is necessary for induction for the Hypothesis 3.

Lemma 9. *Suppose ϕ^{i+1} is a FOL^{*+} formula where every summation in ϕ^{i+1} is stratified at layer $i+1$ (see Def. 1), and $D_{\downarrow}$ is a domain. If Hypotheses 1, 2 and 3 hold, then the grounded formula* $\mathrm{G\_A}(sum^{i+1}, D_{\downarrow})$ *(Alg. 1) is an over-approximation of ϕ^{i+1} (i.e., if ϕ^{i+1} is satisfiable, then* $\mathrm{G\_A}(\phi^{i+1}, D_{\downarrow})$ *is also satisfiable).*

Proof. Every summation $\sum^{i+1}$ in ϕ^{i+1} is encoded as a fresh integer variable i (L: 12 of Alg. 1) subject to the constraint req_{sum} (Def. 6). It is sufficient to show that the range of i, $[\mathrm{GLB}(sum^{i+1}, D_{\downarrow}),)$ includes the possible value of sum_D^{i+1} for all $D \supseteq D_{\downarrow}$. By Lemma 8, $sum_D^{i+1} \geq \mathrm{GLB}(sum^{i+1}, D)$, and thus sum_D^{i+1} is in $[\ \mathrm{GLB}(sum^{i+1}, D_{\downarrow}),)$. Therefore, $\mathrm{G\_A}(\phi^{i+1}, D_{\downarrow})$ is an over-approximation of ϕ^{i+1}.

1.5 Proving correctness of GLB and LUB

Given the base cases Lemmas 4, 5 and 6, and the inductive step, Lemmas 7, 8 and 9, the proofs of correctness of GLB, LUB (Lemma 1) and G_A (Lemma 2) are the direct results of induction.

2 Correctness proof of Search-A

In this section, we first recall the algorithm Search-A (Alg. 2) and then prove the correctness (Thm. 1), termination (Thm. 2) and solution optimality (Thm. 3) of Search-A. Since Search-A is an extension of LEGOS, the proof focuses on the delta between the two: aggregation support, algorithmic enhancements and optional optimizations.

Before proving the main theorems, we first prove that enabling the optimization boundary case reduction (BCR) does not change the outcome of the algorithm. Since BCR applies to G_A, we prove the following lemma.

Lemma 10 (Correctness of Boundary Case Reduction (BCR)). *Let an FOL^{*+} formula ϕ and a domain $D_{\downarrow}$ be given. The grounded formulas* $\mathrm{G\_A}(\phi, D_{\downarrow}, bcr)$ *and* $\mathrm{G\_A}(\phi, D_{\downarrow}, \neg bcr)$ *are either both satisfiable or both unsatisfiable.*

Proof. Let $\phi_g \leftarrow \mathrm{G\_A}(\phi, D_{\downarrow}, \neg bcr)$ and $\phi'_g \leftarrow \mathrm{G\_A}(\phi, D_{\downarrow}, bcr)$. We prove Lemma 10 by contradiction: we assume ϕ_g is satisfiable and ϕ'_g is not. Then the unsatisfiability of ϕ'_g must be due to the BCR constraints. Consider the BCR constraint

$$\forall o : r \cdot\ o.time \leq o_f.time\ \vee\ o.time \geq o_l.time \Rightarrow \neg p(o)$$

Algorithm 2 SEARCH-A: search for a bounded (by n) solution to $\neg P \bigwedge \Psi$.

Input a FOL*+ $\neg P$ and a set of FOL*+ requirements $\Psi = \{\psi_1, \psi_2, ...\}$.
Optional Input bcr, $t_{res} = \infty$ for boundary case reduction and restart
Optional Input vb, the volume bound of the counterexample.
Output a counterexample σ, `UNSAT` or bounded-`UNSAT`.

1: $\Psi_\downarrow \leftarrow \emptyset$, $D_\downarrow \leftarrow \emptyset$
2: $\Psi_\downarrow \leftarrow \Psi_\downarrow \cup req_{sum}$
3: $relaxed \leftarrow \top$
4: **while** $\top$ **do**
5:     **if** $iters > t_{res}$ **then** $\Psi_\downarrow \leftarrow \{\}, t_{res} \leftarrow t_{res} \times 1.25$, $iters \leftarrow 0$
6:     $\phi \leftarrow \neg P \wedge \Psi_\downarrow$
7:     $\phi_g \leftarrow$ G$-A(\phi, D_\downarrow, bcr)$
8:     $\phi_g^\perp \leftarrow \phi_g \wedge Inc(\phi_g, D_\downarrow)$
9:     **if** SOLVE(ϕ_g) = `UNSAT` **then**
10:         **return** `UNSAT`
11:     **else**
12:         $\sigma \leftarrow$ SOLVE($\phi_g^\perp$)
13:         **if** σ = `UNSAT` **then**
14:             $\sigma_{min} \leftarrow$ MINIMIZE(ϕ_g, $relaxed$)
15:             $D_\downarrow$ += $\{act \mid act \in \sigma_{min}\}$
16:             **if** $vol(\sigma_{min}) > vb$ **then**
17:                 **if** $relaxed$ **then** $relaxed \leftarrow \perp$
18:                 **else return** bounded-`UNSAT`
19:         **else**
20:             **if** $\sigma \models \Psi$ **then**
21:                 **if** $relaxed$ **then**
22:                     $relaxed \leftarrow \perp$
23:                 **else**
24:                     **return** σ
25:             **else** $\Psi_\downarrow \leftarrow \Psi_\downarrow \cup \{\psi | \sigma \not\models \psi\}$

for relational objects o_l and o_f created during grounding existential quantification (L: 2 of Alg. 1). However, the constraint is a tautology since every relational object has a time attribute, and time is ordered. Therefore, there always exists a first and last time where some relation holds in a finite time domain. Thus, adding the BCR constraint does not make ϕ_g' UNSAT if ϕ_g is satisfiable.

Lemma 10 ensures that the BCR does not change the outcome of SEARCH-A. Therefore, we safely ignore the impact of BCR when proving the correctness of SEARCH-A in Thm. 1.

Theorem 1 (Correctness of Search-A). *Let an FOL^{*+} formula $\neg P$, a set of FOL^{*+} requirements Ψ and a volume bound $vb \in \mathcal{N}$ be given. Then*

(1) if SEARCH-*A*$(\neg P, \Psi, vb)$ *returns a solution σ then σ is a satisfying solution to $\neg P \wedge \Psi$ and $vol(\sigma) \leq vb$;*
(2) if SEARCH-*A*$(\neg P, \Psi, vb)$ *returns* ***UNSAT*** *then $\neg P \wedge \Psi$ is unsatisfiable;*
(3) if SEARCH-*A*$(\neg P, \Psi, vb)$ *returns bounded-****UNSAT*** *then there is no solution to $\neg P \wedge \Psi$ whose volume is not greater than vb.*

Proof Sketch. The proof considers the three cases of searcha$(\neg P, \Psi, vb)$.

Consider Case (1), where searcha$(\neg P, \Psi, vb)$ returns a solution σ on L: 24. We first prove that (1a) $\sigma \models \neg P \wedge \Psi$, and then (1b) $vol(\sigma) \leq vb$.

Proof of (1a): Since σ is returned by SEARCH-*A, the following conditions hold: $\sigma \models \Psi$ and $\sigma \models \phi_g^\perp$ where $\phi_g^\perp$ is the under-approximating of $\neg P \wedge \Psi_\downarrow$ in the under-approximated domain $D_\downarrow$. Since $\sigma \models \phi_g^\perp$, by Lemma 1, $\sigma \models \neg P \wedge \Psi_\downarrow$ and $\sigma \models \neg P$. Therefore, combined with the fact $\sigma \models \Psi$, we get $\sigma \models \neg P \wedge \Psi$.*

To prove (1b), we consider the fact SEARCH-*A is not in the relaxed domain expansion mode (L: 21). Therefore, the solution (σ_{min}) to the over-approximation query ϕ_g' computed at L: 14 is a minimum solution. We can then prove (1b) the same way as Thm. 4 in [6] by showing that $vol(\sigma) \geq vol(\sigma_{min}')$ where σ_{min}' is the* *minimum solution to an over-approximation ϕ'_g in a previous non-relaxed iteration of Alg. 2. Therefore, if $vol(\sigma) > vb$, then $vol(\sigma'_{min}) > vb$, and* SEARCH-*A would have returned bounded-*`UNSAT` *(L: 18) instead of σ.*

Combining Cases (1a) and (1b), we proved Case (1). □

Consider Case (2) where SEARCH-*A*$(\neg P, \Psi, vb)$ *returns* `UNSAT`*(at L: 10). Then the grounded over-approximation ϕ^g is unsatisfiable (L: 9). By Lemma 2 of [6], $\neg P \wedge \Psi_\downarrow$ is also unsatisfiable. Since $\Psi_\downarrow \subseteq \Psi$, $\neg P \wedge \Psi_\downarrow$ is unsatisfiable as well.* □

Consider case (3): if SEARCH-*A*$(\neg P, \Psi, vb)$ *returns bounded-*`UNSAT`*. When bounded-*`UNSAT` *is returned, the relaxed domain expansion must have been turned off (L: 17, and the minimum solution σ_{min} to the query ϕ_g has the volume greater than vb. By Lemma 2 of [6], ϕ_g is an over-approximation of $\neg P \wedge \Psi_\downarrow$. Moreover, if σ is a solution to $\neg P \wedge \Psi_\downarrow$, then $vol(\sigma) \geq \sigma_{min}$ (Corollary 1 of [6]). Therefore, $vol(\sigma) \geq vb$.* □

Theorem 2 (Termination of Search-A). *Let an FOL^{*+} formula $\neg P$, a set of FOL^{*+} requirements Ψ and a volume bound $vb \in \mathbb{N}$ be given. If $vb \neq \infty$, then* SEARCH-*A terminates on the input $(\neg P, \Psi, vb)$.*

Proof Sketch. The proof is similar to the termination proof of LEGOS (Thm. 3 of [6]). Since LEGOS does not use algorithmic enhancements or optimizations of SEARCH-A, we now prove that the enhancements and optimizations do not affect termination.

1. *BCR*: According to Lemma 10, BCR does not affect the satisfiability of the grounded formula and therefore does not affect termination. □
2. *Restart*: since the restart interval threshold t_{res} increases every time a restart occurs (L: 5), t_{res} would increase indefinitely. Therefore, if SEARCH-A terminates within a finite interval, then t_{res} would eventually reach the interval and terminate. □
3. *Relaxed domain expansion*: relaxed domain expansion would expand the domain and increase the volume of the solution σ_{min} to the over-approximation query (Corollary 1 of [6]). As SEARCH-A executes, eventually, $vol(\sigma_{min})$ would exceed vb, and relaxed domain expansion would be turned off (L: 16). Therefore, relaxed domain expansion does not affect termination. □
4. *Incremental Solving*: the encoding for incremental solving does not change the semantics of the grounded formula for both the over-approximation (L: 7) and the under-approximation (L: 8), and hence does not affect termination. □

Since the enhancements and optimizations preserve the termination guarantees, SEARCH-A terminates if $vb \neq \infty$. □

Theorem 3 (Solution Optimality of Search-A). *Let an FOL^{*+} formula $\neg P$, a set of FOL^{*+} requirements Ψ and a volume bound $vb \in \mathcal{N}$ be given. If $(\neg P, \Psi, vb)$ returns a solution σ, then there is no solution to $\neg P \wedge \Psi$ whose volume is smaller than $vol(\sigma)$.*

Proof. Given that termination of LEGOS has been proven elsewhere (Thm. 4 of [6]), we focus on showing that the SEARCH-A enhancements and optimizations do not affect termination.

1. *BCR*: According to Lemma 10, BCR does not affect the satisfiability of the grounded formula and therefore does not affect the solution optimality. □
2. *Restart*: if "restart" is enabled in SEARCH-A, then the under-approximated requirements in $\Psi_{\downarrow}$ might be removed from time to time. Let $\Psi_{\downarrow}*$ be the set of under-approximated requirements if restart is *not* enabled. Clearly, $\Psi_{\downarrow} \subseteq \Psi_{\downarrow}*$. If σ is an optimal solution to $\neg P \wedge \bigwedge_{\psi \in \Psi_{\downarrow}}$ and $\sigma \models \bigwedge_{\psi \in \Psi}$ where $\Psi \supseteq \Psi_{\downarrow}$, then it must also be the optimal solution to $\sigma \models \neg P \wedge \bigwedge_{\psi \in \Psi_{\downarrow}*}$. Therefore, restart does not affect the solution optimality. □
3. *Relaxed domain expansion*: if SEARCH-A returns a solution, then relaxed domain expansion must have been turned off (L: 21). Therefore, relaxed domain expansion does not affect the solution optimality. □
4. *Incremental Solving*: the encoding for incremental solving does not change the semantics of the grounded formula for both the over-approximation (L: 7) and the under-approximation (L: 8), and hence does not affect the solution optimality. □

3 Support for *Count*, *Max* and *Min*

In this section, we present LEGOS-A's support for aggregation function *Count*, *Max* and *Min*.

We have been focusing on the support for the aggregation function *Sum*. Other aggregation functions, *Count*, *Max* and *Min*, are supported analogously in FOL* using over and under-approximation. Similar to the support of *Sum*, the aggregation's under-approximation is bounded by the domain, and its over-approximation is bounded by its global-lower bound (GLB) and local upper bound (LUB). We now present the support for other aggregation functions.

Count. *Count* in FOL* has the signature $Count(S, p)$, where S is a class and p is a predicate. It is equivalent to $Sum(S, p, One())$, where $One()$ is a constant function returning one. The support for *Count* is realized through the support of *Sum*.

Max. *Max* in FOL* has the signature $Max(S, p, val)$, where S is a class, p is a predicate, and val is a numerical function. We support *Max* using over- and under-approximation. In a domain D, the under-approximation is $\max(\{ite(s.ext \wedge p(s), val(s), -\infty) \mid s \in S \subseteq D\})$. The over-approximation in a domain D is a fresh integer variable i under the constraint req_{max}: (1) i must be no less than its global lower-bound (GLB_D^{max}) and (2) if i is greater than its local upper-bound (LUB_D^{max}), then there exists a relational object s of class S such that $p(s) \wedge val(s) = i$ where

(i) $\text{GLB}_D^{max} = \max(\{ite(\neg s.ext \vee G\_A(\neg p(s), D), -\infty, GLB(val(s), D)) \mid s : S \subseteq D\})$

(ii) $\text{LUB}_D^{max} = \max(\{ite(s.ext \wedge G\_A(p(s), D), GLB(val(s), D), -\infty,) \mid s : S \subseteq D\})$.

P_1: *"If a banking transaction has already been executed with an amount higher than the payer's usual* ***total daily spending*** *(the sum of transaction amounts) in the last 7 days, and if the payer requested a refund, the payment service shall refund the full payment amount within ten business days of receiving the refund request."* ($C1$ $\wedge$ $C2$ $\rightarrow$ O) **(a)**

total daily spending without aggregation: **(b)**
Step 1 Add aggregation index to transaction relation $Trans(u, time, x) \rightarrow Trans_a(u, time, x, i)$
with bijective mapping constraints between $Trans$ and $Trans_a$:
(1) $\forall o : Trans_a \cdot \exists o' : Trans \cdot o.u = o'.u \wedge o.x = o'.x \wedge o.time = o'.time$
(2) $\forall o, o' : Trans_a \cdot o.i = o'.i \wedge o.u = o'.u \wedge o.time = o'.time \Rightarrow (o.x = o'.x)$
Step 2 Introduce relations $Daily_a(u, x, d, i)$ where x is sum of all transactions at day d for $index \leq i$
Step 3: Add aggregation constraints for base case $(a1)$, aggregation step $(a2)$, and the final value $(a3)$:
$(a1)$: $\forall o : Daily_a \cdot o.i = 0 \Rightarrow o.x = 0$
$(a2)$: $\forall o, o' : Daily_a \cdot (o.u = o'.u \wedge o.d = o'.d \wedge o.i + 1 = o'.i) \iff \exists t : Trans_a \cdot t.u = o.u \wedge t.d = o.d \wedge t.x = (o'.x - o.x)$
$(a3)$: $\forall o : Daily \cdot \exists o' : Daily_a \cdot o.u = o'.u \wedge o.d = o'.d \wedge o.x = o'.x \wedge \forall o'' : Daily_a \cdot (o''.u = o'.u \wedge o''.d = o'.d) \Rightarrow o'.i \geq o''.i$
Step 4: Return value x in relation $Daily(u, d, x)$

total daily spending with aggregation: **(c)**
$Sum\ (Trans,\ \lambda trans : trans.user = u \wedge trans.time = d,\ \lambda trans : trans.x)$

Fig. 1. Several FOL* properties inspired from the European directive on payment services in the internal market [5]: **(a)** A legal property P_1; **total daily spending** of a user u on day d defined **(b)** without aggregation; **(c)** with aggregation.

Intuitively, we can obtain the LUB of $Max(S, p, val)$ by over-approximating $p(s)$ as $\mathrm{G\_A}(p(s), D_\downarrow)$ and over-approximating $val(s)$ as $\mathrm{LUB}(val(s), D)$. We can obtain the GLB by over-approximating $\neg p(s)$ (and switching *ite*'s "then" and "else" branches), and under-approximating $val(s)$ as $\mathrm{GLB}(val(s), D)$. Note that the shapes of the GLB, LUB, and over and under-approximations for *Max* are identical to the ones for *Sum*. The only difference between them are value aggregation functions ($\sum \rightarrow \max$) and the default value ($0 \rightarrow -\infty$).

Min. *Min* in FOL* has the signature $Min(S, p, val)$, and it is equivalent to $-Max(S, p, -val)$. Therefore, the support for *Min* is realized though the support for *Max*.

To summarize, we proposed a general method to support aggregation in FOL* with over-and-under approximations. The 'secrete sauce' are the numerical approximation functions (LUB and GLB) that over/under-approximate (upper/lower bound) the value of numerical terms in FOL*. One can use our method to support new aggregation functions by defining their LUB and GLB.

4 Illustration of over-and under-approximating of summations

Consider the property P_1 described in Fig. 1(a) from [1]. P_1 contains the condition $C1$ that compares the size of user transactions with their daily spending of the last 7 days. The daily transaction amount made by a user u at day t is expressed as a summation *sum* in the form:

$$Sum(Trans,\ \lambda tr' : tr'.u = u \wedge tr'.\mathrm{time} = t,\ \lambda tr' :\ tr'.x).$$

Suppose $D_\downarrow$ is domain consistent of three relational objects of the class *Trans* $\{tr_1, tr_2, tr_3\}$.

- The under-approximation sum_D is
  $ite(tr_1.u = u \wedge tr_1.\mathrm{time} = t, tr_1.x, 0) + ite(tr_2.u = u \wedge tr_2.\mathrm{time} = t, tr_2.x, 0)$
  $+ite(tr_3.u = u \wedge tr_3.\mathrm{time} = t, tr_3.x, 0)$.
- The over-approximation of sum is a fresh integer value i such that
  $i \geq \mathrm{GLB}(sum, D_{\downarrow})$ and $i \geq \mathrm{GLB}(sum, D_{\downarrow}) \implies \exists tr : Trans \cdot i = tr.x+$
  $Sum(Trans,\ \lambda tr' : tr'.u = u \wedge tr'.\mathrm{time} = t \wedge tr \not\equiv tr',\ \lambda tr' :\ tr'.x)$.

Since sum is stratified at layer-0, $\mathrm{GLB}(sum, D_{\downarrow}) = \mathrm{GLB}(sum, D_{\downarrow}) = sum_{D_{\downarrow}}$ (Cor. 1). Now consider sum' in the form of

$$Sum(Trans,\ \lambda tr.x \geq sum,\ \lambda tr' :\ tr'.x).$$

- The under-approximation sum'_D is
  $ite(tr_1.x \geq sum_{D_{\downarrow}}, tr_1.x, 0) + ite(tr_2.u \geq sum_{D_{\downarrow}}, tr_2.x, 0)+$
  $ite(tr_3.u \geq sum_{D_{\downarrow}}, tr_3.x, 0)$.
- The over-approximation of sum' is a fresh integer value i' such that
  $i' \geq \mathrm{GLB}(sum', D_{\downarrow})$ and $i' \geq \mathrm{GLB}(sum', D_{\downarrow}) \implies \exists tr : Trans \cdot i' = tr.x+$
  $Sum(Trans,\ \lambda tr' : tr'.x \geq sum \wedge tr \not\equiv tr',\ \lambda tr' :\ tr'.x)$.

Let i be the over-approximation of sum defined above:

- The global lower-bound $\mathrm{GLB}(sum', D_{\downarrow})$ is
  $ite(tr_1.x < i, 0, tr_1.x) + ite(tr_2.x < i, 0, tr_2.x) + ite(tr_3.x < i, 0, tr_3.x)$
- The local upper-bound $\mathrm{LUB}(sum', D_{\downarrow})$ is
  $ite(tr_1.x \geq i, tr_1.x, 0) + ite(tr_2.x \geq i, tr_2.x, 0) + ite(tr_3.x \geq i, tr_3.x, 0)$.

5 Illustration of running IBSC

Consider the banking system that supports transfer between users. A user u can transfer x units of money to a different user v by transfer: $Trans(u, v, x)$.

The bank establishes the following requirements: (R1) A user can transfer at most 5000 units of money every day. (R2) If a single transfer is for an amount greater than 1000 units, then the user who initiated this transfer must have transferred out 3000 units on the previous day. (R3) The banking system only accepts refund requests for transfers under 1000 units of money. Inspired by the legal property P_1 in Fig. 1, we introduce a new property P_2 adapted for the banking industry: For any transfer with amount more than 1000 units, the transfer amount should not be higher than the usual transferer's total daily spending over the last 7 days. We formalize P_2's negation as: $\neg P_2 = \exists tr : Trans \cdot (tr.x > 3000 \wedge (\forall t : Time \cdot trans.time - 7 \leq t < trans.time \Rightarrow Sum(Trans,\ \lambda tr' : tr'.u = tr.u \wedge \mathrm{tr}.time = t,\ \lambda tr' :\ tr'.x < tr.x)))$. We now illustrate Search-A. For each iteration, we denote ϕ_g and $\phi_g^{\perp}$ as the over- and under-approximation queries computed on L: 7 and L: 8 of Alg. 2, respectively. We write $sum^{\uparrow}$ and $sum^{\downarrow}$ as the over- and under-approximation of sum, respectively. Note that for the sum $Sum(Trans,\ \lambda tr' : tr'.u = tr.u \wedge \mathrm{tr}.time = t,\ \lambda tr' :\ tr'.x < tr.x)))$, its GLB (Def. 4), LUB (Def. 5) and $sum^{\downarrow}$ always have the same value in any domain

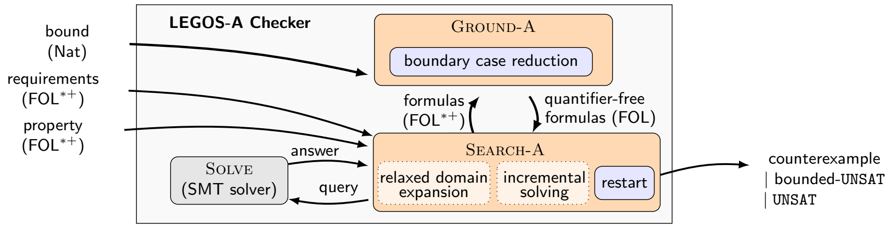

Fig. 2. Architecture of LEGOS-A.

because the filtering function and value function do not return expressions with sums (Cor. 1). Therefore, we use $sum^{\downarrow}$ to represent all of them.
Iteration 1. $D_{\downarrow} = \emptyset$. The over-approximation ϕ_g introduces a relational object tr^1 due to $\neg p$ where $tr^1.x > 1000$. ϕ_g is satisfiable, but $\phi_g^{\perp}$ is not (solved by SMT solver Z3 [3] using techniques from [4]) since $tr_1 \notin D_{\downarrow}$. Therefore, $D_{\downarrow}$ is expanded by adding tr^1.
Iteration 2. $D_{\downarrow} = \{tr^1\}$. ϕ_g introduces a new summation $sum_1^{\uparrow} = 3000$ due to (R2) because the sum of transfer by $tr^1.u$ at $tr^1.time-1$ is 3000. ϕ_g is satisfiable, but $\phi_g^{\perp}$ is UNSAT since $0 = sum_1^{\downarrow} \neq sum_1^{\uparrow}$. Therefore, $D_{\downarrow}$ is expanded by adding sum_1 as a summation object.
Iteration 3. $D_{\downarrow} = \{tr^1, sum_1\}$. ϕ_g introduces a relational object tr^2 and a new sum sum_2 due to req_{sum} where $tr^2.u = tr^1.u$, $tr^2.time = tr^1.time - 1$ and $sum_2^{\uparrow} + tr_2.x = sum_1^{\uparrow}$. ϕ_g is satisfiable, but $\phi_g^{\perp}$ is not since $tr_2 \notin D_{\downarrow}$. We assume that $D_{\downarrow}$ is expanded by tr^2.
Iteration 4. $D_{\downarrow} = \{tr^1, sum_1, tr^2\}$. ϕ_g introduces a new summation $sum_3^{\uparrow} = 3000$ because R2 describes the sum of transfer amounts by $tr^2.u$ at $tr^2.time-1$ when $tr^2.x > 1000$. ϕ_g is satisfiable, but $\phi_g^{\perp}$ is not since $sum_3 \downarrow = 0$ or $sum_2^{\downarrow} + tr^2.x < 3000$. Suppose that $D_{\downarrow}$ is expanded by adding sum_2 to $D_{\downarrow}$.
Iterations 5 - 10. $D_{\downarrow} = \{tr^1, sum_1, tr^2, sum_2\}$. Suppose that the domain expansion follows a process similar to that of iterations 3-4. At the end of iteration 10, there are 4 relational objects, tr^2, tr^3, tr^4, tr^5, all of which occurred one day before $tr^1.time$ and are initiated by $tr^1.u$.
The final iteration. $D_{\downarrow} = \{tr^1, sum_1, tr^2, sum_2, tr^3, sum_4, \ldots, tr^5\}$. $\phi_g^{\perp}$ becomes satisfiable with the solution $tr^1.x = 5000$, $tr^1.u = 0$, $tr^1.time = 1$, $tr^2.x = tr^3.x = tr^4.x = tr^5.x = 800$, $tr^2.u = tr^3.u = tr^4.u = tr^5.u = 0$ and $tr^2.time = tr^3.time = tr^4.time = tr^5.time = 0$. Therefore, $\neg P_2$ is satisfied implying that property P_2 might be violated.

On the other hand, if R1 is tightened to restrict a user from transferring more than 3,000 units of money per day, then $\neg P_2 \wedge R1 \wedge R2$ is UNSAT.

5.1 Tool support

We implemented our approach for bounded FOL^{*+} satisfiability checking in a tool LEGOS-A – see its architecture in Fig. 2. Built on top of LEGOS using Python and PySMT [7] library, LEGOS-A extends components Search and Ground into Search-A and Ground-A, respectively. Specifically, component

SEARCH-A includes two enhancements (shown as boxes with dotted lines), *incremental solving* and the *relaxed domain expansion*. Additionally, SEARCH-A includes the optional optimization *restart*. Component GROUND-A includes the optional optimization *boundary case reduction*. Other components, coloured grey in Fig. 2, are directly reused from LEGOS [6] and Z3 [3]. LEGOS-A includes a web interface implemented in JavaScript (see [2]).

References

1. Anonymous: ASE 2023 submission: LEGOS: Legal Compliance Verifier via Satisfiability Checking (2023)
2. Anonymous: The web interface of LEGOS (2023), `https://legos1.herokuapp.com`
3. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
4. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4144, pp. 81–94. Springer (2006). https://doi.org/10.1007/11817963_11, `https://doi.org/10.1007/11817963_11`
5. EU payment services directive: Directive 2007/64/EC of the European Parliament and of the Council of 13 October 2000 establishing a framework for payment services in the internal market amending (2007), `https://eur-lex.europa.eu/eli/dir/2007/64/2009-12-07`
6. Feng, N., Marsso, L., Sabetzadeh, M., Chechik, M.: Early Verification of Legal Compliance via Bounded Satisfiability Checking. In: Proceedings of the 34th International Conference on Computer-Aided Verification (CAV'23), Paris, France. Lecture Notes in Computer Science, Springer (2023)
7. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT workshop. vol. 2015 (2015)
8. Leivant, D.: Stratified functional programs and computational complexity. In: Deusen, M.S.V., Lang, B. (eds.) Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993. pp. 325–333. ACM Press (1993). https://doi.org/10.1145/158511.158659, `https://doi.org/10.1145/158511.158659`