Combining Abstract Interpreters

Sumit Gulwani
Microsoft Research
sumitg@microsoft.com

Ashish Tiwari
SRI International
tiwari@csl.sri.com

Abstract

We present a methodology for automatically combining abstract interpreters over given lattices to construct an abstract interpreter for the combination of those lattices. This lends modularity to the process of design and implementation of abstract interpreters.

We define the notion of logical product of lattices. This kind of combination is more precise than the reduced product combination. We give algorithms to obtain the join operator and the existential quantification operator for the combined lattice from the corresponding operators of the individual lattices. We also give a bound on the number of steps required to reach a fixed point across loops during analysis over the combined lattice in terms of the corresponding bounds for the individual lattices. We prove that our combination methodology yields the most precise abstract interpretation operators over the logical product of lattices when the individual lattices are over theories that are convex, stably infinite, and disjoint.

We also present an interesting application of logical product wherein some lattices can be reduced to combination of other (unrelated) lattices with known abstract interpreters.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Program analysis

General Terms Algorithms, Theory, Verification

Keywords Abstract Interpreter, Logical Product, Reduced Product, Nelson-Oppen Combination

1. Introduction

Establishing full correctness for general programs is computationally intractable. Hence, program analysis and verification is typically performed over some (sound) abstraction or approximation of the program. This gives rise to false positives, i.e., some properties that are true in the original program may not be true in the abstract version. Abstract Interpretation is a well-known methodology to analyze programs over a given abstraction [5]. There is an efficiency-precision trade-off in the choice of the abstraction. A more precise abstraction leads to fewer false positives but is also harder to reason about.

Research of the second author was supported in part by NSF grant CCR-ITR-0326540.

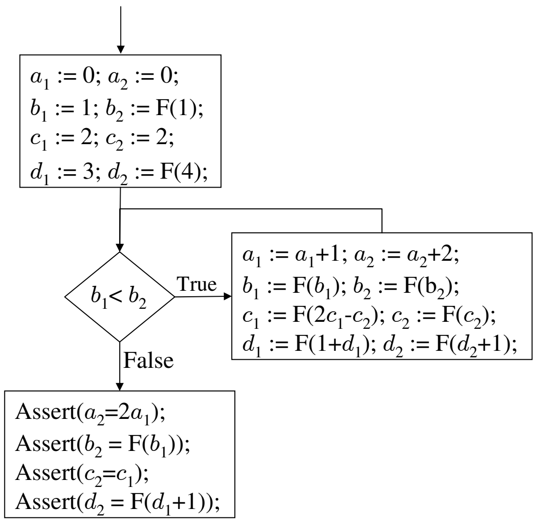
$a_1 := 0$; $a_2 := 0$;
$b_1 := 1$; $b_2 :=$ F(1);
$c_1 := 2$; $c_2 := 2$;
$d_1 := 3$; $d_2 :=$ F(4);
$b_1 < b_2$
True
$a_1 := a_1+1$; $a_2 := a_2+2$;
$b_1 :=$ F(b_1); $b_2 :=$ F(b_2);
$c_1 :=$ F($2c_1$-c_2); $c_2 :=$ F(c_2);
$d_1 :=$ F($1+d_1$); $d_2 :=$ F(d_2+1);
False
Assert($a_2=2a_1$);
Assert($b_2 =$ F(b_1));
Assert($c_2=c_1$);
Assert($d_2 =$ F(d_1+1));

Figure 1. This program illustrates the difference between precision of performing analysis over *direct product*, *reduced product*, and *logical product* of the linear arithmetic lattice and uninterpreted functions lattice. Analysis over direct product can verify the first two assertions, while analysis over reduced product can verify the first three assertions. The analysis over logical product can verify all assertions. F denotes some function without any side-effects and can be modeled as an uninterpreted function for purpose of proving the assertions.

One commonly used method to create more precise abstract domains is by combining simpler ones. There are two commonly used notions of lattice combinations in the literature: the *direct product*, and the *reduced product* [6, 4]. Both these combinations yield a lattice whose elements are a cartesian product of the elements of the individual lattices. The difference is that the lattice operations in the direct product are performed component-wise (also referred to as the *independent attribute method* [20]), while in case of the reduced product the lattice operations take into account both components simultaneously. Hence, the direct product "discovers in one shot the information found separately by the component analyses but we do not learn more by performing all analyses simultaneously than by performing them one after another and finally taking their conjunctions". In case of reduced product the advantage is that "each analysis in the abstract composition benefits from the information brought by the other analyses" [4].

Consider, for example, the program shown in Figure 1. Note that all assertions at the end of the program are true. If this program is analyzed over the linear equalities lattice (using, for example, the abstract interpreter described in [16] or [5]), then only the first assertion can be validated. This is because discovering the relationship between b_1 and b_2, between c_1 and c_2, and between d_1 and d_2 involves reasoning about uninterpreted functions. Similarly, if this program is analyzed over the uninterpreted functions lattice (using, for example, the abstract interpreter described in [12]), then only the second assertion can be validated. Hence, an analysis over the direct product of these lattices can only verify the first two assertions (since performing an analysis over the direct product of these lattices is equivalent to performing analyses over the individual lattices independently and then putting the results together).

An analysis over the reduced product of these lattices can verify the third assertion too. Such an analysis computes at each program point the invariants that involve only linear arithmetic operators or only uninterpreted functions. In particular, it is able to discover the loop invariant $c_1 = c_2$, which is required to prove the third assertion. Hence, the reduced product combination is more precise than the direct product combination. However, there is no automatic way to construct the abstract interpretation operations for the reduced product lattice. In fact, Cousot and Cousot [6] have pointed out that it is not possible to combine two abstract interpreters in a "black-box" manner to obtain the most precise abstract interpreter for the reduced product lattice. The algorithms for the reduced product lattice need to be designed and implemented from scratch.

In this paper, we show how to automatically construct the most precise abstract interpretation for the reduced product combination of lattices that satisfy some constraint (namely, the elements of these lattices are conjunctions of atomic facts over theories that are convex, stably infinite, and disjoint). This constraint is general enough to describe several abstract domains that have been used to build existing abstract interpreters. In fact, we go one step further and define a new notion of combination for such lattices called *logical product*, which is more precise than the reduced product, and we show how to automatically construct abstract interpretation operations for the logical product lattice. The approach of automatically combining abstract interpretation operators lends modularity to the design and implementation of program analyses based on abstract interpretations. It avoids the need for proof of correctness of the analysis over combined domains (in fact, these proofs can be quite involved like our generic proof of correctness of our combination algorithms) and allows for reuse of implementations of analyses over the individual domains.

The logical product of lattices whose elements are conjunctions of atomic facts from theories $\mathbb{T}_1$ and $\mathbb{T}_2$ is the lattice whose elements are conjunctions of atomic facts over the combined theory $\mathbb{T}_1 \cup \mathbb{T}_2$. Note that the logical product lattice consists of more elements than simply the direct product of the lattices, and hence it is more precise than the reduced product. Consider again the program shown in Figure 1. Note that an analysis over the reduced product combination of linear arithmetic and uninterpreted functions lattice cannot verify the fourth assertion because the relevant loop invariant $d_2 = F(d_1 + 1)$ is not expressible as an element of the reduced product lattice, which involves conjunctions of only linear equalities and equalities between uninterpreted function terms as opposed to equalities between mixed expressions (i.e., expressions that involve both linear arithmetic and uninterpreted functions). However, analysis over the logical product of linear arithmetic and uninterpreted functions lattices can verify all four assertions in the program.

Our methodology for combining abstract interpretation operators is inspired by the classic Nelson-Oppen methodology for combining decision procedures [19]. Nelson and Oppen have showed how to combine decision procedures for convex, stably infinite and disjoint theories to obtain a decision procedure for the combined theory with only a polynomial-time blowup in the computational complexity. It turns out that abstract interpretation operators for a lattice over some theory are harder than the decision procedure for that theory. Hence, the problem of combining abstract interpretation operators is harder than the problem of combining decision procedures. As a result, the restrictions on theories that allow for efficient combination of their decision procedures (namely, convexity, stably infiniteness, and disjointness) also transfer to the context of combining abstract interpreters for lattices over those theories.

One of the attractive features of our combination algorithms is that the complexity of the abstract interpretation operators for the logical product lattice is at most quadratic in the complexity of the operations for the individual lattices. Also, our combination methodology is more general than being restricted to lattices over convex, stably infinite, and convex theories. In cases when the lattices to be combined do not satisfy the desired constraint, our combination methodology still gives abstract interpretation operations that are more precise than those for the direct product lattice; however they may not in general be as precise as the reduced product lattice.

In Section 5, we provide an interesting application of reasoning about logical product of lattices. It turns out that some lattices can be modeled as logical product of other unrelated lattices with known abstract interpreters. Hence, abstract interpreters for such decomposable lattices can be constructed by combining abstract interpretation operators for those other unrelated lattices using our combination methodology.

This paper is organized as follows. In Section 2, we introduce some basic terminology and discuss some basic operations in the Nelson-Oppen method for combining decision procedures. We use this terminology and operations in both Section 3 and Section 4. Section 3 defines the logical product combination of lattices, while Section 4 describes how to construct an abstract interpreter for the logical product of lattices given the abstract interpreter for the individual lattices. We describe an interesting application of our logical product combination methodology in Section 5. We then discuss some related work in Section 6 and future work in Section 7.

2. Background

Our methodology for combining abstract interpreters is based on the Nelson-Oppen method of combining decision procedures [19]. In this section, we introduce some terminology and algorithms that are used in the Nelson-Oppen method. We use this terminology and algorithms as part of our combination methodology described in Section 3 and Section 4.

A *theory* $\mathbb{T}$ consists of a signature $\Sigma_{\mathbb{T}}$, which is a set of function and predicate symbols, and some axioms $A_{\mathbb{T}}$, which define the meaning of the function and predicate symbols in $\Sigma_{\mathbb{T}}$. The combination of two theories $\mathbb{T}_1$ and $\mathbb{T}_2$ is the theory $\mathbb{T}_1 \cup \mathbb{T}_2$ such that $\Sigma_{\mathbb{T}_1 \cup \mathbb{T}_2} = \Sigma_{\mathbb{T}_1} \cup \Sigma_{\mathbb{T}_2}$ and $A_{\mathbb{T}_1 \cup \mathbb{T}_2} = A_{\mathbb{T}_1} \cup A_{\mathbb{T}_2}$. A *term* t over theory $\mathbb{T}$ is an expression consisting of variables, and function symbols from $\Sigma_{\mathbb{T}}$. In this paper, we consider the following theories in our examples.

- Theory of parity.
  This theory has the signature $\{=, \mathit{even}, \mathit{odd}, +, -, 0, 1\}$, where even and odd are unary predicates, while $+, -$ are binary functions, and 0 and 1 are constants (nullary functions). The axioms of this theory include all standard axioms of even and odd numbers like $\mathit{even}(0)$, $\mathit{even}(t) \wedge \mathit{odd}(t') \Rightarrow \mathit{odd}(t + t')$, etc. In the latter axiom, t and t' are universally quantified.
- Theory of sign.
  This theory has the signature $\{=, \mathit{positive}, \mathit{negative}, +, -, 0, 1\}$, where $\mathit{positive}$ and $\mathit{negative}$ are unary predicates.

$$\begin{array}{rcl} E & = & x_3 \leq F(2x_2 - x_1) \;\wedge\; x_3 \geq x_1 \;\wedge\; x_1 = F(x_1) \;\wedge\; x_2 = F(F(x_1)) \\ \mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E) & = & \{2x_2 - x_1, F(2x_2 - x_1)\} \\ \mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}(E) & = & \langle V, E_1, E_2 \rangle \text{ where } E_1 \text{ is } t_1 = 2x_2 - x_1 \;\wedge\; x_3 \leq t_2 \;\wedge\; x_3 \geq x_1 \\ & & \qquad \text{and } E_2 \text{ is } t_2 = F(t_1) \;\wedge\; x_1 = F(x_1) \;\wedge\; x_2 = F(F(x_1)) \\ & & \qquad \text{and } V \text{ is } \{t_1, t_2\} \\ \mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}(E_1, E_2) & = & \langle E_1 \wedge E', E_2 \wedge E' \rangle \text{ where } E' \text{ is } x_1 = x_2 \;\wedge\; x_1 = t_1 \;\wedge\; x_1 = t_2 \;\wedge\; x_1 = x_3 \end{array}$$

Figure 2. This example illustrates the functions $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}$, $\mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}$ and $\mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}$, which are used in the Nelson-Oppen method of combining decision procedures. E is a conjunction of atomic facts over the combined theory of linear arithmetic ($\mathbb{T}_1$) and uninterpreted functions ($\mathbb{T}_2$).

- Theory of linear arithmetic.
  This theory has the signature $\{=, \leq, +, -, 0, 1\}$. We sometimes use the phrase *theory of linear arithmetic with only equality* to refer to the theory with signature $\{=, +, -, 0, 1\}$.
- Theory of uninterpreted functions.
  The signature of this theory consists of uninterpreted functions and the equality predicate. The theory of uninterpreted functions (UFS) has only one axiom for each function F^a, namely, $\bigwedge_{i=1}^{a} t_i = t'_i \Rightarrow F^a(t_1,..,t_a) = F^a(t'_1,..,t'_a)$. Alternatively, we can reason about uninterpreted functions using the theory of term algebra (TA), where it is also the case that (1) $F(t_1,\ldots,t_a) = G(t'_1,\ldots,t'_b)$ iff $a = b$, F is same as G, and for all i, $t_i = t'_i$, and (2) any term properly containing x is not equal to x. Due to a technical observation [13], using either of these theories gives the same results in the context of program analysis.
- Theory of lists.
  This theory has the signature $\{\mathit{car}, \mathit{cdr}, \mathit{cons}, =\}$ with the usual axioms.

An *atomic fact* f over theory $\mathbb{T}$ is a predicate of the form $p(t_1,..,t_a)$, where p is a predicate symbol from $\Sigma_{\mathbb{T}}$ and $t_1,..,t_a$ are terms over $\mathbb{T}$. We use the notation $\mathit{Vars}(\gamma)$ and $\mathit{Symbols}(\gamma)$ to denote the set of variables and symbols respectively that occur in γ, where γ may be a term, fact, or conjunction of facts. We use the term *definition* for an atomic fact of the form $x = t$, where the variable x does not occur in the term t.

Let E be any conjunction of atomic facts over combination of two theories $\mathbb{T}_1 \cup \mathbb{T}_2$. We define $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ to be the set of all alien terms that occur in E. A term t in E is *alien* if the top-level function symbol in t belongs to $\Sigma_{\mathbb{T}_1}$ (or $\Sigma_{\mathbb{T}_2}$) and t occurs as an argument of some function or predicate symbol from $\Sigma_{\mathbb{T}_2}$ (or $\Sigma_{\mathbb{T}_1}$ respectively) in E. For example, consider the conjunction E over the combination of the theories of linear arithmetic and uninterpreted functions in Figure 2. Note that $2x_2 - x_1$ is an alien term because it is a linear arithmetic term that occurs as an argument of uninterpreted function F (in the term $F(2x_2 - x_1)$). Similarly, the term $F(2x_2 - x_1)$ is an alien term because its top-level operator is the uninterpreted function F while it occurs as an argument of the inequality predicate in $x_3 \leq F(2x_2 - x_1)$.

The $\mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}$ operator takes as input a conjunction of atomic facts E over combination of two theories $\mathbb{T}_1 \cup \mathbb{T}_2$ and returns $\langle V, E_1, E_2 \rangle$, where E_1 and E_2 are conjunctions of atomic facts over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively, and V is the set of all fresh variables that occur in E_1 or E_2 (but do not occur in E). Furthermore, $E_1 \wedge E_2$ is a *conservative extension* of E, i.e., for all facts f that do not involve variables in V, the following holds:

$$E \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} f \quad \text{iff} \quad E_1 \wedge E_2 \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} f$$

Purification (i.e., the operation $\mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}(E)$) decomposes a conjunction of atomic facts over combined theory $\mathbb{T}_1 \cup \mathbb{T}_2$ into conjunctions of atomic facts, each of which is either over theory $\mathbb{T}_1$ or over theory $\mathbb{T}_2$. The purification of E involves introducing a fresh variable for each alien term in E. For example, consider the conjunction of atomic facts E in Figure 2. $\mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ is obtained from E by introducing new variables t_1 and t_2 to represent the alien terms of E.

The $\mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}(E_1, E_2)$ operator takes as input two conjunctions of atomic facts E_1 and E_2 over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively, and returns E'_1 and E'_2 that are obtained from E_1 and E_2 respectively by saturating them with variable equalities that are implied by $E_1 \wedge E_2$.

$$\begin{array}{rcl} E'_1 & = & E_1 \wedge E \\ E'_2 & = & E_2 \wedge E \\ E & = & \bigwedge_{P(x,y)} x = y \end{array}$$

where $P(x,y)$ is the following predicate:

$$(x, y \in \mathit{Vars}(E_1 \wedge E_2)) \;\wedge\; (E_1 \wedge E_2 \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} x = y)$$

E can be computed by repeatedly sharing variable equalities between E_1 and E_2. For example, consider the conjunction of atomic facts E_1 and E_2 over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively in Figure 2. E'_1 and E'_2 are obtained from E_1 and E_2 by sharing variable equalities between them as follows:

$$E_2 \xrightarrow{x_1 = x_2} E_1 \xrightarrow{x_1 = t_1} E_2 \xrightarrow{x_1 = t_2} E_1 \xrightarrow{x_1 = x_3} E_2$$

A corollary of the correctness of the Nelson-Oppen combination method [19] is that the $\mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}$ operator has the following interesting property when theories $\mathbb{T}_1$ and $\mathbb{T}_2$ are *convex*, *stably infinite* and *disjoint*.

PROPERTY 1 ($\mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}$). *Let $\mathbb{T}_1$ and $\mathbb{T}_2$ be two convex and stably infinite theories that are disjoint. Let E be any conjunction of atomic facts over $\mathbb{T}_1 \cup \mathbb{T}_2$. Let f be any atomic fact over $\mathbb{T}_1$. Let $\langle V, E_1, E_2 \rangle = \mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ and $\langle E'_1, E'_2 \rangle = \mathit{NOSaturation}_{\mathbb{T}_1,\mathbb{T}_2}(E_1, E_2)$. Then,*

$$E \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} f \quad \textit{iff} \quad E'_1 \overset{\mathbb{T}_1}{\Rightarrow} f$$

A theory $\mathbb{T}$ is *convex* iff for every quantifier-free formula ϕ, $\phi \overset{\mathbb{T}}{\Rightarrow} \bigvee_i x_i = y_i$ implies $\phi \overset{\mathbb{T}}{\Rightarrow} x_j = y_j$ for some j. A theory $\mathbb{T}$ is *stably infinite* iff for every quantifier-free formula ϕ, ϕ is satisfiable in $\mathbb{T}$ iff ϕ is satisfiable in an infinite model of $\mathbb{T}$. Two theories $\mathbb{T}_1$ and $\mathbb{T}_2$ are *disjoint* if their signatures $\Sigma_{\mathbb{T}_1}$ and $\Sigma_{\mathbb{T}_2}$ are disjoint, while ignoring the equality symbol. For example, the theories of uninterpreted functions, linear arithmetic, and lists are all disjoint with respect to each other. In contrast, the theories of parity and

$$
\begin{array}{rcl}
E_1 & = & (x = a) \;\wedge\; (y = b) \\
E_2 & = & (x = b) \;\wedge\; (y = a) \\
J_{L_1}(E_1, E_2) & = & (x + y = a + b) \\
J_{L_2}(E_1, E_2) & = & true
\end{array}
$$

$$E_1 \vee E_2 \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} \bigwedge_c F(x+c) + F(y+c) = F(a+c) + F(b+c)$$

L_1: logical lattice over theory of linear arithmetic ($\mathbb{T}_1$)
L_2: logical lattice over theory of term algebra ($\mathbb{T}_2$)

Figure 3. This figure demonstrates why the implication relationship over finite conjunctions of atomic facts in the combined theory of linear arithmetic and term algebra does not form a lattice. Note that $x + y = a + b$ is the only (independent) atomic fact that is implied by both E_1 and E_2 over the theory of linear arithmetic. Hence the join (i.e., least upper bound) of E_1 and E_2 in the logical lattice L_1, denoted by $J_{L_1}(E_1, E_2)$, is $x + y = a + b$. Also, note that there is no atomic fact over uninterpreted functions that is implied by both E_1 and E_2. However, over the combined theory of linear arithmetic and term algebra, there are infinite number of atomic facts that are implied by both E_1 and E_2; one such infinite family is: $F(x + c) + F(y + c) = F(a + c) + F(b + c)$ for all linear arithmetic constants c.

sign are not disjoint. However, all of these five theories are convex and stably infinite.

3. Logical Lattices and Their Combination

A lattice L consists of a set D_L and a partial order $\preceq_L$ among elements of D_L. In this paper, we consider logical lattices, as defined below.

DEFINITION 1 (Logical Lattice). *A lattice L is a logical lattice over some theory $\mathbb{T}$ if D_L is the set of all finite conjunctions of atomic facts from theory $\mathbb{T}$, and the partial order $\preceq_L$ is the implication relationship $\overset{\mathbb{T}}{\Rightarrow}$ in theory $\mathbb{T}$, i.e., $E \preceq_L E'$ iff $E \overset{\mathbb{T}}{\Rightarrow} E'$.*

Any abstract domain can be viewed as a logical lattice over an appropriate theory. (However, the theory may not be convex or stably infinite, which are the assumptions required to guarantee the precision of our combination methodology.) For example, the abstract lattice used for discovering linear equalities between program variables [16, 10, 18] is a logical lattice over the theory of linear arithmetic with only equality, while the one used for discovering linear inequality relationships [7] is over the general theory of linear arithmetic. The abstract lattice used for global value numbering for discovering Herbrand equivalences [11, 12] is a logical lattice over the theory of term algebra.

The implication relation in any theory $\mathbb{T}$ always defines a semi-lattice. A sufficient condition for this semi-lattice to be a lattice is that $\mathbb{T}$ have a *finite basis property*, that is, every infinite conjunction of atomic formulas over a finite number of variables be equivalent to a finite conjunction in the theory.

Let L_1 and L_2 be two logical lattices over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively. In the next section, we describe algorithms that perform abstract interpretation over the semi-lattice induced by $\mathbb{T}_1 \cup \mathbb{T}_2$. Since $\mathbb{T}_1 \cup \mathbb{T}_2$ need not induce a logical lattice even if $\mathbb{T}_1$ and $\mathbb{T}_2$ do, see Figure 3 for example, we precisely characterize the power of our combination algorithms by introducing the notion of logical product of lattices induced by convex, stably infinite, and disjoint theories.

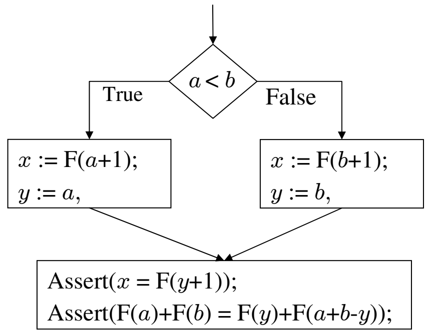

Figure 4. This program illustrates the difference between precision of *strict* logical product combination and logical product combination of lattices (over linear arithmetic and uninterpreted functions). Abstract interpretation over strict logical product combination can verify both assertions because the join of $x = F(a + 1) \wedge y = a$ and $y = F(b + 1) \wedge y = b$ includes equalities in both assertions. (This involves representing and manipulating infinite conjunctions of atomic facts, which is inefficient.) On the other hand, abstract interpretation over the logical product combination can verify only the first assertion because the result of the join is $x = F(y + 1)$.

DEFINITION 2 (Logical Product of Logical Lattices). *Let $\mathbb{T}_1$ and $\mathbb{T}_2$ be two convex, stably-infinite, and disjoint theories. The logical product of the two logical lattices L_1 and L_2 over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively is defined to be the lattice $L_1 \bowtie L_2$ where $D_{L_1 \bowtie L_2}$ is the set of all finite conjunction of atomic facts from theory $\mathbb{T}_1 \cup \mathbb{T}_2$, and $\preceq_{L_1 \bowtie L_2}$ is the following partial order:*

$$E \preceq_{L_1 \bowtie L_2} E' \overset{def}{=} (E \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} E') \;\wedge\; (AlienTerms_{\mathbb{T}_1, \mathbb{T}_2}(E') \subseteq Terms_{\mathbb{T}_1, \mathbb{T}_2}(E))$$

For any conjunction of facts E in combination of two theories $\mathbb{T}_1$ and $\mathbb{T}_2$, we define $Terms_{\mathbb{T}_1, \mathbb{T}_2}(E)$ to be the following set.

$$Terms_{\mathbb{T}_1, \mathbb{T}_2}(E) = \{t \mid \exists t' \in Vars(E) \cup AlienTerms_{\mathbb{T}_1, \mathbb{T}_2}(E) \text{ such that } E \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} t = t'\}$$

It would have been more natural to define $\preceq_{L_1 \bowtie L_2}$ to be simply $\overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow}$; but then as mentioned above, unfortunately, $\overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow}$ does not necessarily form a lattice even when $\preceq_{L_1}$ and $\preceq_{L_2}$ define a logical lattice. One way to solve this problem would be to relax the domain $D_{L_1 \bowtie L_2}$ to also include infinite conjunction of atomic facts; in that case the choice of $\overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow}$ as $\preceq_{L_1 \bowtie L_2}$ defines a lattice (and we refer to this as the *strict* logical product combination). However, this is not a good practical solution because we now need to represent and manipulate infinite conjunctions of atomic facts. Our recent results [13] on hardness of assertion checking for programs that involve only linear arithmetic and uninterpreted functions imply that there cannot be any efficient data structure and algorithms to reason about such infinite conjunctions of atomic facts in general (unless P=coNP). [1]

[1] We have shown that the problem of assertion checking in programs whose expressions involve linear arithmetic and uninterpreted functions, and whose conditionals have been abstracted as non-deterministic branches, is coNP-hard. This problem can be solved by performing abstract interpretation over the lattice whose elements are (potentially infinite) conjunctions of atomic facts over the combined theory of linear arithmetic and uninterpreted functions, and whose partial order is the implication relationship. This implies that there cannot be any data-structures and algorithms that

Our solution to solve the above problem is to define $\preceq_{L_1 \bowtie L_2}$ to have an additional restriction that is as weak as possible and along with the implication relationship defines a lattice. This additional restriction is $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq \mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$. The failure of E_1 and E_2 (as defined in Figure 3) to have a least upper bound under the implication relationship can be attributed to the fact that the number of alien terms that can be constructed over combination of two theories, even with a finite number of program variables, is unbounded. Considering that, the restriction $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq \mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ in the definition of $\preceq_{L_1 \bowtie L_2}$ is quite natural; it has the effect of including only those atomic facts in the least upper bound of E and E' whose alien terms occur semantically in both elements E and E'. By semantic (as opposed to syntactic) occurrence of an alien term t of E' in E, we mean that there is some variable or alien term t' in E such that $E \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} t = t'$, i.e., $t \in \mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ (as opposed to $t \in \mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$). The program in Figure 4 illustrates the differences between the precision of *strict* logical product combination and logical product combination of lattices.

Note that the syntactic restriction $t \in \mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$ would have been an (unnecessarily) stronger restriction compared to the semantic restriction $t \in \mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$. For example, consider the facts $E_1 = (x = F(a{+}1)) \wedge (y = a)$ and $E_2 = (x = F(b+1)) \wedge (y = b)$ over the combined theory of linear arithmetic and uninterpreted functions. The result of join of E_1 and E_2 is $x = F(y+1)$ under our definition of $\preceq_{L_1 \bowtie L_2}$, while it is *true* in case of the stronger syntactic restriction $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq \mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E)$. This is because the alien term $y + 1$ in $x = F(y{+}1)$ belongs to $\mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E_1)$ (since $E_1 \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} y{+}1 = a+1$ and $a+1$ is an alien term in E_1) and $\mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E_2)$ but it is not an alien term in E_1 or E_2.

The following theorem states that $\preceq_{L_1 \bowtie L_2}$ as defined in Definition 2 indeed defines a lattice.

THEOREM 1. *The partial order $\preceq_{L_1 \bowtie L_2}$ between finite conjunctions of facts from theory $\mathbb{T}_1 \cup \mathbb{T}_2$ defines a lattice under the assumptions made in Definition 2.*

The proof of Theorem 1 follows from the fact that any two elements $E, E' \in D_{L_1 \bowtie L_2}$ have a least upper bound (which is computed by the algorithm described in Figure 6(a)) and a greatest upper bound (which is $E \wedge E'$).

4. Abstract Interpreter for Combination of Logical Lattices

Let L_1 and L_2 be some logical lattices over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ respectively. In this section, we show how to efficiently combine the abstract interpreters that operate over the lattices L_1 and L_2 to obtain an abstract interpreter that operates over the combined lattice $L_1 \bowtie L_2$. Our combination methodology yields the most precise abstract interpreter for the combined lattice $L_1 \bowtie L_2$ when (a) the theories $\mathbb{T}_1$ and $\mathbb{T}_2$ are convex, stably infinite, and disjoint, and (b) the individual abstract interpreters that operate over the lattices L_1 and L_2 are most precise themselves. The key idea of our combination methodology is to combine the corresponding transfer functions of the abstract interpreters that operate over the lattices L_1 and L_2 to yield the transfer functions of the abstract interpreter that operates over the lattice $L_1 \bowtie L_2$.

An abstract interpreter performs a forward analysis on the program computing invariants (which are elements of the underlying lattice over which the analysis is being performed) at each program point. The invariants are computed at each program point from the invariants at the preceding program points in an iterative manner using appropriate transfer functions. The abstract interpreter that operates over the lattice $L_1 \bowtie L_2$ uses the following transfer functions to compute these invariants across the different flowchart nodes shown in Figure 5.

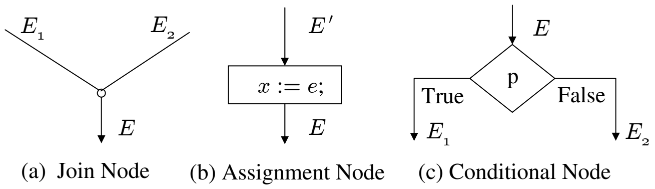

Figure 5. Flowchart Nodes.

- Join Node. See Figure 5(a).
  The element E after a join node is obtained by computing the least upper bound of the elements E_1 and E_2 before the join node (in the lattice $L_1 \bowtie L_2$).

$$E = J_{L_1 \bowtie L_2}(E_1, E_2)$$

  The join operator J_L for any lattice L takes as input two elements E_1 and E_2 from D_L and computes their least upper bound (under the partial order $\preceq_L$). In Section 4.1, we show how to obtain $J_{L_1 \bowtie L_2}$ from J_{L_1} and J_{L_2}.

- Assignment Node. See Figure 5(b).
  First note that an assignment $x := e$ is general enough to model any programming language assignment. Memory, for example, can be modeled using array variables and select and update expressions, without losing any precision.

  The element E after an assignment node $x := e$ is the strongest postcondition of the element E' before the assignment node. It is computed by using an existential quantification operator $Q_{L_1 \bowtie L_2}$ as described below.

$$\begin{array}{rcl} E & = & Q_{L_1 \bowtie L_2}(E_1, \{x'\}) \\ \text{where } E_1 & = & E'[x'/x] \wedge E'_1 \\ \text{and } E'_1 & = & \begin{cases} x = e[x'/x] & \text{if } \mathit{Symbols}(e) \subseteq \Sigma_{\mathbb{T}_1 \cup \mathbb{T}_2} \\ \mathit{true} & \text{otherwise} \end{cases} \end{array}$$

  The existential quantification operator Q_L for any lattice L takes as input an element E from D_L and a set of variables V, and produces the least upper bound of E (in the lattice L) that does not involve any variables in V. In Section 4.2, we show how to obtain $Q_{L_1 \bowtie L_2}$ from Q_{L_1} and Q_{L_2}.

- Conditional Node. See Figure 5(c).
  The elements E_1 and E_2 on the two branches of a conditional are obtained by computing the meet (i.e., the greatest lower bound) of the element E before the join node with any atomic fact over the theory $\mathbb{T}_1 \cup \mathbb{T}_2$ that is implied by the conditional on the corresponding branch.

$$\begin{array}{rcl} E_1 & = & M_{L_1 \bowtie L_2}(E, E'_1) \\ E'_1 & = & \begin{cases} p & \text{if } p \text{ is an atomic fact over theory } \mathbb{T}_1 \cup \mathbb{T}_2 \\ \mathit{true} & \text{otherwise} \end{cases} \end{array}$$

can represent and perform abstract interpretation operations on infinite conjunctions of atomic facts in polynomial time unless P=coNP.

```
    $J_{L_1 \bowtie L_2}(E^\ell, E^r)$ =
1   $\langle V^\ell, E_1^{\ell 0}, E_2^{\ell 0}\rangle$ := $Purify_{\mathbb{T}_1,\mathbb{T}_2}(E^\ell)$;
2   $\langle E_1^{\ell 1}, E_2^{\ell 1}\rangle$ := $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^{\ell 0}, E_2^{\ell 0})$;
3   $\langle V^r, E_1^{r0}, E_2^{r0}\rangle$ := $Purify_{\mathbb{T}_1,\mathbb{T}_2}(E^r)$;
4   $\langle E_1^{r1}, E_2^{r1}\rangle$ := $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^{r0}, E_2^{r0})$;
5   $V$ := $\{\langle x,y\rangle \mid x \in V^\ell \cup Vars(E^\ell), y \in V^r \cup Vars(E^r)\}$;
6   $E^{\ell 2}$ := $\bigwedge_{\langle x,y\rangle \in V} x = \langle x,y\rangle$ ;
7   $E^{r2}$ := $\bigwedge_{\langle x,y\rangle \in V} y = \langle x,y\rangle$ ;
8   $E_1$ := $J_{L_1}(E_1^{\ell 1} \wedge E^{\ell 2}, E_1^{r1} \wedge E^{r2})$;
9   $E_2$ := $J_{L_2}(E_2^{\ell 1} \wedge E^{\ell 2}, E_2^{r1} \wedge E^{r2})$;
10  $E$ := $Q_{L_1 \bowtie L_2}(E_1 \wedge E_2, V)$;
11  return $E$;
```

(a) Algorithm

Inputs:

$$\begin{aligned} E^\ell &= (u = F(w)) \wedge (w = v+1) \\ E^r &= (u = F(u)) \wedge (v = F(u) - 1) \end{aligned}$$

Trace of $J_{L_1 \bowtie L_2}(E_1, E_2)$:

$$\begin{aligned} \langle V^\ell, E_1^{\ell 0}, E_2^{\ell 0}\rangle &= \langle \{\}, w = v+1, u = F(w)\rangle \\ \langle E_1^{\ell 1}, E_2^{\ell 1}\rangle &= \langle w = v+1, u = F(w)\rangle \\ \langle V^r, E_1^{r0}, E_2^{r0}\rangle &= \langle \{b\}, v = b-1, b = F(u) \wedge u = F(u)\rangle \\ \langle E_1^{r1}, E_2^{r1}\rangle &= \langle v = b-1 \wedge u = b, b = F(u) \wedge u = F(u)\rangle \\ V &= \{\langle w,b\rangle, \langle w,u\rangle, \langle w,v\rangle, \langle u,b\rangle, \langle u,u\rangle, \langle u,v\rangle, \ldots\} \\ E^{\ell 2} &= (w = \langle w,b\rangle) \wedge (w = \langle w,u\rangle) \wedge (w = \langle w,v\rangle) \ldots \\ E^{r2} &= (b = \langle w,b\rangle) \wedge (u = \langle w,u\rangle) \wedge (v = \langle w,v\rangle) \ldots \\ E_1 &= (v = \langle w,b\rangle + 1) \\ E_2 &= (u = F(\langle w,b\rangle)) \\ E &= (u = F(v+1)) \end{aligned}$$

(b) Example

Figure 6. This figure describes the algorithm for join operator for combined lattice $L_1 \bowtie L_2$ in terms of the join operators for the lattices L_1 and L_2 along with an example.

Similarly,

$$\begin{aligned} E_2 &= M_{L_1 \bowtie L_2}(E, E_2') \\ E_2' &= \begin{cases} \neg p & \text{if } \neg p \text{ is an atomic fact over theory } \mathbb{T}_1 \cup \mathbb{T}_2 \\ true & \text{otherwise} \end{cases} \end{aligned}$$

Here $M_{L_1 \bowtie L_2}$ denotes the meet operator for the lattice $L_1 \bowtie L_2$ and can be implemented simply as a conjunction operator.

$$M_{L_1 \bowtie L_2}(E', E'') = E' \wedge E''$$

In presence of loops in programs, the abstract interpreter goes around each loop until a fixed point is reached. A fixed point is said to be reached when the elements at any program point inside the loop in two successive iterations of that loop represent the same lattice element. We show in Section 4.3 that the number of steps required to reach a fixed point across a loop (when the analysis is performed over the lattice $L_1 \bowtie L_2$) is linear in the number of steps required to reach a fixed point across that loop when the analysis is performed over the lattices L_1 or L_2. If the lattices L_1 or L_2 have infinite chains above a given element, then fixed point for a loop may not be reached in a finite number of steps. In that case, a widening operation may be used to over-approximate the analysis results at loop headers. A widening operator for a lattice L takes as input two elements from D_L and produces an upper bound of those elements (which may not necessarily be the least upper bound). A widening operator has the property that it guarantees fixed point computation across loops terminates in a finite number of steps even for infinite height lattices. A widening operator for the lattice $L_1 \bowtie L_2$ can be constructed from widening operators for the lattices L_1 and L_2 respectively in exactly the same way as $J_{L_1 \bowtie L_2}$ is constructed from J_{L_1} and J_{L_2}. Section 4.3 discusses these issues in more detail.

4.1 Combining Join Operators

The join operator J_L for a lattice L takes as input two elements E_1 and E_2 from D_L and computes the least upper bound of E_1 and E_2 with respect to the partial order $\preceq_L$. The following definition makes this more precise.

DEFINITION 3 (Join Operator J_L). *Let $E = J_L(E_1, E_2)$. Then,*

- *Soundness: $E_1 \preceq_L E$ and $E_2 \preceq_L E$.*
- *Completeness: If E' is such that $E_1 \preceq_L E'$ and $E_2 \preceq_L E'$, then $E \preceq_L E'$.*

Figure 6 shows how to implement the join operator $J_{L_1 \bowtie L_2}$ for the combined lattice $L_1 \bowtie L_2$ using the join operators J_{L_1} and J_{L_2} for the logical lattices L_1 and L_2. Lines 1 and 2 perform purification and NO-saturation of the input E^ℓ. Purification of E^ℓ (which involves atomic facts over combination of two theories $\mathbb{T}_1 \cup \mathbb{T}_2$) serves the purpose of splitting the input E^ℓ into two parts $E_1^{\ell 0}$ and $E_2^{\ell 0}$ each of which involves facts over either $\mathbb{T}_1$ or $\mathbb{T}_2$, and hence can be understood by either J_{L_1} or J_{L_2}. The NO-saturation of $E_1^{\ell 0}$ and $E_2^{\ell 1}$ serves the purpose of sharing information between $E_1^{\ell 0}$ and $E_2^{\ell 1}$ so that each of them can independently imply the atomic facts in corresponding theories that are implied by E^ℓ (Property 1). Similarly, lines 3 and 4 perform purification and NO-saturation of the other input E^r.

Lines 5 through 7 introduce some dummy variables and definitions for purpose of creating variable names for some potential alien terms in the output. If we leave out lines 5 through 7 in the combination algorithm, then we simply obtain a join operator for the reduced product combination of lattices L_1 and L_2 because both E_1 and E_2 are atomic facts involving only those variables that occur in E^ℓ and E^r. However, presence of these dummy variables and definitions allow the individual join operators J_{L_1} and J_{L_2} to output result in terms of these dummy variables along with their definitions. Elimination of these dummy variables in line 10 (using the existential quantification operator $Q_{L_1 \bowtie L_2}$ described in the next section) results in *mixed* facts over the combined theory $\mathbb{T}_1 \cup \mathbb{T}_2$. In the example in Figure 6(b), $v+1$ is such an alien term in the output, which is represented by the dummy variable $\langle a, b\rangle$ in $E_1^{\ell 1} \wedge E^{\ell 2}$ as well as in $E_1^{r2} \wedge E^{r2}$.

The operator $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ involves sharing variable equalities between its input elements until no more equalities can be shared. It can be implemented using an operator $VE_{\mathbb{T}}$ that takes as input a conjunction of atomic facts in theory $\mathbb{T}$ and discovers all variable equalities implied by it. The $VE_{\mathbb{T}}$ operator for any theory $\mathbb{T}$ is theoretically at most as hard as the join operator J_L for the logical lattice L over theory $\mathbb{T}$, as is exhibited by the following construction.

```
   $Q_{L_1 \bowtie L_2}(E,V)$ =
1    $\langle V^0, E_1^0, E_2^0\rangle$ := $Purify_{\mathbb{T}_1,\mathbb{T}_2}(E)$;
2    $\langle E_1^1, E_2^1\rangle$ := $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^0, E_2^0)$;
3    $V^1$ := $V^0 \cup V$;
4    $\langle V^2$,Defs$\rangle$ := $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^1, E_2^1, V^1)$;
5    $E_1^2$ := $Q_{L_1}(E_1^1, V^2)$;
6    $E_2^2$ := $Q_{L_2}(E_2^1, V^2)$;
7    $E_1^3$ := $E_1^2$[Defs$(y)/y$] for all $y \in V^2 - V^1$;
8    $E_2^3$ := $E_2^2$[Defs$(y)/y$] for all $y \in V^2 - V^1$;
9    return $E_1^3 \wedge E_2^3$;

   $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^1, E_2^1, V^1)$ =
1    $V^2 := V^1$;
2    Defs $:= \emptyset$;
3    repeat
4      for all $y \in V^1$
5        $t := Alternate_{\mathbb{T}_1}(E_1^1, y, V^2)$;
6        if $t = \bot$, then $t := Alternate_{\mathbb{T}_2}(E_2^1, y, V^2)$;
7        if $t \neq \bot$, then Defs := Defs $\wedge\, y = t$;
8                               $V^2 := V^2 - \{y\}$;
9    until no change in $V^2$;
10   return $\langle V^2$,Defs$\rangle$;
```

(a) Algorithm

Inputs:

$$\begin{array}{rcl} E & = & (x \le y) \wedge (y \le u) \wedge (x = F(F(1+y))) \wedge (v = F(y+1)) \\ V & = & \{x, y\} \end{array}$$

Trace of $Q_{L_1 \bowtie L_2}(E,V)$:

$$\begin{array}{rcl} \langle V^0, E_1^0, E_2^0\rangle & = & \langle \{a\}, x \le y \ \wedge\ y \le u \ \wedge\ a = 1+y \ \wedge\ b = y+1, \\ & & \quad x = F(F(a)) \ \wedge\ v = F(b)\rangle \\ \langle E_1^1, E_2^1\rangle & = & \langle x \le y \ \wedge\ y \le u \ \wedge\ a = 1+y \ \wedge\ b = y+1, \\ & & \quad x = F(F(a)) \ \wedge\ v = F(b) \ \wedge\ a = b\rangle \\ V^1 & = & \{x, y, a, b\} \\ \langle V^2, \texttt{Defs}\rangle & = & \langle \{y, a, b\}, x = F(v)\rangle \\ E_1^2 & = & x \le u \\ E_2^2 & = & \mathit{true} \\ E_1^3 & = & F(v) \le u \\ E_2^3 & = & \mathit{true} \end{array}$$

(b) Example

Figure 7. This figure describes the algorithm for existential quantification operator for combined lattice $L_1 \bowtie L_2$ in terms of the existential quantification operators for the lattices L_1 and L_2 along with an example.

$$\begin{array}{rcl} VE_{\mathbb{T}}(E) & = & J_L(E, E') \\ \text{where } E' & = & \bigwedge_{i=1}^{k} x_0 = x_i \\ \text{and } \{x_0, \ldots, x_k\} & = & Vars(E) \end{array}$$

However, for several theories, the $VE_{\mathbb{T}}$ operator for a theory can be implemented in a simpler and more efficient manner than the join operator for the corresponding logical lattice. For example, the $VE_{\mathbb{T}}$ operator for the theory of uninterpreted functions can be implemented using congruence closure algorithm, while the join operator for the theory of uninterpreted functions is based on automata intersection [15]. The $VE_{\mathbb{T}}$ operator for the theory of linear arithmetic with only equality (i.e., no inequalities) can be implemented simply using Gaussian elimination, while the join algorithm is more involved [16].

The following theorem states that our algorithm for the join operator $J_{L_1 \bowtie L_2}(E^\ell, E^r)$ (as described in Figure 6(a)) computes an upper bound of E^ℓ and E^r.

THEOREM 2 (Soundness of $J_{L_1 \bowtie L_2}$ algorithm). *If $E = J_{L_1 \bowtie L_2}(E^\ell, E^r)$, then $E^\ell \preceq_{L_1 \bowtie L_2} E$ and $E^r \preceq_{L_1 \bowtie L_2} E$.*

The proof of Theorem 2 is not difficult and can be found in the full version of this paper [14].

The following theorem states that the upper bound computed by our algorithm for the join operator $J_{L_1 \bowtie L_2}(E^\ell, E^r)$ is least if the underlying theories are convex, stably infinite, and disjoint.

THEOREM 3 (Completeness of $J_{L_1 \bowtie L_2}$ algorithm). *Suppose the theories $\mathbb{T}_1$ and $\mathbb{T}_2$ are convex, stably infinite and disjoint. Let $E = J_{L_1 \bowtie L_2}(E^\ell, E^r)$. Let E' be such that $E^\ell \preceq_{L_1 \bowtie L_2} E'$ and $E^r \preceq_{L_1 \bowtie L_2} E'$. Then $E \preceq_{L_1 \bowtie L_2} E'$.*

The proof of Theorem 3 is non-trivial and is given in the appendix.

4.2 Combining Existential Quantification Operators

$Q_L(E, V)$ computes the best approximation to the existentially quantified element $\exists x E$ (where V is some set of variables) in the lattice L. The following definition makes this more precise.

DEFINITION 4 (Existential Quantification Operator Q_L). *Let $E' = Q_L(E, V)$. Then,*

- *Soundness: $E \preceq_L E'$ and $Vars(E') \cap V = \emptyset$*
- *Completeness: If E'' is such that $E \preceq_L E''$ and $Vars(E'') \cap V = \emptyset$, then $E' \preceq_L E''$.*

Figure 7 shows how to combine existential quantification operators for logical lattices. Lines 1 and 2 perform purification and NO-saturation of the input E (for the same reasons as in case of our algorithm for $J_{L_1 \bowtie L_2}$ operator). Purification introduces some new variables V^0, which also need to be eliminated. V^1 on line 3 represents the set of all variables that need to be eliminated. We first filter out those variables from V^1 that have some definition t such that $Vars(t) \cap V' = \emptyset$ using the operator $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1, E_2, V^1)$. The following claim states this more formally.

CLAIM 1. *Let $(V^2, \mathit{Defs}) = QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^1, E_2^1, V^1)$, where E_1^1 and E_2^1 are purified and NO-saturated. If $E_1^1 \wedge E_2^1 \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} y = t$ and $Vars(t) \cap V^1 = \emptyset$, then $y \notin V^2$.*

In lines 5 and 6, we use the existential quantification operators for the individual lattices to eliminate the variables V^2 from E_1^1 and E_2^1. We then eliminate all variables in the set $V^1 - V^2$ by replacing them by their definitions (lines 7 and 8).

The implementation of $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ operator makes use of operators $Alternate_{\mathbb{T}_1}$ and $Alternate_{\mathbb{T}_2}$. $Alternate_{\mathbb{T}}(E, y, V)$ returns a term t (if it exists) such that $E \overset{\mathbb{T}}{\Rightarrow} y = t$ and $Vars(t) \cap (V \cup \{y\}) = \emptyset$. If no such term exists, then $Alternate_{\mathbb{T}}(E, y, V)$ returns $\bot$. The implementation of $Alternate_{\mathbb{T}}$ operator is theory-specific. For example, the $Alternate_{\mathbb{T}}$ operator for the theory of uninterpreted functions involves constructing a congruence closed EDAG representing E, erasing all variables in $V \cup \{y\}$ from the EDAG, and then returning any term t from the equivalence class of y. The $Alternate_{\mathbb{T}}$ operator for the theory of linear arithmetic can be implemented by discovering all equalities implied by E (by using a linear inequalities reasoning algorithm like simplex or ellipsoid algorithm), eliminating all variables in V using Gaussian elimination, and then returning any term t from any equality that involves variable y. We do not know of any general algorithm for implementing $Alternate_{\mathbb{T}}$, however we conjecture that the $Alternate_{\mathbb{T}}$ operator for any theory $\mathbb{T}$ is at most as hard as the existential quantification operator Q_L for the corresponding logical lattice L since $Q_L(E, V \cup \{y\})$ should ideally replace all occurrences of y in E by the desired term t (if it exists) unless it rewrites the facts in a different manner.

The following theorem states that our algorithm for the existential quantification operator $Q_{L_1 \bowtie L_2}(E, V)$ (as described in Figure 7(a)) computes an upper bound of E that does not involve any variables in V. The proof of this theorem is not difficult and can be found in the full version of this paper [14].

THEOREM 4 (Soundness of $Q_{L_1 \bowtie L_2}$ algorithm). *Let $E' = Q_{L_1 \bowtie L_2}(E, V)$. Then, $E \preceq_{L_1 \bowtie L_2} E'$ and $Vars(E') \cap V = \emptyset$.*

The following theorem states that the upper bound of E computed by our algorithm for the existential quantification operator $J_{L_1 \bowtie L_2}(E, V)$ is least if the underlying theories are convex, stably infinite, and disjoint.

THEOREM 5 (Completeness of $Q_{L_1 \bowtie L_2}$ algorithm). *Suppose the theories $\mathbb{T}_1$ and $\mathbb{T}_2$ are convex, stably infinite and disjoint. Let $E' = Q_{L_1 \bowtie L_2}(E, V)$. Let E'' be such that $E \preceq_{L_1 \bowtie L_2} E''$ and $Vars(E'') \cap V = \emptyset$. Then, $E' \preceq_{L_1 \bowtie L_2} E''$.*

The proof of Theorem 5 is non-trivial and can be found in the full version of this paper [14]. Figure 8 describes an example that illustrates why the completeness of our combination algorithm for existential quantification operator relies on the fact that the underlying theories are disjoint. This example is adapted from Cousots' early work [6] in which they argued that it is not possible to combine abstract interpreters for any two arbitrary abstractions in a black-box manner to obtain the most precise abstract interpreter for the combined lattice. The example used to illustrate this point was the computation of strongest postcondition of $even(x) \wedge positive(x)$ with respect to the assignment $x := x - 1$. The abstract interpreter operating over the parity abstraction will compute $odd(x)$ while the abstract interpreter operating over the sign abstraction will compute $true$, and there is no way to combine the results of these abstract interpreters in a black-box manner to generate the most precise result, which is $odd(x) \wedge positive(x)$.

It is interesting to note that if we define V^2 on line 4 to be V^1 (and not invoke the $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ operator), then we obtain an existential quantification operator for the reduced product combination of lattices L_1 and L_2.

4.3 Fixed Point Computation

In presence of loops, the abstract interpreter goes around each loop until a fixed point is reached. The number of times a node η inside a loop is processed is bounded above by the maximum number of elements above E along any chain in the lattice $L_1 \bowtie L_2$, where E is the element before the node η when it is first processed. Let $H_L(E)$ denote the maximum number of elements above E in any chain in any lattice L. The following theorem specifies a bound on $H_{L_1 \bowtie L_2}(E)$ in terms of $H_{L_1}(E_1)$ and $H_{L_2}(E_2)$, where E_1 and E_2 are obtained by purification and NO-saturation of E.

$$\begin{aligned}
E_1 &= even(x') \wedge x = x' - 1 \\
E_2 &= positive(x') \wedge x = x' - 1 \\
Q_{L_1}(E_1, \{x'\}) &= odd(x) \\
Q_{L_2}(E_2, \{x'\}) &= true \\
Q_{L_1 \bowtie L_2}(E_1 \wedge E_2, \{x'\}) &= odd(x) \wedge positive(x)
\end{aligned}$$

Figure 8. This figure illustrates the incompleteness of our algorithm for $Q_{L_1 \bowtie L_2}$ in presence of theories that share common function symbols. Since, $E_1 \wedge E_2$ is already purified and NO-saturated, and $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ does not return any definitions, our algorithm for $Q_{L_1 \bowtie L_2}(E_1 \wedge E_2, \{x'\})$ outputs $Q_{L_1}(E_1, \{x'\}) \wedge Q_{L_2}(E_2, \{x'\})$, i.e., $odd(x)$, which is less precise than the result $odd(x) \wedge positive(x)$. The example in this figure is adapted from [6].

THEOREM 6. *Let $E \in D_{L_1 \bowtie L_2}$, $\langle V, E_1^0, E_2^0 \rangle = Purify_{\mathbb{T}_1,\mathbb{T}_2}(E)$, and $\langle E_1, E_2 \rangle = NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^0, E_2^0)$. Then,*

$$H_{L_1 \bowtie L_2}(E) \leq H_{L_1}(E_1) + H_{L_2}(E_2) + |AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E)|$$

where E_1 and E_2 are obtained by purification and NO-saturation of E. $|AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E)|$ denotes the size of the set $AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E)$.

The proof of Theorem 6 is non-trivial and is given in the appendix.

It follows from Theorem 6 that the number of times each node inside a loop is processed is linear in the number of times each node is processed when analysis is done over the lattices L_1 and L_2. However, if the lattice L_1 (or L_2) has infinite chains above some elements, then fixed point may not be reached across the loop in a finite number of steps when the analysis is performed over the lattice $L_1 \bowtie L_2$ or even the lattice L_1 (or lattice L_2). In that case, a widening operation is needed to over-approximate the analysis results at loop headers. A widening operator for the lattice $L_1 \bowtie L_2$ can be constructed from the widening operations for the lattices L_1 and L_2 using the same algorithm that combines join operators (described in Figure 6). The proof of correctness of the construction of the widening operator is same as the soundness proof of the combination of join algorithms.

4.4 Complexity

Let $T_{J_L}(n)$, $T_{Q_L}(n)$, $T_{M_L}(n)$ and $T_{A_{\mathbb{T}}}(n)$ denote the time taken by the join operator J_L, existential quantification operator Q_L, meet operator M_L and $Alternate_{\mathbb{T}}$ operator respectively to operate on inputs of size n. It follows from reduction of $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ operator in terms of the join operators J_{L_1} and J_{L_2} (as described in Section 4.1) that the complexity of $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ operator (for two input elements of size n over theories $\mathbb{T}_1$ and $\mathbb{T}_2$ is at most n times the sum of $T_{J_{L_1}}(n) + T_{J_{L_2}}(n)$. It now follows from Figure 6 and Figure 7 that:

$$\begin{aligned}
T_{J_{L_1 \bowtie L_2}}(n) &= O(T_{J_{L_1}}(n^2) + T_{J_{L_2}}(n^2) + T_{Q_{L_1 \bowtie L_2}}(n^2)) \\
T_{Q_{L_1 \bowtie L_2}}(n) &= O(T_{Q_{L_1}}(n) + T_{Q_{L_2}}(n) + n \times T_{A_{\mathbb{T}_1}}(n) + \\
&\qquad n \times T_{A_{\mathbb{T}_2}}(n) + n \times T_{J_{L_1}}(n) + n \times T_{J_{L_2}}(n)) \\
T_{M_{L_1 \bowtie L_2}}(n) &= O(n)
\end{aligned}$$

Above we use the fact that complexity of $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}$ operator on inputs of size n is $O(n \times T_{J_{L_1}}(n) + n \times T_{J_{L_2}}(n))$, which is also $O(T_{J_{L_1}}(n^2) + T_{J_{L_2}}(n^2))$.

5. Applications of Combination Methodology

The obvious application of our combination methodology to combine abstract interpreters for given lattices is to analyze programs over the logical product combination of those lattices. However, there is another interesting application of this combination methodology. It turns out that certain lattices L can be reduced to strict logical product combination of some other lattices, which may be unrelated to L. If analyses for those other lattices are already known, then the analyses can be combined using our combination methodology to construct an analysis for L [2]. We next describe some lattices L that can be reduced to strict logical product combination of other (unrelated) lattices.

5.1 Commutative Functions

Commutative functions are useful in modeling binary program operators that are commutative, e.g. floating-point addition and multiplication. Note that these floating-point operators should not be abstracted as linear arithmetic operators since they do not obey associativity because of overflow issues.

The lattice of commutative functions can be reduced to the strict logical product combination of the lattice of single unary uninterpreted function and the linear arithmetic lattice. This can be done by using some injective mapping to transform terms that use commutative functions to terms that use a single unary uninterpreted function and linear arithmetic as described below.

Consider the following language of terms that use commutative functions. Here x denotes some variable, while G_i denotes some uninterpreted function.

$$t ::= x \quad | \quad G_i(t_1, t_2)$$

The following mapping M maps these terms that use commutative functions G_i's to terms that use a single uninterpreted function F and linear arithmetic.

$$\begin{array}{rcl} M(x) & = & x \\ M(G_i(t_1, t_2)) & = & F(i + M(t_1) + M(t_2)) \end{array}$$

The following claim implies that the mapping M is injective and equivalence preserving. Hence, this transformation of lattices (obtained by mapping terms) is both sound and complete.

CLAIM 2. *$t_1 = t_2$ iff $M(t_1) = M(t_2)$*

Claim 2 can be proved by induction on structure of term e.

5.2 Uninterpreted Functions

Uninterpreted functions are commonly used to model programming language operators that are otherwise hard to reason about. They are also used to abstract procedure calls with no side-effects.

The lattice of uninterpreted functions can be reduced to the strict logical product of the lattice of a single unary uninterpreted function and the linear arithmetic lattice. For this purpose, we describe a transformation that converts terms involving (potentially multiple) uninterpreted function symbols of any finite arity to terms that use only one unary uninterpreted function symbol.

Consider the following language of terms. Here x denotes some program variable, while G_i^a denotes some uninterpreted function of arity a.

$$t ::= x \quad | \quad G_i^a(t_1, \ldots, t_a)$$

[2] The analysis thus obtained for L is not the most precise one since our reduction is in terms of strict logical product combination, while the combination methodology yields an analysis for the logical product combination, which is less precise than the strict logical product combination.

The following mapping M maps these terms to terms that use a single uninterpreted function F and linear arithmetic.

$$\begin{array}{rcl} M(x) & = & x \\ M(G_i^a(t_1, \ldots, t_a)) & = & F(i + 2^1 M(t_1) + \ldots + 2^a M(t_a)) \end{array}$$

Claim 2 holds for this mapping too. Hence, this transformation of lattices is both sound and complete.

6. Related Work

The problem of combining abstract interpreters for combination of abstract lattices was first considered by Cousot and Cousot [6, 4]. They defined different notions of combination of lattices: the direct product and the more precise reduced product. The notion of reduced product of combination of lattices does not specify any automatic way to construct the abstract interpretation operations for the combined lattice. In this paper, we introduce a new notion of combination of lattices called logical product, which is more powerful than the reduced product. We also describe algorithms to automatically construct the abstract interpretation operations for the logical product lattice from the abstract interpretation operations for the individual logical lattices. Our algorithms can be simplified to yield abstract interpretation operations for the reduced product of the underlying logical lattices.

An abstract lattice can be lifted to its powerset lattice (called disjunctive completion), which is more precise since there is no loss of information in join computations [6]. It is interesting to note that the precision of logical product combination is incomparable with the disjunctive completion of a reduced product combination. The former yields more precise information across assignment nodes since it can represent atomic facts that involve combination of signatures of the individual lattices. The latter yields more precise information across join points since it can represent disjunction of atomic facts (each of which is pure though).

Codish et. al. [2] have studied the problem of automatically generating abstract interpretation for reduced product combination of lattices. However, their work has focused on logic programs, while we are concerned with imperative programs. Also, their combination methodology does not provide any precision guarantees with respect to the reduced product lattice. On the other hand, our combination methodology gives precision guarantees over reduced product (in fact, even over a more precise logical product) combination of a general class of lattices.

Chang and Leino [1] have presented preliminary results on combining a given abstract domain with the domain of uninterpreted symbols, but do not define and prove correctness for their combination. They point out several subtleties that can arise when dealing with the semi-lattice induced by uninterpreted symbol (see also [15]). Our results here consolidate and build a nontrivial framework under which to define and modularly prove correctness of these past approaches.

Lerner, Grove and Chambers have described how to automatically combine dataflow analyses in the context of compiler optimizations [17]. Their technique involves implicit communication between the individual components of a super-analysis based on graph transformations. They have shown that under a certain monotonicity condition, their combination algorithm produces no worse results than running arbitrarily iterated sequences of the individual analyses.

Reps, Sagiv and Yorsh [21] show that an abstract interpreter can be constructed using only a decision procedure (that can also produce *models*) and a join operator. This suggests that abstract interpreter for combination lattices can be constructed *assuming* a combination decision procedure and a join algorithm for the combined lattice. In our work, we make no such assumptions.

Also, unlike our algorithms, the complexity of the procedure for computing meet and SP suggested by Reps et. al. is linear in the height of the lattice, which is usually unbounded in rich domains.

The idea of decomposing a lattice into a reduced product of simpler lattices has been formally studied by Cortesi et. al. [3]. However, the results in Section 5 present nontrivial reductions of some specific lattices into *logical products* of other (unrelated) lattices.

7. Conclusion and Future Work

This paper describes how to automatically combine the power of abstract interpreters in a non-trivial manner. We define a notion of logical product of lattices, which is more precise than the commonly known reduced product. Our combination methodology yields the most precise abstract interpreter for the logical product combination of lattices, when the individual lattices are over theories that are convex, stably infinite, and disjoint. In other cases, our combination methodology acts as an efficient heuristic to combine the power of given abstract interpreters in a non-trivial manner.

The process of analyzing a program using abstract interpretation over a given abstract domain involves two main steps: designing the abstract interpretation operators for the abstract domain, and implementing them. The combination methodology described in this paper helps in modularizing both these steps. A user can focus on designing and/or implementing abstract interpreters for potentially simple domains, and then use our combination methodology to automatically construct abstract interpreter for the combination of those domains, which is more precise than the individual domains.

The concept of logical product suggests several directions for future work. It may be worthwhile to explore logical products and compare them to reduced products in the framework of abstract domains and closures [8, 9]. The conditions under which our combination methodology generates the most precise abstract interpreter over logical product are inherited from the Nelson-Oppen combination result for combining decision procedures. It would be useful to see if our results can be extended to perform a precise analysis for non-convex theories (e.g., the theory of arrays), or combination of non-disjoint theories. There has been a lot of work in the theorem-proving community to extend Nelson-Oppen combination methodology for decision procedures to reason about non-convex or non-disjoint theories that could also be relevant here. On the other hand, it would be interesting to experimentally evaluate the imprecision of our combination methodology in reasoning over combination of non-convex or non-disjoint theories. Since the notion of logical product is more precise than reduced product and direct product, another interesting piece of experimentation would be to compare the cost and precision of an analysis over logical product as opposed to direct product or reduced product.

Acknowledgments

We thank the anonymous reviewers of this paper for their useful and insightful comments.

References

[1] E. Chang and R. Leino. Abstract interpretation with alien expressions and heap structures. In *VMCAI*, volume 3385 of *LNCS*, pages 147–163. Springer, 2005.

[2] M. Codish, A. Mulkers, M. Bruynooghe, M. G. de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. *ACM Transactions on Programming Languages and Systems*, 17(1):28–44, 1995.

[3] A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM Trans. Program. Lang. Syst.*, 19(1):7–47, 1997.

[4] P. Cousot. Forward relational infinitary static analysis, Lecture Notes, Available at http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www. 2005.

[5] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on POPL*, pages 234–252, 1977.

[6] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th ACM Symp. on POPL*, pages 269–282, 1979.

[7] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on POPL*, pages 84–97, 1978.

[8] R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In *Proc. 24th ICALP*, volume 1256 of *LNCS*, pages 771–781, 1997.

[9] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpreters complete. *J. of the ACM*, 47(2):361–416, 2000.

[10] S. Gulwani and G. C. Necula. Discovering affine equalities using random interpretation. In *30th ACM Symposium on POPL*, pages 74–84. ACM, Jan. 2003.

[11] S. Gulwani and G. C. Necula. Global value numbering using random interpretation. In *31st ACM Symposium on POPL*, pages 342–352, Jan. 2004.

[12] S. Gulwani and G. C. Necula. A polynomial-time algorithm for global value numbering. In *11th Static Analysis Symposium*, volume 3148 of *LNCS*, pages 212–227. Springer-Verlag, Aug. 2004.

[13] S. Gulwani and A. Tiwari. Assertion checking over combined abstraction of linear arithmetic and uninterpreted functions. In *15th European Symposium on Programming*, volume 3924 of *LNCS*. Springer, Mar. 2006.

[14] S. Gulwani and A. Tiwari. Combining abstract interpreters. Technical Report MSR-TR-2006-25, Microsoft Research, Mar. 2006.

[15] S. Gulwani, A. Tiwari, and G. C. Necula. Join algorithms for the theory of uninterpreted symbols. In *Conf. on Foundations of Software Tech. and Theor. Comp. Sci., FST&TCS '2004*, volume 3328 of *LNCS*, pages 311–323, 2004.

[16] M. Karr. Affine relationships among variables of a program. In *Acta Informatica*, pages 133–151. Springer, 1976.

[17] S. Lerner, D. Grove, and C. Chambers. Composing dataflow analyses and transformations. In *29th ACM Symposium in POPL*, pages 270–282, 2002.

[18] M. Müller-Olm and H. Seidl. A note on Karr's algorithm. In *ICALP*, pages 1016–1028, 2004.

[19] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, Oct. 1979.

[20] F. Nielson, H. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 2005.

[21] T. W. Reps, S. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In *VMCAI*, volume 2937 of *LNCS*, pages 252–266. Springer, 2004.

A. Proof of Theorem 3

Let $\langle\{z_1,\ldots,z_m\}, E_1', E_2'\rangle = \mathit{Purify}_{\mathbb{T}_1,\mathbb{T}_2}(E')$. Since $E^\ell \preceq_{L_1 \bowtie L_2} E'$, we have that $E^\ell \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} E'$ and $\mathit{AlienTerms}_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq \mathit{Terms}_{\mathbb{T}_1,\mathbb{T}_2}(E^\ell)$. This implies that there exist variables $x_1,\ldots,x_m$ in $V^\ell \cup \mathit{Vars}(E^\ell)$ such that

$$E_1^{\ell 0} \wedge E_2^{\ell 0} \overset{\mathbb{T}_1 \cup \mathbb{T}_2}{\Rightarrow} (E_1' \wedge E_2')[x_1/z_1,\ldots,x_m/z_m]$$

It follows from Property 1 (property of $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}$) that

$$E_1^{\ell 1} \overset{\mathbb{T}_1}{\Rightarrow} E_1'[x_1/z_1,\ldots,x_m/z_m]$$
$$E_2^{\ell 1} \overset{\mathbb{T}_2}{\Rightarrow} E_2'[x_1/z_1,\ldots,x_m/z_m]$$

Similarly, there exist variables $y_1,\ldots,y_m \in V^r \cup Vars(E^r)$ s.t.

$$E_1^{r1} \overset{\mathbb{T}_1}{\Rightarrow} E_1'[y_1/z_1,\ldots,y_m/z_m]$$
$$E_2^{r1} \overset{\mathbb{T}_2}{\Rightarrow} E_2'[y_1/z_1,\ldots,y_m/z_m]$$

Hence,

$$E_1^{\ell 1} \wedge E^{\ell 2} \overset{\mathbb{T}_1}{\Rightarrow} E_1'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m]$$
$$E_2^{\ell 1} \wedge E^{\ell 2} \overset{\mathbb{T}_2}{\Rightarrow} E_2'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m]$$
$$E_1^{r1} \wedge E^{r2} \overset{\mathbb{T}_1}{\Rightarrow} E_1'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m]$$
$$E_2^{r1} \wedge E^{r2} \overset{\mathbb{T}_2}{\Rightarrow} E_2'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m]$$

It follows from completeness of join algorithms J_{L_1} and J_{L_2} that

$$E_1 \overset{\mathbb{T}_1}{\Rightarrow} E_1'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m] \quad (1)$$

$$E_2 \overset{\mathbb{T}_2}{\Rightarrow} E_2'[\langle x_1,y_1\rangle/z_1,\ldots,\langle x_m,y_m\rangle/z_m] \quad (2)$$

Hence, $E_1 \wedge E_2 \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} E'$. It now follows from the completeness of $Q_{L_1 \bowtie L_2}$ that

$$E \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} E'$$

We now show that $AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq Terms_{\mathbb{T}_1,\mathbb{T}_2}(E)$. Consider any $t \in AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E')$. $\exists i$ s.t. $1 \le i \le m$ and

$$E_1' \wedge E_2' \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} z_i = t \quad (3)$$

It follows from Equation 1, Equation 2, and Equation 3 that

$$E_1 \wedge E_2 \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} \langle x_i,y_i\rangle = t$$

This implies that the variable $\langle x_i,y_i\rangle$ does not belong to the set V_2 (on Line 4) in our algorithm of the operator $Q_{L_1 \bowtie L_2}$ when $Q_{L_1 \bowtie L_2}$ is called from within $J_{L_1 \bowtie L_2}$. Hence, $Q_{L_1 \bowtie L_2}$ eliminates variable $\langle x_i,y_i\rangle$ in $E_1 \wedge E_2$ by substitution with some term t'. Note that $E_1 \wedge E_2 \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} t = t'$. The variable z_i (introduced during the purification step) occurs in a non-trivial manner in both E_1' and E_2' and hence it follows from Equation 1 and Equation 2 that the variable $\langle x_i,y_i\rangle$ occurs in both E_1 and E_2. This implies that the term t' occurs in both E_1^3 and E_2^3 (as defined on Lines 7 and 8 in pseudo-code of $Q_{L_1 \bowtie L_2}$) and hence $t' \in Terms_{\mathbb{T}_1,\mathbb{T}_2}(E)$ (note that $E = E_1^3 \wedge E_2^3$). Since $E_1 \wedge E_2 \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} t = t'$, we have that $E \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} t = t'$. Thus, $t \in Terms_{\mathbb{T}_1,\mathbb{T}_2}(E)$.

B. Proof of Theorem 6

Let E' be such that $E \preceq_{L_1 \bowtie L_2} E'$ and $E' \not\preceq_{L_1 \bowtie L_2} E$. Let E_1' and E_2' be obtained by purification and NO-saturation of E'. Since $AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E') \subseteq Terms_{\mathbb{T}_1,\mathbb{T}_2}(E)$, we can assume without loss of generality that E' has been purified using the variables and definitions used to purify E. It suffices to show the following.
(i) $E_1 \preceq_{L_1} E_1'$ and $E_2 \preceq_{L_2} E_2'$.
(ii) Either $E_1' \not\preceq_{L_1} E_1$ or $E_2' \not\preceq_{L_2} E_2$ or $AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E') \subset AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E)$.

We first prove (i). Let `Defs` be the conjunction of definitions required for purification of E. Since $E \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} E'$, we have that:

$$E \wedge \mathtt{Defs} \quad \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} \quad E' \wedge \mathtt{Defs} \quad (4)$$

Note that $E \wedge \mathtt{Defs} \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Longleftrightarrow} E_1 \wedge E_2$, and $E' \wedge \mathtt{Defs} \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} E_1' \wedge E_2'$. Hence, it follows from Eq. 4 that

$$E_1 \wedge E_2 \quad \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} \quad E_1' \wedge E_2' \quad (5)$$

Since E_1 and E_2 are NO-saturated, it follows from Eq. 5 that

$$E_1 \overset{\mathbb{T}_1}{\Rightarrow} E_1' \quad \text{and} \quad E_2 \overset{\mathbb{T}_2}{\Rightarrow} E_2'$$

We now prove (ii). Suppose for the purpose of contradiction that

$$E_1' \preceq_{L_1} E_1 \quad \text{and} \quad E_2' \preceq_{L_2} E_2 \quad (6)$$
$$AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E') \quad = \quad AlienTerms_{\mathbb{T}_1,\mathbb{T}_2}(E) \quad (7)$$

Since $E \wedge \mathtt{Defs} \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Longleftrightarrow} E_1 \wedge E_2$, and $E' \wedge \mathtt{Defs} \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} E_1' \wedge E_2'$, it follows from Equation 6 that

$$E' \wedge \mathtt{Defs} \quad \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} \quad E \wedge \mathtt{Defs} \quad (8)$$

Since `Defs` is simply a conjunction of definitions for variables that do not occur in E' and E, it follows from Equation 8 that

$$E' \quad \overset{\mathbb{T}_1\cup\mathbb{T}_2}{\Rightarrow} \quad E \quad (9)$$

It follows from Equation 7 and Equation 8 that $E' \preceq_{L_1 \bowtie L_2} E$. This is a contradiction.