Supplementary Material for “Tunable Control-Flow Sensitivity For Program Analysis”

Ben Hardekopf, Berkeley Churchill, Vineeth Kashyap, Ben Wiedermann

1 Introduction

This supplement contains a category theoretical formulation of the space of trace abstractions. It contributes the following to our work:

1. The definition of a category of trace abstractions (Section 2)
2. A proof that the products and sums in the paper correspond to the categorical notations of product and coproduct (Section 2)
3. A demonstration that the ordering in Definition 2 of Section 5.2 of the paper corresponds to arrows in our category (Section 3)
4. An explicit construction of the lattice described in Theorem 3 of Section 5.2, along with a proof of this theorem. (Section 3)

2 Category of Trace Abstractions

For the purposes of this supplement a trace abstraction is a pair $(\Theta^\sharp, \tau_{update})$ where $\Theta^\sharp$ is an unspecified finite set and $\tau_{update} : (\Sigma^\sharp \times \Theta^\sharp) \uplus \mathbf{1} \to \Theta^\sharp$. [1] For a trace abstraction $X = (\Theta^\sharp, \tau_{update})$, let Θ_X denote $\Theta^\sharp$, τ_X denote τ_{update} and $\mathbf{1}_X$ denote $\tau_{update}(\mathbf{1})$. These are known as the *underlying set*, *update function* and *starting trace*, respectively. It is often useful in the following to think of a trace abstraction as a finite automaton; in this language $\Theta^\sharp$ is a set of states, τ_{update} as a transition function, $\Sigma^\sharp$ as an input alphabet and $\tau_{update}(\mathbf{1})$ as a starting state of the automaton.

Let X and Y be trace abstractions. An arrow $f : X \to Y$ is a function $f : \Theta_X \to \Theta_Y$ such that

1. $f(\mathbf{1}_X) = f(\mathbf{1}_Y)$.
2. for all $\hat{\varsigma} \in \Sigma^\sharp$ and $x \in \Theta_X$, if $f(x) = y$ then $f(\tau_X(\hat{\varsigma}, x)) = \tau_Y(\hat{\varsigma}, y)$

This is equivalent to saying the following diagram commutes:

$$\begin{tikzcd}
(\Sigma^\sharp \times \Theta_X) \uplus 1 \arrow[r, "id \times f \uplus id"] \arrow[d, "\tau_X"'] & (\Sigma^\sharp \times \Theta_Y) \uplus 1 \arrow[d, "\tau_Y"] \\
X \arrow[r, "f"] & Y
\end{tikzcd}$$

Lemma 1. *This collection of objects and arrows form a category.*

Proof. Composition of arrows corresponds is exactly composition of relations. For each X, the identity relation serves as an identity arrow $id_X : X \to X$. The standard laws hold. □

[1] $\uplus$ is used for disjoint union of sets to disambiguate it from +, which is used for sums in our category, and $\sqcup$ which is traditionally used for lattice join by the abstract interpretation community.

Astute readers will recognize that in the category **Set** of sets and functions, $F(X) = (\Sigma^\sharp \times X) \uplus \mathbf{1}$ is a functor. An arrow $\tau_X : FX \to X$ is known as an *F-algebra*. For a particular F, the collection of F-algebras form a category, where an arrow between $FX \to X$ and $FY \to Y$ is a function $f : X \to Y$ that makes the following diagram commute. This is exactly the category of trace abstractions we describe. We continue to use F to represent this functor in the rest of the supplement.

$$\begin{tikzcd} FX \arrow[r, "Ff"] \arrow[d, "\tau_X"'] & FY \arrow[d, "\tau_Y"] \\ X \arrow[r, "f"] & Y \end{tikzcd}$$

Initial Object The initial object in this category corresponds to the flow-insensitive control flow sensitivity, $\mathbf{1}$, whose underlying set is a singleton. For any trace abstraction X, there is a unique arrow $\mathbf{1} \to X$, given by $\mathbf{1_1} \mapsto \mathbf{1}_X$. The flow-insensitive sensitivity is also the terminal object, so it is known as an *zero object*. This nicely frames the flow-insensitive abstraction as the unique endpoint in our space.

Products. A product of trace abstractions X and Y corresponds to the cartesian product of the underlying sets. The new update function is defined component-wise in terms of τ_X and τ_Y, namely $\tau_{X\times Y} = \tau_X \times \tau_Y$. There are the projection functions $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$ that correspond to set-wise projection of the underlying sets. This satisfies the usual category theoretic definition of product (the details are easy to work out, and are entirely analogous to the proof in **Set**).

Coproducts. *Coproducts*, or *sums* are more interesting; the new trace abstraction is a partitioning of the disjoint union of the two trace abstractions. The equivalence relation $\sim$ is defined on $\Theta_X \uplus \Theta_Y$, as the symmetric, reflexive and transitive closure of the relation inductively generated by the rules (these are the same ones in the paper, but expanded for clarity in the proof):

$$\mathbf{1}_X \ \sim\ \mathbf{1}_Y$$

$$\frac{x \in \Theta_X \qquad y \in \Theta_Y \qquad x \ \sim\ y}{\tau_X(\hat{\varsigma}, x) \ \sim\ \tau_Y(\hat{\varsigma}, y)}$$

$$\frac{x, x' \in \Theta_X \qquad x \ \sim\ x'}{\tau_X(\hat{\varsigma}, x) \ \sim\ \tau_Y(\hat{\varsigma}, x')}$$

$$\frac{y, y' \in \Theta_X \qquad y \ \sim\ y'}{\tau_Y(\hat{\varsigma}, y) \ \sim\ \tau_Y(\hat{\varsigma}, y')}$$

The abstraction $X + Y$ has underlying set $(X \uplus Y)/\sim$, the set of equivalence classes of $X \uplus Y$ under $\sim$. There are inclusion arrow $i_X : X \to X + Y$ and $i_Y : Y \to X + Y$ given by $x \mapsto [x]$ and $y \mapsto [y]$, where $[x]$ denotes the equivalence class of x under $\sim$. The following lemma is a prerequisite to showing that $X + Y$ is a coproduct in the category.

Lemma 2. *Let X, Y, Z be any trace abstractions with $f : X \to Z, g : Y \to Z$. Let $\sim$ be the equivalence relation defined above. Then:*

1. *For all $x, x' \in \Theta_X$, $x \sim x'$ implies $f(x) = f(x')$.*
2. *For all $y, y' \in \Theta_Y$, $y \sim y'$ implies $g(y) = g(y')$.*
3. *For all $x \in \Theta_X$, $y \in \Theta_Y$, $x \sim y$ implies $f(x) = g(y)$.*

Equivalently, the function $(f \uplus g) : \Theta_X \uplus \Theta_Y \to \Theta_Z$ satisfies $(f \uplus g)(u) = (f \uplus g)(v)$ whenever $u \sim v$.

Proof. The pertinent commutative diagram is

$$\begin{tikzcd}
FX \arrow[r, "Ff"] \arrow[d, "\tau_X"'] & FZ \arrow[d, "\tau_Y"'] & FY \arrow[l, "Fg"'] \arrow[d, "\tau_Z"'] \\
X \arrow[r, "f"] & Z & Y \arrow[l, "g"']
\end{tikzcd}$$

It follows from the commutativity of this diagram (equivalently, from the definition of an arrow) that

1. $f(\mathbf{1}_X) = \mathbf{1}_Z = g(\mathbf{1}_Y)$
2. For all $x \in \Theta_X, y \in \Theta_Y, \hat{\varsigma} \in \Sigma^\sharp$, if $f(x) = g(y)$ then $f(\tau_X(\hat{\varsigma}, x)) = g(\tau_Y(\hat{\varsigma}, y))$.
3. For all $x \in \Theta_X, x' \in \Theta_X, \hat{\varsigma} \in \Sigma^\sharp$, if $f(x) = f(x')$ then $f(\tau_X(\hat{\varsigma}, x)) = g(\tau_X(\hat{\varsigma}, x'))$.
4. For all $y \in \Theta_Y, y' \in \Theta_Y, \hat{\varsigma} \in \Sigma^\sharp$, if $g(y) = g(y')$ then $g(\tau_Y(\hat{\varsigma}, y)) = g(\tau_Y(\hat{\varsigma}, y'))$.

The proof is by induction on the rules that generate $\sim$. Before application of any rules, $\sim$ is the empty relation, and this lemma vacuously holds. At each step when a rule is applied to enlarge $\sim$, the corresponding fact from the above list guarantees the lemma still holds. This not only proves the lemma, but also guarantees that the equality relation contains $\sim$, so they are in fact equal. □

Lemma 2 has done the bulk of the work of showing that $X + Y$ is indeed a coproduct.

Theorem 3. *$X + Y$ is the coproduct of X and Y.*

Proof. Suppose X, Y and Z are trace abstractions with $f : X \to Z$ and $g : Y \to Z$. Suppose there exists $u : X + Y \to Z$ such that the standard diagram commutes:

$$\begin{tikzcd}
& Z & \\
X \arrow[ur, "f"] \arrow[r, "i_X"'] & X + Y \arrow[u, "u"'] & Y \arrow[ul, "g"'] \arrow[l, "i_Y"]
\end{tikzcd}$$

Then for any $x \in X$, necessarily $u([x]) = f(x)$; similarly for any $y \in Y$, $u([y]) = g(y)$. This entirely defines u. Lemma 2 guarantees that this function is well defined. If any $w, w' \in X + Y$ belong to the same equivalence class, then $f(w) = f(w') = g(w) = g(w')$. Therefore u must be unique and does exist, which completes the proof. □

3 Ordering on Control Flow Sensitivities

Recall the definition in the paper provided for the preorder. We use the convention from abstract interpretation that in a lattice, precision increases from top to bottom. Therefore, we say that X is more precise than Y when $X \leq Y$.

Definition 4. $X \leq Y$ if there is a relation $R \subset Y \times X$ such that

1. $(\mathbf{1}_Y, \mathbf{1}_X) \in R$
2. $(y, x) \in R$ implies for all $\hat{\varsigma} \in \Sigma^\sharp$, $(\tau_Y(\hat{\varsigma}, y), \tau_X(\hat{\varsigma}, x)) \in R$
3. R is injective, meaning $(y, x) \in R$ and $(y', x) \in R$ implies $y = y'$.

This definition is a kind bisimulation between two state transition systems. In the language of program analysis, it entails that the two kinds of control flow sensitivity operate in lock-step with each other: when one makes a transition, the other makes a corresponding transition. The injectivity requirement guarantees that one is at least the size of the other. This $\leq$ is a preorder. We say two control flow sensitivities X and Y are equivalent if $X \leq Y$ and $Y \leq X$, also denoted $X \equiv Y$. When $X \equiv Y$ it means the control flow sensitivities behave exactly the same way: neither is more or less precise than the other. Using $\equiv$ as an equivalence relation, we implicitly lift $\leq$ to be a partial order on the equivalence classes.

We present the definition this way to make the intuition about the order clear without category theory. Observe that if $X \leq Y$ then the corresponding relation is the inverse of an arrow $X \to Y$ (and vice-versa). This makes it very easy to reason about the ordering using this category. Namely, $X \leq Y$ if and only if there is an arrow $X \to Y$. Finally, we provide the proof that coproducts and products are the join and meet operators of this partial order.

Theorem 5. *Control flow sensitivities form a lattice up to equivalence using this order. The join operator corresponds to coproducts and the meet operator corresponds to products.*

Proof. Only the proof for coproducts is provided; the proof for products is identical. Suppose X and Y are trace abstractions. There are functions $i_X : X \to X + Y$ and $i_Y : Y \to X + Y$. So, $X \leq X + Y$ and $Y \leq X + Y$. Suppose there is some Z such that $X \leq Z$ and $Y \leq Z$. Then there exist arrows such that the following diagram holds,

$$\begin{tikzcd} & Z & \\ X \arrow[ur, "f"] \arrow[r, "i_X"'] & X + Y & Y \arrow[ul, "g"'] \arrow[l, "i_Y"] \end{tikzcd}$$

By definition of coproduct, there must exist a unique function $u : X + Y \to Z$, so $X + Y \leq Z$. This establishes $X + Y$ as the unique least upper bound for X and Y. □

In conclusion, we wrap up our results with a summarizing theorem:

Theorem 6. *The space of trace abstractions forms a category and (up to an equivalence relation) a lattice, where meet and join are given by coproducts and products, respectively.*

The takeaway is that given implementations of two trace abstractions A and B we can programmatically compute $A + B$ and $A \times B$, and use them in a program analysis. All of the proofs follow immediately from elementary results in category and order theory, inspection of commutative diagrams, and the definition of the category we use.

Future Work. Category theory provides many ways to construct new objects from old. The authors hope that this work can be expanded and give rise to new constructions for control-flow sensitivities.