


# Information Flow Is Linear Refinement of Constancy

View metadata, citation and similar papers at [core.ac.uk](http://core.ac.uk)

brought to you by  CORE

provided by Catalogo dei prodotti della ricerca

Dipartimento di Informatica, Università di Verona, Italy  
fausto.spoto@univr.it

**Abstract.** Detecting information flows inside a program is useful to check non-interference of program variables, an important aspect of software security. Information flows have been computed in the past by using abstract interpretation over an abstract domain IF which expresses sets of flows. In this paper we reconstruct IF as the *linear refinement*  $C \rightarrow C$  of a basic domain C expressing *constancy* of program variables. This is important since we also show that  $C \rightarrow C$ , and hence IF, is closed *w.r.t.* linear refinement, and is hence *optimal* and *condensing*. Then a compositional, input-independent static analysis over IF has the same precision of a non-compositional, input-driven analysis. Moreover, we show that  $C \rightarrow C$  has a natural representation in terms of Boolean formulas, efficiently implementable through binary decision diagrams.

## 1 Introduction

Language-based security is recognised as an important aspect of modern programming languages design and implementation [11]. One of its aspects is *non-interference*, which determines the pairs of program variables that do not affect each other's values during the execution of a program. From non-interference it is then possible to study the confinement of confidential information injected in the program through some input variables. Non-interference is often implemented above an information-flow analysis, which tracks the flows of information in a program [15,12,3,11,6].

Information flows in a program can be computed through abstract interpretation [4] by using an abstract domain, that we call IF in this paper, which models sets of flows [11,6]. Abstract interpretation consists in executing the program over the description of the concrete data as provided by IF. Correctness states that if a program features a flow, then it must be included in the description that the analysis computes. The domain IF has been implemented by using Boolean formulas [6] to represent sets of flows. This leads to an efficient analysis [7] which uses binary decision diagrams [2] to implement such formulas. Moreover, that analysis is *input-independent i.e.*, it is performed only once, without any assumption on the input provided to the program. The input variables containing confidential information are specified *after* the analysis is performed. An *input-driven* analysis, instead, would require the input to be available *before*

the analysis, so that it must be re-executed for each different input. Hence it is not possible to analyse a library independently from the applications that use it.

In this paper we show that **IF** coincides with the *linear refinement*  $C \rightarrow C$  of an abstract domain  $C$  which expresses *constancy* of program variables *i.e.*, the set of variables that are definitely bound to a constant value in a given program point. Linear refinement [10] is a formal technique which adds input/output relational information to an abstract domain. In our case, the added relational information over  $C$  corresponds to the flows of information between variables.

This result is important since

- it shows that an independently developed abstract domain such as **IF** can be reconstructed through a methodological technique such as linear refinement;
- we later prove that  $C \rightarrow C$  is closed *w.r.t.* linear refinement. This entails that  $C \rightarrow C$  (and hence **IF**) is an *optimal* and *condensing* abstract domain [9]. This means that **IF** is the minimal abstract domain which models information flows and all relational information between them (optimality) and that it can be used for a compositional, input-independent static analysis without sacrificing precision *w.r.t.* a non-compositional, input-driven static analysis (condensing). None of these properties was known before for **IF**;
- we finally show that the elements of  $C \rightarrow C$ , and hence of **IF**, have a *natural* representation in terms of Boolean formulas. This formally justifies the use of Boolean formulas to implement **IF** [6].

The rest of this paper is organised as follows. Section 2 presents the preliminaries and defines  $C$ . Section 3 formalises the abstract domain **IF**. Section 4 shows that  $\mathbf{IF} = C \rightarrow C$ . Section 5 proves that  $C \rightarrow C$  is closed *w.r.t.* linear refinement, and is hence optimal and condensing. Section 6 provides a representation of the elements of  $C \rightarrow C$  in terms of Boolean formulas. Section 7 concludes.

## 2 Preliminaries

### 2.1 Functions and Ordered Sets

A total (partial) function  $f$  is denoted by  $\mapsto (\rightarrow)$ . The *domain* of  $f$  is  $dom(f)$ . We denote by  $[v_1 \mapsto t_1, \dots, v_n \mapsto t_n]$  the function  $f$  where  $dom(f) = \{v_1, \dots, v_n\}$  and  $f(v_i) = t_i$  for  $i = 1, \dots, n$ . Its *update* is  $f[w_1 \mapsto d_1, \dots, w_m \mapsto d_m]$ , where the domain may be enlarged. By  $f|_s$  ( $f|_{-s}$ ) we denote the *restriction* of  $f$  to  $s \subseteq dom(f)$  (to  $dom(f) \setminus s$ ). The composition  $f \circ g$  of functions  $f$  and  $g$  is such that  $(f \circ g)(x) = g(f(x))$ . A *poset* is a set  $S$  with a reflexive, transitive and antisymmetric relation  $\leq$ . An *upper* (respectively, *lower*) *bound* of  $S' \subseteq S$  is an element  $u \in S$  such that  $u' \leq u$  (respectively,  $u' \geq u$ ) for every  $u' \in S'$ . A *complete lattice* is a poset where *least* upper bounds ( $\sqcup$ ) and *greatest* lower bounds ( $\sqcap$ ) always exist. The top and bottom elements of a lattice are denoted by  $\top$  and  $\perp$ , respectively.

### 2.2 Denotations

We model the *state* of an interpreter of a computer program at a given program point as a function from the variables in scope to their *values*. We consider

integers as values, but any other domain of values would do. Our state can be seen as the activation frame on top of the activation stack of the interpreter. Since in this paper we use a denotational semantics of programs [16], we do not need to model the whole activation stack. Instead, we assume that procedure calls are resolved by plugging the meaning or *interpretation* of a procedure in the calling point. This is standard in denotational semantics, and has been used for years in the semantics of logic programs [1].

**Definition 1 (State).** *Let  $V$  be a finite set of variables (this will be assumed in the rest of the paper). A state over  $V$  is a total function from  $V$  into integer values. The set of states over  $V$  is  $\Sigma_V$ , where  $V$  is usually omitted.*

The set  $V$  contains only the variables in scope in the program point under analysis.

*Example 1.* An example of state  $\sigma \in \Sigma$  is such that  $\sigma(v) = 3$  for each  $v \in V$ .

A denotational semantics associates a *denotation* to each piece of code *i.e.*, a function from input states to output states. Possible divergence is modelled by using partial functions as denotations.

**Definition 2 (Denotation).** *A denotation over  $V$  is a partial function  $\delta : \Sigma_V \rightarrow \Sigma_V$ . The set of denotations is  $\Delta_V$ , where  $V$  is usually omitted. Let  $\sigma \in \Sigma_V$ . If  $v \in V$  and  $\delta(\sigma)$  is not defined, then we let  $\delta(\sigma)(v) = \text{undef}$ .*

A *denotational semantics* is a *compositional* (*i.e.*, inductive) definition of the denotations of each language construct. This definition is irrelevant here, since expressivity and precision of a static analysis are domain-related issues [4]. Hence we give complete freedom to the language designer, so that for instance we impose no constraint on the denotations of Definition 2. The interested reader can find in [16] an example of denotational semantics.

*Example 2.* The denotation for the assignment  $y := x+1$  is  $\delta_1$  such that  $\delta_1(\sigma) = \sigma[y \mapsto \sigma(x) + 1]$  for all  $\sigma \in \Sigma$ . That is, the successor of the input value of  $x$  is stored in the output value of  $y$ . The other variables are not modified.

*Example 3.* The denotation of the assignment  $x := 4$  is  $\delta_2$  such that  $\delta_2(\sigma) = \sigma[x \mapsto 4]$  for all  $\sigma \in \Sigma$ . That is, the output value of  $x$  is constantly bound to 4. The other variables are not modified.

*Example 4.* The denotation of `if  $y = 0$  then  $x := 4$  else while true do skip` is  $\delta_4$ , *compositionally* defined as

$$\delta_4(\sigma) = \begin{cases} \delta_2(\sigma) & \text{if } \sigma(y) = 0 \\ \delta_3(\sigma) & \text{if } \sigma(y) \neq 0, \end{cases}$$

where  $\delta_2$  is the denotation of  $x := 4$  (Example 3) and  $\delta_3$  is the denotation of `while true do skip`, which is *always* undefined.

*Example 5.* The denotation of  $x := 4; y := x+1$  is the functional composition  $\delta_2 \circ \delta_1$  (Examples 3 and 2). In general,  $\circ$  is the semantical counterpart of the sequential composition of commands.

*Constancy* is a property of denotations. Namely, a variable  $v$  is constant in a denotation  $\delta$  when  $\delta$  always binds  $v$  to a given value.

**Definition 3.** Let  $\delta \in \Delta$ . The set of variables which are constant in  $\delta$  is

$$const(\delta) = \{v \in V \mid \text{for all } \sigma_1, \sigma_2 \in \Sigma \text{ we have } \delta(\sigma_1)(v) = \delta(\sigma_2)(v)\}.$$

*Example 6.* The denotation  $\delta_1$  of Example 2 copies  $x+1$  into  $y$ . Hence  $const(\delta_1) = \emptyset$ . The denotation  $\delta_2$  of Example 3 binds  $x$  to 4. Then  $const(\delta_2) = \{x\}$ .

Constancy is closed *w.r.t.* composition of denotations. Namely, for any  $\delta, \bar{\delta} \in \Delta$  and  $v \in V$ , if  $v \in const(\delta)$  then  $v \in const(\bar{\delta} \circ \delta)$ .

### 2.3 Abstract Domains and Abstract Interpretation

Let  $C$  be a complete lattice playing the role of the *concrete* domain. For instance, in this paper  $C$  will be the powerset  $\wp(\Delta)$  of the *concrete* denotations of Subsection 2.2. Each element of  $C$  is an *abstract property*. For instance, the set of concrete denotations which bind  $x$  to 4 is an element of  $\wp(\Delta)$  expressing the property: “ $x$  holds 4 in the output of the denotation”. An *abstract domain*  $A$  is a collection of abstract properties *i.e.*, a subset of  $C$ .

*Example 7 (The Abstract Domain C).* Let us use  $\wp(\Delta_V)$  as concrete domain and let  $\mathbf{v}_1 \cdots \mathbf{v}_n = \{\delta \in \Delta_V \mid v_i \in const(\delta) \text{ for } 1 \leq i \leq n\}$ . An abstract domain of  $\wp(\Delta_V)$  is

$$C_V = \{\mathbf{v}_1 \cdots \mathbf{v}_n \mid \{v_1, \dots, v_n\} \subseteq V\}.$$

It expresses the properties of *being constant* for a set of variables in a denotation. Its top element is  $\emptyset$ . We will usually omit  $V$  in  $C_V$ . From Example 6 we conclude that  $\delta_1 \in \emptyset$  and  $\delta_2 \in \mathbf{x}$ . However,  $\delta_2 \notin \mathbf{xy}$  since  $y$  is not constant in  $\delta_2$  (Example 3).

Abstract interpretation theory [4] requires  $A$  to be meet-closed, which guarantees the existence in  $A$  of a *best approximation* for each element of  $C$ . That is,  $A$  must be a *Moore family* of  $C$  *i.e.*, a complete meet-sublattice of  $C$  (for any  $Y \subseteq A$  we have  $\sqcap_C Y \in A$ ). Note that  $A$  is not, in general, a complete sublattice of  $C$ , since the join  $\sqcup_A$  might be different from  $\sqcup_C$ .

*Example 8.* The set  $C$  of Example 7 is closed *w.r.t.* intersection *i.e.*, the  $\cap$  operation on  $\wp(\Delta)$ . Hence  $C$  deserves the name of *abstract domain*. Namely,  $(\mathbf{v}_1 \cdots \mathbf{v}_n) \cap (\mathbf{w}_1 \cdots \mathbf{w}_m) = \mathbf{x}_1 \cdots \mathbf{x}_p$  where  $\{x_1, \dots, x_p\} = \{x \mid x \in \{v_1, \dots, v_n\} \text{ and } x \in \{w_1, \dots, w_m\}\}$ .

For any  $X \subseteq C$ , we denote by  $\lambda X = \{\sqcap_C I \mid I \subseteq X\}$  the *Moore closure* of  $X$  *i.e.*, the least Moore family of  $C$  containing  $X$ . Hence the operation  $\lambda$  *constructs* the smallest abstract domain which includes the set of properties  $X$ .

*Example 9.* We write the set of denotations where  $x$  is constant as  $\mathbf{x} = \{\delta \in \Delta \mid x \in \text{const}(\delta)\}$ . The abstract domain of Example 7 can be constructed as  $C_V = \lambda\{\mathbf{x} \mid x \in V\}$ . We write the elements of  $C$  as  $\mathbf{v}_1 \cdots \mathbf{v}_n$ , standing for  $\cap\{\mathbf{v}_i \mid 1 \leq i \leq n\}$ . If  $us \subseteq V$  then by  $\mathbf{vs}$  we mean  $\cap\{\mathbf{v} \mid v \in us\}$ .

Once an abstract domain is defined, abstract interpretation theory provides the abstract semantics induced by each given concrete semantics. Hence, from a theoretical point of view, the abstract domain is an exhaustive definition of an abstract semantics for a programming language, which can then be implemented and used for static analysis. For this reason, and for space concerns, we concentrate in this paper on abstract domains only, without any consideration on the induced abstract semantics.

### 2.4 Linear Refinement

The definition of an appropriate abstract domain for a static analysis is not in general easy. Although a *basic* abstract domain  $A$  can be immediately constructed ( $\lambda$ ) from the abstract properties one wants to model, there is no guarantee that the induced abstract semantics is precise enough to be useful. The intuition and experience of the abstract domain designer helps in determining what  $A$  is missing in order to improve its precision. In alternative, there are more methodological techniques which *refine*  $A$  to get a more precise domain.

*Reduced product* [5] allows one to refine two abstract domains  $A_1$  and  $A_2$  into an abstract domain  $A_1 \sqcap A_2 = \lambda(A_1 \cup A_2)$  which expresses the conjunction of properties of  $A_1$  and  $A_2$ .

*Linear refinement* [10] is another domain refinement operator. It allows one to enrich an abstract domain with information relative to the *propagation* of the abstract properties before and after the application of a concrete operator  $\boxtimes$ . It requires the concrete domain  $C$  to be a *quantale w.r.t.  $\boxtimes$  i.e.*,

1.  $C$  must be a complete lattice;
2.  $\boxtimes : C \times C \rightarrow C$  must be (in general partial and) associative;
3. for any  $a \in C$  and  $\{b_i\}_{i \in I} \subseteq C$  with  $I \subseteq \mathbb{N}$  we must have  $a \boxtimes (\sqcup_{i \in I} b_i) = \sqcup_{i \in I} \{a \boxtimes b_i\}$  and  $(\sqcup_{i \in I} b_i) \boxtimes a = \sqcup_{i \in I} \{b_i \boxtimes a\}$ .

For instance, the complete lattice  $\wp(\Delta)$ , ordered by set-inclusion, is a quantale *w.r.t.* the composition operator  $\circ$ , extended to sets of denotations as  $d_1 \circ d_2 = \{\delta_1 \circ \delta_2 \mid \delta_1 \in d_1 \text{ and } \delta_2 \in d_2\}$ .

Let  $a, b \in C$ . The abstract property  $a \rightarrow^{\boxtimes} b$  which *transforms* every element of  $a$  into an element of  $b$  is

$$a \rightarrow^{\boxtimes} b = \bigsqcup_C \{c \in C \mid \text{if } a \boxtimes c \text{ is defined then } a \boxtimes c \leq_C b\}.$$

Given  $a \in C$ ,  $I \subseteq \mathbb{N}$  and  $\{b_i\}_{i \in I} \subseteq C$ , we have  $a \rightarrow^{\boxtimes} (\cap_{i \in I} b_i) = \cap_{i \in I} (a \rightarrow^{\boxtimes} b_i)$ .

**Definition 4 (Linear Refinement).** *The (forward) linear refinement of an abstract domain  $A_1 \subseteq C$  w.r.t. another abstract domain  $A_2 \subseteq C$  is the abstract domain  $A_1 \rightarrow^{\boxtimes} A_2 = \lambda\{a \rightarrow^{\boxtimes} b \mid a \in A_1 \text{ and } b \in A_2\}$ . That is, it collects all possible arrows between elements of  $A_1$  and elements of  $A_2$ .*

The following results hold [10]

1.  $\rightarrow_{\boxtimes}$  is argument-wise monotonic;
2.  $A_1 \rightarrow_{\boxtimes} (A_2 \rightarrow_{\boxtimes} A_3) = (A_1 \rightarrow_{\boxtimes} A_2) \rightarrow_{\boxtimes} A_3$ , so parentheses are not relevant;
3.  $A_1 \rightarrow_{\boxtimes} A_2 \rightarrow_{\boxtimes} A_3 = (A_1 \sqcap A_2) \rightarrow_{\boxtimes} A_3$ , where  $\sqcap$  is the reduced product.

We now instantiate  $\rightarrow_{\boxtimes}$  over the quantale  $\langle \wp(\Delta), \circ \rangle$ . The intuition under the choice of  $\circ$  for  $\boxtimes$  is that the denotational semantics of an imperative program is defined by *composing* smaller denotations to form larger denotations [16]. Hence we must refine the composition operation if we want to improve the precision of the abstractions of  $\wp(\Delta)$ .

We first provide an explicit definition for  $\rightarrow^\circ$ .

**Proposition 1.** *Let  $d_1, d_2 \subseteq \Delta$ . Then  $d_1 \rightarrow^\circ d_2 = \{\delta \in \Delta \mid \text{for every } \bar{\delta} \in d_1 \text{ we have } \bar{\delta} \circ \delta \in d_2\}$ .*

*Proof.*

$$\begin{aligned}
 d_1 \rightarrow^\circ d_2 &= \bigcup \{d \in \wp(\Delta) \mid \text{if } d_1 \circ d \text{ is defined then } d_1 \circ d \subseteq d_2\} \\
 &= \bigcup \{d \in \wp(\Delta) \mid d_1 \circ d \subseteq d_2\} \\
 &= \bigcup \{d \in \wp(\Delta) \mid \{\bar{\delta} \circ \delta \mid \bar{\delta} \in d_1 \text{ and } \delta \in d\} \subseteq d_2\} \\
 &= \{\delta \in \Delta \mid \{\bar{\delta} \circ \delta \mid \bar{\delta} \in d_1\} \subseteq d_2\} \\
 &= \{\delta \in \Delta \mid \text{for every } \bar{\delta} \in d_1 \text{ we have } \bar{\delta} \circ \delta \in d_2\}.
 \end{aligned}$$

The intuition behind  $d_1 \rightarrow^\circ d_2$  is that it is the set of denotations that when composed with a denotation in  $d_1$  become a denotation in  $d_2$ .

*Example 10.* Consider the abstract domain  $\mathbf{C}$  of Example 7 and its two elements  $\mathbf{x}$  and  $\mathbf{y}$ . The denotation  $\delta_1$  of Example 2 belongs to  $\mathbf{x} \rightarrow^\circ \mathbf{y}$  since  $\delta_1$  stores the input value of  $x$  plus 1 in the output value of  $y$ , so that if  $x$  is constant in  $\delta_1$ 's input then  $y$  is constant in  $\delta_1$ 's output.

From now on, we will omit  $\circ$  in  $\rightarrow^\circ$ .

### 3 A Classical Domain for Information Flow Analysis

We present here a traditional abstract domain for information flow analysis. It expresses which *termination-sensitive* flows [3] are allowed in a denotation.

**Definition 5 (Information-Flow).** *Let  $\delta \in \Delta$  and  $x, y \in V$ . We say that  $\delta$  features an information flow from  $x$  to  $y$  [11] if there exist  $\sigma_1, \sigma_2 \in \Sigma$  such that*

1.  $\sigma_1|_{V \setminus x} = \sigma_2|_{V \setminus x}$  ( $\sigma_1$  and  $\sigma_2$  agree on  $x$ );
2.  $\delta(\sigma_1)(y) \neq \delta(\sigma_2)(y)$  (the input value of  $x$  affects the output value of  $y$ ).

Definition 5 entails that  $\sigma_1(x) \neq \sigma_2(x)$ . Moreover, if exactly one between  $\delta(\sigma_1)$  and  $\delta(\sigma_2)$  is defined, then by Definition 2 the condition  $\delta(\sigma_1)(y) \neq \delta(\sigma_2)(y)$  holds. This is why Definition 5 formalises termination-sensitive information flows.

*Example 11.* The denotation  $\delta_1$  of Example 2 is such that  $\delta_1(\sigma) = \sigma[y \mapsto \sigma(x) + 1]$  for every  $\sigma \in \Sigma$ . Let  $\sigma_1$  and  $\sigma_2$  be such that  $\sigma_1(v) = 0$  for every  $v \in V$ ,  $\sigma_2(x) = 1$  and  $\sigma_2(v) = 0$  for every  $v \in V \setminus x$ . We have  $\sigma_1|_{V \setminus x} = \sigma_2|_{V \setminus x}$  and  $\delta_1(\sigma_1)(y) = 1 \neq 2 = \delta_1(\sigma_2)(y)$ . Then  $\delta_1$  features a flow from  $x$  to  $y$ . Moreover,  $\delta_1(\sigma_1)(x) = 0$  and  $\delta_1(\sigma_2)(x) = 1$ . Then  $\delta_1$  features a flow from  $x$  to  $x$ . These are both *explicit* flows [11] *i.e.*, generated by copying input values into output values in a denotation. They are the only flows featured by  $\delta_1$ . For instance,  $\delta_1$  does not feature any flow from  $y$  to  $y$ , since for every  $\sigma_1, \sigma_2 \in \Sigma$  such that  $\sigma_1|_{V \setminus y} = \sigma_2|_{V \setminus y}$  we have  $\delta_1(\sigma_1)(y) = \sigma_1(x) + 1 = \sigma_2(x) + 1 = \delta_1(\sigma_2)(y)$ .

*Example 12.* The denotation  $\delta_4$  of Example 4 features a flow from  $y$  to  $x$ . Namely, take  $\sigma_1$  and  $\sigma_2$  such that  $\sigma_1(v) = 0$  for every  $v \in V$ ,  $\sigma_2(v) = 0$  for every  $v \in V \setminus y$  and  $\sigma_2(y) = 1$ . We have  $\sigma_1|_{V \setminus y} = \sigma_2|_{V \setminus y}$ ,  $\delta_4(\sigma_1)(x) = 4 \neq \text{undef} = \delta_4(\sigma_2)(x)$ . Since we consider termination-sensitive flows, the denotation  $\delta_4$  actually features a flow from  $y$  to *any* variable  $v \in V$ , since the initial value of  $y$  determines the termination of the conditional statement in Example 4. These flows are called *implicit* [11] since they arise from the conditional execution of program statements on the basis of the initial value of some variables.

The abstract domain for information flow analysis is the powerset of the set of flows. Each abstract element expresses which flows a denotation can feature.

**Definition 6 (Abstract Domain IF).** Let  $x_i, y_i \in V$  for  $i = 1, \dots, n$ . We define

$$x_1 \rightsquigarrow y_1, \dots, x_n \rightsquigarrow y_n = \left\{ \delta \in \Delta_V \mid \begin{array}{l} \text{if } \delta \text{ features a flow from } x \text{ to } y \text{ then} \\ \text{there exists } i \text{ such that } x \equiv x_i \text{ and } y \equiv y_i \end{array} \right\}.$$

The abstract domain for information flow analysis is

$$\text{IF}_V = \{x_1 \rightsquigarrow y_1, \dots, x_n \rightsquigarrow y_n \mid n \geq 0 \text{ and } x_i, y_i \in V \text{ for every } i = 1, \dots, n\}$$

where  $V$  is usually omitted. It is ordered by inverse set-inclusion.

Each element of IF is a set of denotations. In order to justify the name of *abstract domain* for IF, we must prove that the set of its elements is closed by intersection.

**Proposition 2.** The set IF is a Moore family of  $\wp(\Delta)$ .

*Proof.* Let  $f^i = x_1^i \rightsquigarrow y_1^i, \dots, x_{n^i}^i \rightsquigarrow y_{n^i}^i \in \text{IF}$  with  $I \subseteq \mathbb{N}$  and  $i \in I$ . We prove that  $X = \{x \rightsquigarrow y \mid x \rightsquigarrow y \in f^i \text{ for all } i \in \mathbb{N}\}$  (which belongs to IF) is their intersection. We have  $\delta \in \bigcap_{i \in I} f^i$  if and only if  $\delta \in f^i$  for each  $i \in I$ , if and only if whenever  $\delta$  features a flow from  $x$  to  $y$  then  $x \rightsquigarrow y \in f^i$  for each  $i \in I$ , if and only if whenever  $\delta$  features a flow from  $x$  to  $y$  then  $x \rightsquigarrow y \subseteq X$ , if and only if  $\delta \in X$ .

Figure 1 shows the abstract domain  $\text{IF}_{\{x, y\}}$ . The top of the domain allows denotations to feature any flow, and hence coincides with  $\Delta$ .

*Example 13.* Assume  $V = \{x, y\}$ . The denotation  $\delta_1$  of Example 2 belongs to  $x \rightsquigarrow x, x \rightsquigarrow y$  since it only features flows from  $x$  to  $y$  and from  $x$  to  $x$  (Example 11). It also belongs to the upper bound  $x \rightsquigarrow x, x \rightsquigarrow y, y \rightsquigarrow y$ . However,  $\delta_1$  does not belong to  $x \rightsquigarrow x$  since  $\delta_1$  features a flow from  $x$  to  $y$  (Example 11), not allowed in  $x \rightsquigarrow x$ .

## 4 The Linear Refinement $\mathbf{C} \rightarrow \mathbf{C}$

Example 7 defines a basic domain for constancy  $\mathbf{C}$  which models the set of variables which are constant in the output of a denotation. Here, we linearly refine  $\mathbf{C}$  into  $\mathbf{C} \rightarrow \mathbf{C}$ , which is a new abstract domain for *constancy propagation*. Then we show that  $\mathbf{IF}$  and  $\mathbf{C} \rightarrow \mathbf{C}$  coincide.

The following result shows that  $\mathbf{C} \rightarrow \mathbf{C}$  includes  $\mathbf{C}$ .

**Proposition 3.** *We have  $\mathbf{C} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ . If  $\#V \geq 2$ , the inclusion is strict.*

*Proof.* Since  $\emptyset \in \mathbf{C}$  and  $\emptyset = \wp(\Delta)$ , then for every  $v \in V$  we have  $\emptyset \rightarrow \mathbf{v} \in \mathbf{C} \rightarrow \mathbf{C}$ . If we show that  $\emptyset \rightarrow \mathbf{v} = \mathbf{v}$ , we conclude that  $\mathbf{C} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ . Let  $\delta \in \emptyset \rightarrow \mathbf{v}$  and  $\iota$  be the identity denotation, such that  $\iota(\sigma) = \sigma$  for every  $\sigma \in \Sigma$ . We have  $\iota \in \emptyset = \wp(\Delta)$ , so that  $\iota \circ \delta = \delta \in \mathbf{v}$ . Conversely, let  $\delta \in \mathbf{v}$ . Constancy is closed by composition, so  $\bar{\delta} \circ \delta \in \mathbf{v}$  for every  $\bar{\delta} \in \wp(\Delta) = \emptyset$ . Hence  $\delta \in \emptyset \rightarrow \mathbf{v}$ .

To prove the strict inclusion, let  $x, y \in V$ ,  $x \neq y$ . Let  $\iota$  be the identity denotation. Since no variable is constant in  $\iota$ , we have  $\iota \in \emptyset$  and  $\iota \notin c$  for all  $c \in \mathbf{C} \setminus \{\emptyset\}$ . We have  $\iota \in \mathbf{x} \rightarrow \mathbf{x}$ . This is because for all  $\bar{\delta} \in \mathbf{x}$  we have  $\bar{\delta} \circ \iota = \bar{\delta} \in \mathbf{x}$ . To prove that  $\mathbf{C} \subset \mathbf{C} \rightarrow \mathbf{C}$  is then enough to show that  $\mathbf{x} \rightarrow \mathbf{x} \neq \emptyset$ . Consider  $\delta$  such that  $\delta(\sigma) = \sigma[x \mapsto \sigma(y)]$ . We have  $\delta \in \emptyset$  since no variable is constant in  $\delta$ . But  $\delta \notin \mathbf{x} \rightarrow \mathbf{x}$ , since if we take  $\bar{\delta} \in \mathbf{x}$  such that  $\bar{\delta}(\sigma) = \sigma[x \mapsto 0]$  we have  $\bar{\delta} \circ \delta = \delta \notin \mathbf{x}$  (no variable is constant in  $\delta$ ).

The following lemma states that if a denotation does not feature any flow from a set of variables  $V \setminus S$  into a given variable  $y$ , then  $y$ 's value in the output of the denotation depends only on the input values of the variables in  $S$ .

**Lemma 1.** *Let  $\sigma_1, \sigma_2 \in \Sigma$ ,  $y \in V$ ,  $S \subseteq V$  and  $\delta \in \Delta$  which does not feature any flow  $v \rightsquigarrow y$  with  $v \in V \setminus S$ . Then  $\sigma_1|_S = \sigma_2|_S$  entails  $\delta(\sigma_1)(y) = \delta(\sigma_2)(y)$ .*

*Proof.* Let  $V \setminus S = \{v_1, \dots, v_n\}$ . Define  $\sigma_1^0 = \sigma_1$ ,  $\sigma_2^0 = \sigma_2$  and  $\sigma_1^i = \sigma_1^{i-1}[v_i \mapsto \max(\sigma_1(v_i), \sigma_2(v_i))]$ ,  $\sigma_2^i = \sigma_2^{i-1}[v_i \mapsto \max(\sigma_1(v_i), \sigma_2(v_i))]$  for  $1 \leq i \leq n$ . Note that whether  $\sigma_1^i = \sigma_1^{i-1}$  or they differ at  $v_i \in V \setminus S$  only. The same holds for  $\sigma_2^i$  and  $\sigma_2^{i-1}$ . Since  $\delta$  does not feature any flow  $v_i \rightsquigarrow y$ , in both cases we have  $\delta(\sigma_1^i) = \delta(\sigma_1^{i-1})$  and  $\delta(\sigma_2^i) = \delta(\sigma_2^{i-1})$ . Moreover,  $\sigma_1^n = \sigma_2^n$ . Hence  $\delta(\sigma_1) = \delta(\sigma_1^0) = \delta(\sigma_1^1) = \dots = \delta(\sigma_1^n) = \delta(\sigma_2^n) = \dots = \delta(\sigma_2^1) = \delta(\sigma_2^0) = \delta(\sigma_2)$ .

We can now state that  $\mathbf{IF}$  coincides with  $\mathbf{C} \rightarrow \mathbf{C}$ . We first prove that  $\mathbf{IF} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ , by *implementing* each element of  $\mathbf{IF}$  through an element of  $\mathbf{C} \rightarrow \mathbf{C}$ .

**Lemma 2.** *Let  $f = x_1 \rightsquigarrow y_1, \dots, x_n \rightsquigarrow y_n \in \mathbf{IF}$ . We have*

$$f = \cap \{ \mathbf{S}(\mathbf{y}) \rightarrow \mathbf{y} \mid y \in V \text{ and } S(y) = \{x_i \mid x_i \rightsquigarrow y \in f\} \}.$$

*Proof.* Let  $\delta \in x_1 \rightsquigarrow y_1, \dots, x_n \rightsquigarrow y_n$ . We prove that  $\delta \in \mathbf{S}(\mathbf{y}) \rightarrow \mathbf{y}$  for each  $y \in V$ . Let  $\bar{\delta} \in \mathbf{S}(\mathbf{y})$  and  $\sigma_1, \sigma_2 \in \Sigma$ . We have  $\bar{\delta}(\sigma_1)|_{S(y)} = \bar{\delta}(\sigma_2)|_{S(y)}$ . Moreover,  $\delta$  does not feature any flow  $v \rightsquigarrow y$  with  $v \notin S(y)$ . By Lemma 1 we have  $(\bar{\delta} \circ \delta)(\sigma_1)(y) = \delta(\bar{\delta}(\sigma_1))(y) = \delta(\bar{\delta}(\sigma_2))(y) = (\bar{\delta} \circ \delta)(\sigma_2)(y)$  i.e.,  $\bar{\delta} \circ \delta \in \mathbf{y}$ .

Conversely, assume that  $\delta \in \mathbf{S}(\mathbf{y}) \rightarrow \mathbf{y}$  for every  $y \in V$ . We show that if  $\delta$  features a flow  $v \rightsquigarrow w$  then  $v \equiv x_i$  and  $w \equiv y_i$  for some  $1 \leq i \leq n$ .



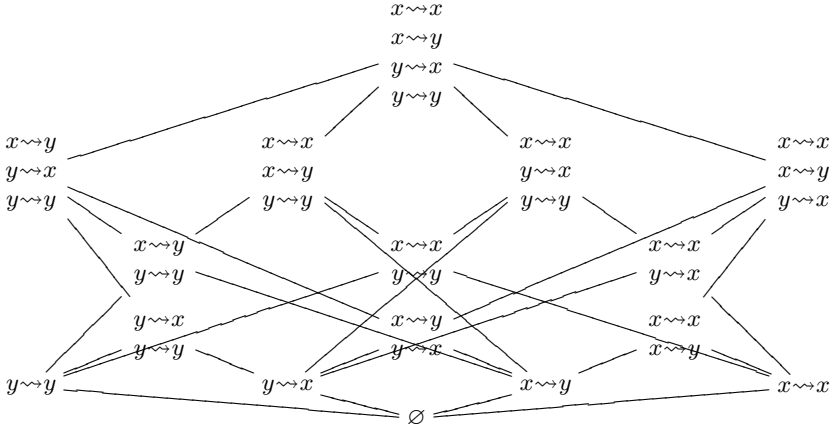


Fig. 1. The abstract domain  $\text{IF}_{\{x,y\}}$

$w \in \{y_1, \dots, y_n\}$ . Let by contradiction  $w \notin \{y_1, \dots, y_n\}$ . There exist  $\sigma_1, \sigma_2 \in \Sigma$  such that  $\sigma_1|_{V \setminus v} = \sigma_2|_{V \setminus v}$  and  $\delta(\sigma_1)(w) \neq \delta(\sigma_2)(w)$ . Since  $S(w) = \emptyset$ , we have  $\delta \in \emptyset \rightarrow \mathbf{w} = \mathbf{w}$ . Then  $\delta(\sigma_1)(w) = \delta(\sigma_2)(w)$ , a contradiction.

$v \in \{x_i \mid x_i \rightsquigarrow w \in f\}$ . Let by contradiction  $v \notin \{x_i \mid x_i \rightsquigarrow w \in f\}$  i.e.,  $v \notin S(w)$ . There exist  $\sigma_1, \sigma_2 \in \Sigma$  such that  $\sigma_1|_{V \setminus v} = \sigma_2|_{V \setminus v}$  and  $\delta(\sigma_1)(w) \neq \delta(\sigma_2)(w)$ . Let  $\bar{\delta}$  be such that  $\bar{\delta}(\sigma) = \sigma_1[v \mapsto \sigma(v)]$ . We have  $\bar{\delta}(\sigma_1) = \sigma_1$ ,  $\bar{\delta}(\sigma_2) = \sigma_2$ . Moreover, we have  $\bar{\delta} \in \mathbf{S}(\mathbf{w})$  since  $v \notin S(w)$ . We conclude that  $\bar{\delta} \circ \delta \in \mathbf{w}$ . But  $(\bar{\delta} \circ \delta)(\sigma_1)(w) = \delta(\bar{\delta}(\sigma_1))(w) = \delta(\sigma_1)(w) \neq \delta(\sigma_2)(w) = \delta(\bar{\delta}(\sigma_2))(w) = (\bar{\delta} \circ \delta)(\sigma_2)(w)$ , which is a contradiction.

*Example 14.* Consider the abstract element  $y \rightsquigarrow y$  over  $V = \{x, y\}$ . We have  $S(x) = \emptyset$  and  $S(y) = \{y\}$ . Then  $x \rightsquigarrow y = (\emptyset \rightarrow \mathbf{x}) \cap (\mathbf{y} \rightarrow \mathbf{y}) = \mathbf{x} \cap (\mathbf{y} \rightarrow \mathbf{y})$ .

We prove now that  $\mathbf{C} \rightarrow \mathbf{C} \subseteq \text{IF}$ . We first show that each single arrow in  $\mathbf{C} \rightarrow \mathbf{C}$  belongs to  $\text{IF}$  (Lemma 3) and then lift this result to arbitrary elements of  $\mathbf{C} \rightarrow \mathbf{C}$  (Proposition 4).

**Lemma 3.** *Let  $x_1, \dots, x_n, y \in V$ . We have*

$$\mathbf{x}_1 \cdots \mathbf{x}_n \rightarrow \mathbf{y} = \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in \{x_1, \dots, x_n\}\}.$$

*Proof.* Let  $\delta \in \mathbf{x}_1 \cdots \mathbf{x}_n \rightarrow \mathbf{y}$ . Assume that  $\delta$  features a flow  $v \rightsquigarrow w$ . If  $w \neq y$  then  $v \rightsquigarrow w \in \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\}$ . Assume then  $w \equiv y$ . We must prove that  $v \in \{x_1, \dots, x_n\}$ . Let by contradiction  $v \notin \{x_1, \dots, x_n\}$ . There are  $\sigma_1, \sigma_2 \in \Sigma$  such that  $\sigma_1|_{V \setminus v} = \sigma_2|_{V \setminus v}$  and  $\delta(\sigma_1)(y) \neq \delta(\sigma_2)(y)$ . Let  $\bar{\delta}(\sigma) = \sigma_1[v \mapsto \sigma(v)]$ . We have  $\bar{\delta}(\sigma_1) = \sigma_1$  and  $\bar{\delta}(\sigma_2) = \sigma_2$ . Moreover, since  $v \notin \{x_1, \dots, x_n\}$ , we have  $\bar{\delta} \in \mathbf{x}_1 \cdots \mathbf{x}_n$ . Then  $\bar{\delta} \circ \delta \in \mathbf{y}$ . But  $(\bar{\delta} \circ \delta)(\sigma_1)(y) = \delta(\bar{\delta}(\sigma_1))(y) = \delta(\sigma_1)(y) \neq \delta(\sigma_2)(y) = \delta(\bar{\delta}(\sigma_2))(y) = (\bar{\delta} \circ \delta)(\sigma_2)(y)$ , which is a contradiction.

Conversely, let  $\delta$  feature flows in  $\{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in \{x_1, \dots, x_n\}\}$  only. Let  $\bar{\delta} \in \mathbf{x}_1 \cdots \mathbf{x}_n$ . We must prove that  $\bar{\delta} \circ \delta \in \mathbf{y}$ . Given  $\sigma_1, \sigma_2 \in$

$\Sigma$ , we have  $\bar{\delta}(\sigma_1)|_{\{x_1, \dots, x_n\}} = \bar{\delta}(\sigma_2)|_{\{x_1, \dots, x_n\}}$  since  $\bar{\delta} \in \mathbf{x}_1 \cdots \mathbf{x}_n$ . Moreover,  $\delta$  does not feature any flow from any  $v \in V \setminus \{x_1, \dots, x_n\}$  to  $y$ . By Lemma 1 we have  $(\bar{\delta} \circ \delta)(\sigma_1)(y) = \delta(\bar{\delta}(\sigma_1))(y) = \delta(\bar{\delta}(\sigma_2))(y) = (\bar{\delta} \circ \delta)(\sigma_2)(y)$ . Since  $\sigma_1$  and  $\sigma_2$  are arbitrary, we conclude that  $\bar{\delta} \circ \delta \in \mathbf{y}$ .

*Example 15.* Consider the abstract element  $\mathbf{x} \cap (\mathbf{y} \rightarrow \mathbf{y})$  and assume  $V = \{x, y\}$ . By Lemma 3 we have  $\mathbf{x} = \emptyset \rightarrow \mathbf{x} = \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus x\} \cup \{v \rightsquigarrow x \mid v \in \emptyset\} = \{x \rightsquigarrow y, y \rightsquigarrow y\} \cup \emptyset = \{x \rightsquigarrow y, y \rightsquigarrow y\}$ . By the same lemma,  $\mathbf{y} \rightarrow \mathbf{y} = \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in \{y\}\} = \{x \rightsquigarrow x, y \rightsquigarrow x\} \cup \{y \rightsquigarrow y\} = \{x \rightsquigarrow x, y \rightsquigarrow x, y \rightsquigarrow y\}$ . The intersection of  $\mathbf{x}$  and  $\mathbf{y} \rightarrow \mathbf{y}$  is then  $\{y \rightsquigarrow y\}$ . Compare this result with Example 14.

**Corollary 1.** *Let  $vs_1, vs_2 \subseteq V$  and  $y \in V$ . We have  $(\mathbf{vs}_1 \rightarrow \mathbf{y}) \cap (\mathbf{vs}_2 \rightarrow \mathbf{y}) = (\mathbf{vs}_1 \cap \mathbf{vs}_2) \rightarrow \mathbf{y}$ .*

*Proof.* By Lemma 3 we have  $(\mathbf{vs}_1 \rightarrow \mathbf{y}) \cap (\mathbf{vs}_2 \rightarrow \mathbf{y}) = (\{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in vs_1\}) \cap (\{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in vs_2\}) = \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup (\{v \rightsquigarrow y \mid v \in vs_1\} \cap \{v \rightsquigarrow y \mid v \in vs_2\}) = \{v \rightsquigarrow w \mid v \in V \text{ and } w \in V \setminus y\} \cup \{v \rightsquigarrow y \mid v \in vs_1 \cap vs_2\} = (\mathbf{vs}_1 \cap \mathbf{vs}_2) \rightarrow \mathbf{y}$ .

We can now prove that the traditional domain for information flow analysis is the linear refinement of the basic domain for constancy.

**Proposition 4.** *We have  $\mathbf{IF} = \mathbf{C} \rightarrow \mathbf{C}$ .*

*Proof.* By Lemma 2 we conclude that  $\mathbf{IF} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ . Conversely every element of  $\mathbf{C} \rightarrow \mathbf{C}$  is the intersection of arrows of the form  $\mathbf{vs} \rightarrow \mathbf{vs}'$ . We can assume that  $vs'$  is a single variable, since  $\mathbf{vs} \rightarrow (\mathbf{vs}_1 \cap \mathbf{vs}_2) = (\mathbf{vs} \rightarrow \mathbf{vs}_1) \cap (\mathbf{vs} \rightarrow \mathbf{vs}_2)$  (Subsection 2.4). By Lemma 3 and since  $\mathbf{IF}$  is closed by intersection (Proposition 2) we conclude that  $\mathbf{C} \rightarrow \mathbf{C} \subseteq \mathbf{IF}$ .

As a consequence, the abstract domain in Figure 1 can be rewritten in terms of elements of  $\mathbf{C} \rightarrow \mathbf{C}$ . The result is in Figure 2. You can pass from Figure 1 to Figure 2 by using Lemma 2 (as in Example 14) and from Figure 2 to Figure 1 by using Lemma 3 (as in Example 15). Note that in Figure 2 there is one variable at most on the left of arrows. This is because, if  $V = \{x, y\}$ , then  $\mathbf{xy} \rightarrow \mathbf{x} = \mathbf{xy} \rightarrow \mathbf{y} = \wp(\Delta)$ , so that these arrows are tautologies (if everything is constant in the input, the output must be constant). This is false for larger  $V$ . For instance, in  $C_{\{x, y, z\}} \rightarrow C_{\{x, y, z\}}$  the arrow  $\mathbf{xy} \rightarrow \mathbf{x}$  is not a tautology.

## 5 IF Is Optimal and Condensing

We have just seen that  $\mathbf{C} \rightarrow \mathbf{C} = \mathbf{IF}$ . We show now that, if we linearly refine  $\mathbf{C} \rightarrow \mathbf{C}$ , we end up with  $\mathbf{C} \rightarrow \mathbf{C}$  itself. Hence  $\mathbf{C} \rightarrow \mathbf{C}$  already contains all possible dependencies between constancy of variables. This entails [9] that  $\mathbf{IF}$  is *optimal* and *condensing* i.e., a compositional, input-independent static analysis over  $\mathbf{IF}$ , such as that implemented in [7], has the same precision as a non-compositional, input-driven analysis. These results were unknown for  $\mathbf{IF}$  up to now.

The following result states that an arrow between an element of  $\mathbf{C} \rightarrow \mathbf{C}$  and an element of  $\mathbf{C}$  is equal to an element of  $\mathbf{C} \rightarrow \mathbf{C}$ .

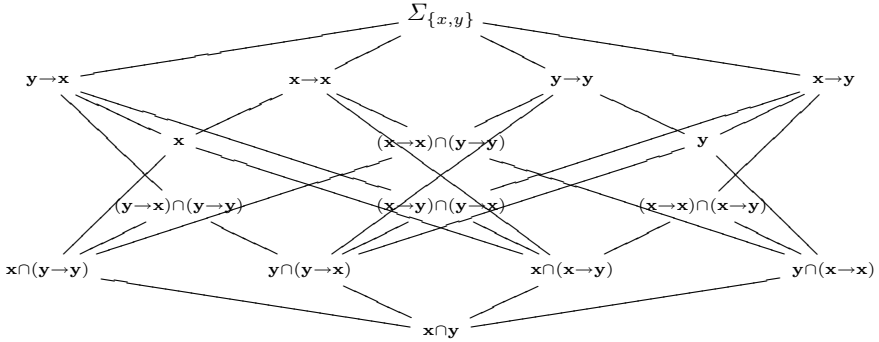


Fig. 2. The abstract domain  $C_{\{x,y\}} \rightarrow C_{\{x,y\}}$

**Lemma 4.** Let  $V = \{v_1, \dots, v_n\}$  and  $vs_i \subseteq V$  for  $1 \leq i \leq n$ . Then

$$((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n)) \rightarrow \mathbf{y} = (\cap\{\mathbf{v}_i \mid vs_i = \emptyset\}) \rightarrow \mathbf{y}.$$

*Proof.* Let  $\delta \in ((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n)) \rightarrow \mathbf{y}$ . Let  $vs = \{v_i \mid vs_i = \emptyset\}$  and  $\bar{\delta} \in \mathbf{vs}$ . We must prove that  $\bar{\delta} \circ \delta \in \mathbf{y}$ . Assume by contradiction that  $\bar{\delta} \circ \delta \notin \mathbf{y}$ . Then there are  $\sigma_1, \sigma_2 \in \Sigma$  such that  $(\bar{\delta} \circ \delta)(\sigma_1)(y) \neq (\bar{\delta} \circ \delta)(\sigma_2)(y)$ .

We can assume without any loss of generality that  $\bar{\delta}(\sigma)(v) = \sigma(v)$  for every  $\sigma \in \Sigma$  and  $v \in V \setminus vs$ , since otherwise we can take  $\bar{\delta}'$  such that  $\bar{\delta}'(\sigma) = \sigma[v \mapsto \bar{\delta}(\sigma)(v) \mid v \in vs]$ ,  $\sigma'_1 = \bar{\delta}(\sigma_1)$ ,  $\sigma'_2 = \bar{\delta}(\sigma_2)$  and still have  $\bar{\delta}' \in \mathbf{vs}$ ,  $(\bar{\delta}' \circ \delta)(\sigma'_1)(y) = \delta(\bar{\delta}'(\bar{\delta}(\sigma_1)))(y) = \delta(\bar{\delta}(\sigma_1))(y) \neq \delta(\bar{\delta}(\sigma_2))(y) = \delta(\bar{\delta}'(\bar{\delta}(\sigma_2)))(y) = (\bar{\delta}' \circ \delta)(\sigma'_2)(y)$ .

Let  $k_1, k_2$  be two distinct concrete values. Define  $\delta'$  such that, for all  $\sigma \in \Sigma$ ,

$$\delta'(\sigma)(v_i) = \begin{cases} \sigma_1(v_i) & \text{if } vs_i \neq \emptyset \text{ and for all } w \in vs_i \text{ we have } \sigma(w) = k_1 \\ \sigma_2(v_i) & \text{if } vs_i \neq \emptyset \text{ and for some } w \in vs_i \text{ we have } \sigma(w) \neq k_1 \\ k_1 & \text{otherwise.} \end{cases}$$

Define the states  $\varsigma_1, \varsigma_2$  such that  $\varsigma_1(w) = k_1$  and  $\varsigma_2(w) = k_2$  for every  $w \in V$ .

By construction, we have  $\delta' \in ((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n))$ . Moreover, we have  $\delta'(\varsigma_1)(v) = \sigma_1(v)$  and  $\delta'(\varsigma_2)(v) = \sigma_2(v)$  for every  $v \in V \setminus vs$ . Since we assume that  $\bar{\delta}$  leaves the variables in  $S \setminus vs$  unaffected, we conclude that  $\delta' \circ \bar{\delta} \in ((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n))$ . Moreover, for  $i = 1, 2$  we have

$$(\delta' \circ \bar{\delta})(\varsigma_i)(v) = (\bar{\delta}(\delta'(\varsigma_i)))(v) = \begin{cases} \bar{\delta}(\sigma_i)(v) & \text{if } v \in vs \\ \delta'(\varsigma_i)(v) = \sigma_i(v) = \bar{\delta}(\sigma_i)(v) & \text{if } v \notin vs. \end{cases}$$

Then  $((\delta' \circ \bar{\delta}) \circ \delta)(\varsigma_1)(y) = (\delta((\delta' \circ \bar{\delta})(\varsigma_1)))(y) = (\delta(\bar{\delta}(\sigma_1)))(y) \neq (\delta(\bar{\delta}(\sigma_2)))(y) = (\delta((\delta' \circ \bar{\delta})(\varsigma_2)))(y) = ((\delta' \circ \bar{\delta}) \circ \delta)(\varsigma_1)(y)$ . But by definition of  $\delta$ , we have  $(\delta' \circ \bar{\delta}) \circ \delta \in \mathbf{y}$ , which is a contradiction.

Conversely, let  $\delta \in (\cap\{\mathbf{v}_i \mid vs_i = \emptyset\}) \rightarrow \mathbf{y}$ . Let  $\bar{\delta} \in ((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n))$ . We must prove that  $\bar{\delta} \circ \delta \in \mathbf{y}$ . For each  $i$  such that  $vs_i = \emptyset$  we have  $\bar{\delta} \in \emptyset \rightarrow \mathbf{v}_i = \mathbf{v}_i$ . We conclude that  $\bar{\delta} \in \cap\{\mathbf{v}_i \mid vs_i = \emptyset\}$  and then  $\bar{\delta} \circ \delta \in \mathbf{y}$ .

**Corollary 2.** *We have  $C \rightarrow C \rightarrow C = C \rightarrow C$ .*

*Proof.* By monotonicity (Subsection 2.4) and Proposition 3 we have  $C \rightarrow C \rightarrow C = C \rightarrow (C \rightarrow C) \supseteq C \rightarrow C$ . Conversely, each element of  $C \rightarrow C \rightarrow C = (C \rightarrow C) \rightarrow C$  is the intersection of arrows  $a_1 \rightarrow \mathbf{vs}$  with  $a_1 \in C \rightarrow C$  and  $\mathbf{vs} \in C$ . We can assume that  $\mathbf{vs} = \mathbf{y}$  with  $y \in V$  since  $a_1 \rightarrow (\mathbf{vs}_1 \cap \mathbf{vs}_2) = (a_1 \rightarrow \mathbf{vs}_1) \cap (a_1 \rightarrow \mathbf{vs}_2)$ . The set  $a_1$  is the intersection of arrows  $\mathbf{vs}_1 \rightarrow \mathbf{vs}'_1 \cap \dots \cap \mathbf{vs}_n \rightarrow \mathbf{vs}'_n$  with  $vs_i, vs'_i \subseteq V$  for  $1 \leq i \leq n$ . We can assume that each  $vs'_i$  is a singleton variable  $v_i$  for the same reason used above for  $\mathbf{vs}$ . Moreover, we can assume that  $v_1, \dots, v_n$  are all distinct (and hence  $n$  is finite) since if otherwise  $v_i \equiv v_j$  with  $i \neq j$ , then by Corollary 1 we can substitute  $(\mathbf{vs}_i \rightarrow \mathbf{v}_i) \cap (\mathbf{vs}_j \rightarrow \mathbf{v}_j)$  with  $(\mathbf{vs}_i \cap \mathbf{vs}_j) \rightarrow \mathbf{v}_i$ . Moreover, we can assume that  $\{v_1, \dots, v_n\} = V$  since if there is  $v \in V \setminus \{v_1, \dots, v_n\}$  then we can add the tautological arrow  $\cap\{\mathbf{w} \mid w \in V\} \rightarrow \mathbf{v} = \Delta$ . In conclusion, every element  $e$  of  $(C \rightarrow C) \rightarrow C$  is the intersection of arrows of the form  $((\mathbf{vs}_1 \rightarrow \mathbf{v}_1) \cap \dots \cap (\mathbf{vs}_n \rightarrow \mathbf{v}_n)) \rightarrow \mathbf{y}$  with  $v \in V$ ,  $vs_i \subseteq V$  and  $v_i \in V$  for each  $1 \leq i \leq n$ . By Lemma 4,  $e$  is equal to the intersection of arrows  $(\cap\{\mathbf{v}_i \mid vs_i = \emptyset\}) \rightarrow \mathbf{y}$  i.e., to the intersection of elements of  $C \rightarrow C$ . Since  $C \rightarrow C$  is closed by intersection, we have the thesis.

It is easy now to prove that  $C \rightarrow C$  is closed *w.r.t.* linear refinement, and is hence optimal and condensing [9].

**Proposition 5.** *We have  $(C \rightarrow C) \rightarrow (C \rightarrow C) = C \rightarrow C$ .*

*Proof.* We have  $(C \rightarrow C) \rightarrow (C \rightarrow C) = ((C \rightarrow C) \sqcap C) \rightarrow C$  (Subsection 2.4) and since  $C \subseteq C \rightarrow C$  (Proposition 3) we conclude that  $(C \rightarrow C) \sqcap C = C \rightarrow C$  [5] and hence  $(C \rightarrow C) \rightarrow (C \rightarrow C) = (C \rightarrow C) \rightarrow C$ . The thesis follows by Corollary 2.

## 6 A Logical Representation for IF

We have seen in Section 4 that IF coincides with  $C \rightarrow C$ . We show here that Boolean formulas can be used to represent elements of  $C \rightarrow C$ .

Since the elements of  $C \rightarrow C$  express dependencies between the constancy of variables in the input and the constancy of variables in the output, we need to distinguish such variables. Hence we write  $\check{v}$  for the variable  $v$  in the input of a denotation, and  $\hat{v}$  for the same variable in the output of a denotation [6].

**Definition 7 (Denotational Formulas).** *The denotational formulas over  $V$  are the Boolean (propositional) formulas over the variables  $\{\check{v} \mid v \in V\} \cup \{\hat{v} \mid v \in V\}$ , modulo logical equivalence.*

**Definition 8.** *Let  $vs \subseteq V$ . We define  $\check{vs} = \{\check{v} \mid v \in vs\}$  and  $\hat{vs} = \{\hat{v} \mid v \in vs\}$ . Let  $vs \subseteq \{\check{v} \mid v \in V\} \cup \{\hat{v} \mid v \in V\}$ . We define  $\wedge vs = \wedge\{v \mid v \in vs\}$ .*

We specify now the meaning or *concretisation* of a denotational formula  $\phi$ . It is the set of denotations whose behaviour *w.r.t.* constancy is consistent with the propositional models of  $\phi$ .

**Definition 9.** The concretisation of a denotational formula  $\phi$  is

$$\gamma(\phi) = \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \phi\}.$$

**Lemma 5.** Let  $\phi_1, \phi_2$  be denotational formulas. Then  $\gamma(\phi_1 \wedge \phi_2) = \gamma(\phi_1) \cap \gamma(\phi_2)$ .

*Proof.*

$$\begin{aligned} \gamma(\phi_1 \wedge \phi_2) &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models (\phi_1 \wedge \phi_2)\} \\ &= \left\{ \delta \in \Delta \mid \begin{array}{l} \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \phi_1 \\ \text{and } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \phi_2 \end{array} \right\} \\ &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \phi_1\} \\ &\quad \cap \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \phi_2\} \\ &= \gamma(\phi_1) \cap \gamma(\phi_2). \end{aligned}$$

**Lemma 6.** Let  $x \in V$ . We have  $\gamma(\hat{x}) = \mathbf{x}$ .

*Proof.*

$$\begin{aligned} \gamma(\hat{x}) &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \hat{x}\} \\ &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } x \in \text{const}(\bar{\delta} \circ \delta)\} \\ &= \{\delta \in \Delta \mid x \in \text{const}(\delta)\} = \mathbf{x}, \end{aligned}$$

since if  $x \in \text{const}(\bar{\delta} \circ \delta)$  then  $x \in \text{const}(\delta)$  since we can choose  $\bar{\delta} = \iota$ , the identity denotation. Conversely, if  $x \in \text{const}(\delta)$  then  $x \in \text{const}(\bar{\delta} \circ \delta)$ .

**Lemma 7.** Let  $\{x_1, \dots, x_n\} \subseteq V$ ,  $y \in V$ .  $\check{X} = \bigwedge_{1 \leq i \leq n} \check{x}_i$  and  $\hat{X} = \bigwedge_{1 \leq i \leq n} \hat{x}_i$ . Then  $\gamma(\check{X} \Rightarrow \hat{y}) = \gamma(\hat{X}) \rightarrow \gamma(\hat{y})$ .

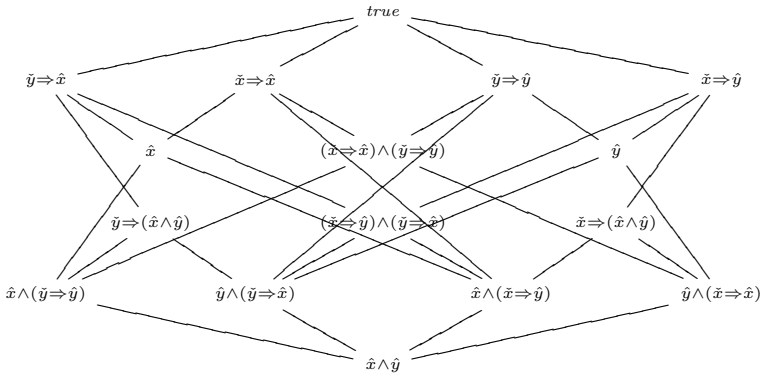
*Proof.*

$$\begin{aligned} \gamma(\check{X} \Rightarrow \hat{y}) &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \Delta \text{ we have } \check{\text{const}}(\bar{\delta}) \cup \hat{\text{const}}(\bar{\delta} \circ \delta) \models \check{X} \Rightarrow \hat{y}\} \\ &= \left\{ \delta \in \Delta \mid \begin{array}{l} \text{for all } \bar{\delta} \in \Delta \\ (x_i \in \text{const}(\bar{\delta}) \text{ for all } i \text{ such that } 1 \leq i \leq n) \text{ entails} \\ y \in \text{const}(\bar{\delta} \circ \delta) \end{array} \right\} \\ &= \left\{ \delta \in \Delta \mid \begin{array}{l} \text{for all } \bar{\delta} \in \Delta \\ (\bar{\delta} \in \gamma(\hat{x}_i) \text{ for all } i \text{ such that } 1 \leq i \leq n) \text{ entails} \\ \bar{\delta} \circ \delta \in \gamma(\hat{y}) \end{array} \right\} \\ &= \{\delta \in \Delta \mid \text{for all } \bar{\delta} \in \gamma(\hat{X}) \text{ we have } \bar{\delta} \circ \delta \in \gamma(\hat{y})\} = \gamma(\hat{X}) \rightarrow \gamma(\hat{y}). \end{aligned}$$

**Proposition 6.** The domain  $\text{IF} = \mathbf{C} \rightarrow \mathbf{C}$  is isomorphic to the set of denotational formulas of the form  $\bigwedge \check{v}s \Rightarrow \bigwedge \hat{w}s$  with  $v_s, w_s \subseteq V$ .

*Proof.* By Lemmas 5, 6 and 7, since  $\mathbf{v}s \rightarrow (\mathbf{w}_1 \cap \dots \cap \mathbf{w}_m) = (\mathbf{v}s \rightarrow \mathbf{w}_1) \cap \dots \cap (\mathbf{v}s \rightarrow \mathbf{w}_m)$  and  $\bigwedge \check{v}s \Rightarrow (w_1 \wedge \dots \wedge w_m) = (\bigwedge \check{v}s \Rightarrow w_1) \wedge \dots \wedge (\bigwedge \check{v}s \Rightarrow w_m)$ .

Figure 3 shows the Boolean representation of  $\text{IF}_{\{x,y\}} = \mathbf{C}_{\{x,y\}} \rightarrow \mathbf{C}_{\{x,y\}}$ .



**Fig. 3.** The representation of  $IF_{\{x,y\}} = C_{\{x,y\}} \rightarrow C_{\{x,y\}}$  through denotational formulas

## 7 Conclusion

We have used linear refinement to reconstruct an existing domain for information flow analysis, to prove it optimal and condensing, and to provide an efficient representation in terms of Boolean formulas. The size of the abstract domain  $IF_V = C_V \rightarrow C_V$  grows exponentially with  $V$ , but  $V$  only contains the variables in scope in the program point under analysis. Its actual application to the analysis of relatively large programs has been experimentally validated in [7].

Our work has similarities with the reconstruction through linear refinement of abstract domains for groundness analysis of logic programs [13]. In particular, constancy is the imperative counterpart of groundness in logic programming. There, however, two iterations of linear refinement (only one here) are needed to reach an abstract domain which is closed *w.r.t.* further refinements. There might also be relations with strictness analysis of functional programs, which has also been proved to enjoy some optimality property [14]. There, optimality means that precision cannot be improved as long as constant symbols are abstracted away. It is enlightening to observe that the same abstraction is used in groundness analysis of logic programs, where all functor symbols are abstracted away. In information flow analysis, values are abstracted away, and only their constancy is observed. These similarities might not be casual.

We are confident that our work can be generalised to *declassified* forms of non-interference, such as *abstract non-interference* [8]. One should consider a declassified form of constancy as the basic domain to refine. Declassified constancy means that a variable, in the output of a denotation, is *always* bound to a given *abstract* value, as specified by the declassification criterion.

## References

1. A. Bossi, M. Gabbrilli, G. Levi, and M. Martelli. The s-Semantics Approach: Theory and Applications. *Journal of Logic Programming*, 19/20:149–197, 1994.
2. R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

3. D. Clark, C. Hankin, and S. Hunt. Information Flow for Algol-like Languages. *Computer Languages and Security*, 28(1):3–28, April 2002.
4. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of the 4th ACM Symposium on Principles of Programming Languages (POPL)*, pages 238–252, 1977.
5. P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. of the 6th ACM Symp. on Principles of Programming Languages*, pages 269–282, 1979.
6. S. Genaim, R. Giacobazzi, and I. Mastroeni. Modeling Secure Information Flow with Boolean Functions. In P. Ryan, editor, *ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security*, pages 55–66, April 2004.
7. S. Genaim and F. Spoto. Information Flow Analysis for Java Bytecode. In R. Cousot, editor, *Proc. of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 346–362, Paris, France, January 2005.
8. R. Giacobazzi and I. Mastroeni. Abstract Non-Interference: Parameterizing Non-Interference by Abstract Interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*, pages 186–197, Venice, Italy, January 2004. ACM-Press.
9. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making Abstract Domains Condensing. *ACM Transactions on Computational Logic (ACM-TOCL)*, 6(1):33–60, 2005.
10. R. Giacobazzi and F. Scozzari. A Logical Model for Relational Abstract Domains. *ACM Transactions on Programming Languages and Systems*, 20(5):1067–1109, 1998.
11. A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
12. A. Sabelfeld and D. Sands. A PER Model of Secure Information Flow in Sequential Programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
13. S. Scozzari. Logical Optimality of Groundness Analysis. *Theoretical Computer Science*, 277(1-2):149–184, 2002.
14. M. C. Sekar, P. Mishra, and I. V. Ramakrishnan. On the Power and Limitation of Strictness Analysis Based on Abstract Interpretation. In *Proc. of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'91)*, pages 37–48, Orlando, Florida, January 1991.
15. D. Volpano, G. Smith, and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.
16. G. Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.