J. Functional Programming 2 (4): 407–435, October 1992 © 1992 Cambridge University Press

Abstract interpretation by dynamic partitioning

FRANÇOIS BOURDONCLE

DIGITAL Paris Research Laboratory, 85, avenue Victor Hugo, 92500 Rueil-Malmaison, France
Centre de Mathématiques Appliquées, Ecole des Mines de Paris, Sophia-Antipolis, 06560 Valbonne, France (e-mail: bourdoncle@pr1.dec.com)

Abstract

The essential part of abstract interpretation is to build a machine-representable abstract domain expressing interesting properties about the possible states reached by a program at run-time. Many techniques have been developed which assume that one knows in advance the class of properties that are of interest. There are cases however when there are no *a priori* indications about the 'best' abstract properties to use. We introduce a new framework that enables non-unique representations of abstract program properties to be used, and expose a method, called dynamic partitioning, that allows the dynamic determination of interesting abstract domains using data structures built over simpler domains. Finally, we show how dynamic partitioning can be used to compute non-trivial approximations of functions over infinite domains and give an application to the computation of minimal function graphs.

Capsule review

Dynamic partitioning is a technique which may be used during abstract interpretation to dynamically select the set of abstract properties that are used in a particular analysis. For many problems of abstract interpretation there may be no *a priori* knowledge about the 'best' abstract properties to use. The techniques introduced in this paper describe methods of, given some initial unsatisfactory abstract domain, building a new abstract domain that is better able to capture the properties of interest.

The work reported in the paper is an important contribution to the field of abstract interpretation in that it generalizes the traditional approach to a situation where there is no obvious complete lattice of abstract properties. The work is also important to functional programming in that it gives an alternative treatment of minimal function graphs. Minimal function graphs have been used as a basis for program analysis and also as an efficient way of computing partial fixed points during analysis.

1 Introduction

Abstract interpretation, as defined by Cousot and Cousot (1977), Cousot (1978) and Cousot (1981), provides a general framework aimed at computing *invariance properties* of programs. These properties describe the run-time states that can be reached from a set of initial states P_0 by means of a transition function τ over the subsets of the set of run-time states S, which defines the operational semantics of a given program. When τ is continuous over the lattice $(\mathbf{P}(S), \subseteq)$, the invariant $\mathrm{I} =$

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

$\bigcup_{n \in N} \tau^n(P_0)$ is also the least fixed point $\mathrm{lfp}(\Phi)$ of the function $\Phi = \lambda P.(P_0 \cup \tau(P))$. In most cases, however, S is infinite, and methods must be developed to determine a safe and finitely represented approximation of this least fixed point. Cousot and Cousot (1977) introduced the notion of Galois connection (α, γ) as a general tool for building such approximate frameworks. The *abstraction* function α maps a set of states P to an element $P^{\#}$ of a finitely represented abstract (approximate) lattice $(\mathbf{P}^{\#}(S), \sqsubseteq)$, whereas the *concretization* function γ maps an abstract state $P^{\#}$ to a set of states, called its *meaning*. Then, by defining a safe approximation $\Phi^{\#}$ of Φ, i.e. a function such that $\Phi^{\#} \sqsupseteq \alpha \circ \Phi \circ \gamma$, one can determine an approximate invariant $\mathrm{I}^{\#} = \mathrm{lfp}(\Phi^{\#})$ which is a safe approximation of I, i.e. $\gamma(\mathrm{I}^{\#}) \supseteq \mathrm{I}$.

However, when the approximate lattice is of infinite height, the iterative computation of the approximate invariant may not finitely converge, and speedup techniques, such as widening and narrowing, must be used to determine a safe approximation of $\mathrm{I}^{\#}$. But in many cases, there is not even a clear indication about how to build a *good* and *finitely represented* abstract lattice. This happens when there is no indication about what the least fixed point will 'look like', and therefore no advance knowledge of the properties that should be expressed in the abstract lattice $\mathbf{P}^{\#}(S)$ to precisely describe the invariant I. Differently stated, there is a gap between the exact lattice $\mathbf{P}(S)$ and the abstract lattices that one can *a priori* build or think about. This is true in particular when functions over infinite domains are approximated. Moreover, most implementations of abstract interpretation have to deal with the problem of testing the equivalence of the data structures used to represent lattice elements (i.e. testing the equality of their meaning). This test is often very costly and difficult to implement, as with the abstract interpretation of functional or logic programs for instance, and it would be desirable to design a framework that avoids such a test.

The aim of this paper is to discuss a technique, which we call *dynamic partitioning*, that can be used to compute non-trivial, safe approximations of program invariants in the above cases, by dynamically selecting interesting and finitely represented abstract properties without having to test the equality of their meaning.

We shall first recall, in Section 2, the classical definition of a widening operator, and then describe, in Section 3, a framework that generalizes the classical lattice-oriented framework to cases where no Galois connection can be defined and where testing the equivalence of data structures can be avoided by using properly generalized widening operators over general partial orders. We shall then discuss in Section 4 several classical situations in interprocedural abstract interpretation and show how our technique can be applied in each case. We will show in particular how non-trivial approximations of functional fixed points can be computed by using our framework and an adequate data structure. Finally, we shall give in Section 5 two practical applications and show how dynamic partitioning can be used to effectively compute non-trivial, safe approximations of minimal function graphs.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

2 Widening operators

The Galois connection approach described above is perfectly adequate when the abstract lattice $\mathbf{P}^{\#}(S)$ is of *finite height*, since the iterative computation of the least fixed point of $\Phi^{\#}$ over $\mathbf{P}^{\#}(S)$ will necessarily converge in a finite amount of time. However, some interesting program properties, such as the range of integer variables, are expressed in a lattice of infinite height. Even if the integer variables were bounded, choosing for instance $\mathbf{Z} = \{\omega^-, \ldots, \omega^+\}$, $\omega^- = -2^{w-1}$, $\omega^+ = 2^{w-1}-1$, the interval lattice $\mathbf{I}(\mathbf{Z})$ is still of height 2^w, and fixed point computations may in theory require up to the same number of iterations. A speedup technique has been proposed by Cousot and Cousot (1977) that uses so called *widening operators* to transform infinite iterative computations into finite but approximated ones. So let us suppose for instance that one has a program function ***Loop*** defined in ML as follows:

```
fun Loop i = if i < 100 then
                  Loop (i+1)
             else
                  i
```

Suppose now that one wants to compute the values returned by ***Loop*** for a set of *input data specifications*. ***Loop*** being recursive, this computation may require computing the value returned by ***Loop*** for arguments that were not present in the initial data set. Therefore, the goal of an interprocedural abstract interpretation framework will be to determine this *minimal function graph* describing the minimal information about ***Loop*** needed to compute its value for every argument in the original specification. This notion of minimal function graph was first introduced in Jones and Mycroft (1986), but was in essence already present by Cousot and Cousot (1978). A program state $\langle a, b\rangle \in \mathbf{Z} \times \mathbf{Z}_\perp$ therefore consists of an input value a and a return value $b = \boldsymbol{Loop}(a)$, where the special value $\perp$ denotes nontermination. Generalizing the idea used by Jones and Mycroft for constant propagation, we can approximate the minimal function graph of ***Loop*** by a pair of intervals representing an approximation of all of ***Loop***'s arguments and all of ***Loop***'s results. This approximate minimal function graph is therefore the least fixed point $X^{\#}$ of the monotonic function $\Phi^{\#}$ over the lattice $\mathbf{I}(\mathbf{Z}) \times \mathbf{I}(\mathbf{Z})$ defined as follows:

$$\Phi^{\#}\langle i, v\rangle = \langle i_0, \perp\rangle \vee \langle \Phi_1^{\#}(i), \Phi_2^{\#}(i, v)\rangle$$

where i_0 is the input data specification, and the two functions $\Phi_1^{\#}$ and $\Phi_2^{\#}$ are defined by:

$$\Phi_1^{\#}(i) = \mathbf{incr}^{\#}(i \wedge [\omega^-, 99])$$
$$\Phi_2^{\#}(i, v) = v \vee (i \wedge [100, \omega^+])$$

where the strict abstract function $\mathbf{incr}^{\#}$ is defined by:

$$\mathbf{incr}^{\#}(\perp) = \perp$$
$$\mathbf{incr}^{\#}[a, b] = [\mathbf{min}(a+1, \omega^+), \mathbf{min}(b+1, \omega^+)].$$

The least fixed point $X^\#$ is known to be the upper limit of the increasing chain:

$$X_0^\# = \langle \bot, \bot \rangle$$
$$X_{n+1}^\# = \Phi^\#(X_n^\#).$$

But this limit can take a very long time to compute. For instance, if $i_0 = [0,0]$, this least fixed point is equal to $\langle [0,100],[100,100] \rangle$ and is reached after 102 iterations. One must therefore find a way to *speedup* this computation in order for it to be tractable. To this end, we introduce the *widening operator* ∇_{I} over $\mathbf{I}(\mathbf{Z})$, taken from Cousot and Cousot (1977, p. 247), or Cousot (1981, p. 334):

$$\bot \nabla_{\mathrm{I}} x = x \nabla_{\mathrm{I}} \bot = x$$
$$\begin{aligned}[a_1, b_1] \nabla_{\mathrm{I}} [a_2, b_2] = [&\textbf{if } a_2 < a_1 \textbf{ then } \omega^- \textbf{ else } a_1,\\ &\textbf{if } b_2 > b_1 \textbf{ then } \omega^+ \textbf{ else } b_1].\end{aligned}$$

This non-commutative operator generalizes 'unstable' bounds of its right argument. It is a safe approximation of the join operator, and is such that for every increasing chain $(x_n)_{n \in \mathrm{N}}$, the increasing chain $(y_n)_{n \in \mathrm{N}}$ defined by:

$$\begin{cases} y_0 = x_0 \\ y_{n+1} = y_n \nabla_{\mathrm{I}} x_{n+1} \end{cases}$$

is always eventually stable, i.e. there exists a n_0 such that: $\forall n \geqslant n_0 : y_n = y_{n_0}$. Under these assumptions, it is well known (Cousot, 1981, theorem 10–30, p. 334) that the upper limit $Y^\#$ of the increasing chain:

$$\begin{cases} Y_0^\# = \langle \bot, \bot \rangle \\ Y_{n+1}^\# = Y_n^\# \nabla_{\mathrm{I}} \Phi^\#(Y_n^\#) & \textbf{if } \neg (\Phi^\#(Y_n^\#) \leqslant Y_n^\#) \\ Y_{n+1}^\# = Y_n^\# & \textbf{if } \Phi^\#(Y_n^\#) \leqslant Y_n^\# \end{cases}$$

where the widening operator is applied componentwise, is a post fixed point of $\Phi^\#$, i.e. $\Phi^\#(Y^\#) \leqslant Y^\#$, and, therefore, is a safe approximation of $X^\#$, i.e. $X^\# \leqslant Y^\#$. Note that since here:

$$y \leqslant x \Rightarrow x \nabla_{\mathrm{I}} y = x$$

this chain could be simply defined by:

$$\begin{cases} Y_0^\# = \langle \bot, \bot \rangle \\ Y_{n+1}^\# = Y_n^\# \nabla_{\mathrm{I}} \Phi^\#(Y_n^\#). \end{cases}$$

So, for example, with input data specification $i_0 = [0,0]$, one can compute the increasing chain:

$$Y_0^\# = \langle \bot, \bot \rangle$$
$$Y_1^\# = \langle [0,0], \bot \rangle$$
$$Y_2^\# = \langle [0,\omega^+], \bot \rangle$$
$$Y_3^\# = Y^\# = \langle [0,\omega^+],[100,\omega^+] \rangle$$

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

whose limit $Y^\#$ is a safe approximation of the least fixed point:

$$X^\# = \langle [0, 100], [100, 100] \rangle.$$

This result could also be improved by using the *narrowing operator* Δ_I defined by:

$$\begin{aligned} \bot \, \Delta_I \, x = x \, \Delta_I \, \bot &= \bot \\ [a_1, b_1] \, \Delta_I \, [a_2, b_2] &= [\textbf{if } a_1 = \omega^- \textbf{ then } a_2 \textbf{ else } \textbf{min}(a_1, a_2), \\ &\qquad \textbf{if } b_1 = \omega^+ \textbf{ then } b_2 \textbf{ else } \textbf{max}(b_1, b_2)]. \end{aligned}$$

This operator satisfies the canonical condition:

$$\forall x, y \in \mathbf{I}(\mathbf{Z}) : x \geqslant y \Rightarrow x \geqslant x \, \Delta_I \, y \geqslant y$$

and is such that for every decreasing chain $(y_n)_{n \in \mathrm{N}}$, the chain $(z_n)_{n \in \mathrm{N}}$ defined by:

$$\begin{cases} z_0 = y_0 \\ z_{n+1} = z_n \, \Delta_I \, y_{n+1} \end{cases}$$

is always eventually stable. It is known that the lower limit $Z^\#$ of the decreasing chain:

$$\begin{cases} Z_0^\# = Y^\# \\ Z_{n+1}^\# = Z_n^\# \, \Delta_I \, \Phi^\#(Z_n^\#) \end{cases}$$

starting from the post fixed point $Y^\#$, is a safe approximation of $X^\#$. On our example, this gives, after only two iterations:

$$\begin{aligned} Z_0^\# &= \langle [0, \omega^+], [100, \omega^+] \rangle \\ Z_1^\# = Z^\# &= \langle [0, 100], [100, \omega^+] \rangle. \end{aligned}$$

This example show how good results can be obtained by using very naïve and 'brute force' widening and narrowing operators. Of course, it might be argued that the interval [0, 100] inferred by the computation could have been easily determined by simply looking at the *text* of the program, and that a finite abstract lattice could thus have been built *a priori*. We shall see, in Section 5.2, an example that shows that this is not always the case, and in practice, building *ad hoc* approximate functional lattices is simply not feasible. However, since we already know how to deal with intervals, it is very tempting to describe minimal function graphs by *sets* of interval pairs, instead of a single pair of intervals. The advantage of such a representation is that it is very flexible, and does not establish in advance any particular tradeoff between complexity and precision.

This is however a difficult problem to solve if we want to use this approach, in that there is no canonical representation of abstract minimal function graphs. For example, it is quite clear that the two sets $\{\langle [1, 2], [0, 0] \rangle\}$ and $\{\langle [1, 1], [0, 0] \rangle, \langle [2, 2], [0, 0] \rangle\}$ are equivalent in the sense that they represent the same minimal function graph, but no one is 'better' than the other. What we would like to do, however, is to work with such representations, even though they are not unique, and still ensure the convergence and safeness of every computation. The traditional complete lattice framework is clearly not appropriate in this case, so we need to generalize the approach to general partial orders.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

3 Representations

Definition 1

Let $(D, \bot, \sqsubseteq)$ be a countable complete partial order (cpo), R a set, and $\gamma: R \to D$ a meaning function. Then $(R, \preccurlyeq, \gamma, \nabla)$ is said to be a representation of D if:

(i) *$(R, \bot, \preccurlyeq)$ is a partial order*

(ii) *The meaning function γ is monotonic*

(iii) *Each element $d \in D$ can be safely abstracted by an element $\alpha(d) \in R$, i.e.:*

$$\gamma(\alpha(d)) \sqsupseteq d$$

(iv) *Each binary operator $\nabla_i: R \times R \to R$ of the sequence $\nabla = (\nabla_i)_{i \in \mathbb{N}}$ is such that:*

$$\forall r, r' \in R: \begin{cases} r \preccurlyeq r \nabla_i r' \\ \gamma(r') \sqsubseteq \gamma(r \nabla_i r') \end{cases}$$

(v) *For every $\{r_i\}_{i \in \mathbb{N}} \subseteq R$, the chain $(r'_i)_{i \in \mathbb{N}}$ defined by:*

$$\begin{cases} r'_0 = r_0 \\ r'_{i+1} = r'_i \nabla_i r_{i+1} \end{cases}$$

has an upper bound.

A representation is said to be complete if $(R, \preccurlyeq)$ is a cpo, finite if every element of R has a finite encoding, and tractable if the chain $(r'_i)_{i \in \mathbb{N}}$ is always eventually stable.

This definition has some similarities with the definition of the upper approximation $(D^\#, \leqslant)$ of a complete lattice $(D, \sqsubseteq)$ using a Galois connection, i.e. a pair of monotonic functions $(\alpha, \gamma), \alpha: D \to D^\#$ and $\gamma: D^\# \to D$ such that:

$$\forall (d, d^\#) \in D \times D^\#: \alpha(d) \leqslant d^\# \Leftrightarrow d \sqsubseteq \gamma(d^\#).$$

The difference between the two definitions is that our framework makes very weak assumptions about R and D, and generalizes the case where R and D are both complete lattices, since we only require that α be safe. As a matter of fact, α is not even needed in the framework, and only the *existence* of a safe approximation for every concrete element is required. This allows in particular different representations to have the same meaning and one can choose arbitrarily between them, hence the term *representation*. Therefore, the traditional inequality $\alpha \circ \gamma \preccurlyeq \mathrm{Id}_R$, which becomes an equality when γ is one-to-one, does not hold in this framework.

Each *elementary widening operator* ∇_i of the *widening operator* $\nabla = (\nabla_i)_{i \in \mathbb{N}}$ is an alternative to a join operator over the abstract lattice $D^\#$ which does not necessarily exist if R is a partial order. The conditions imposed on ∇_i simply ensure that safe and increasing chains over R can be built from increasing chains over D, as we shall see below. When R and D are both complete lattices, the operator ∇ of a tractable representation is a straightforward generalization of the classical widening operator, in the sense that $\gamma(r \nabla_i r') \sqsupseteq \gamma(r) \sqcup \gamma(r')$ and every increasing chain built using ∇ is eventually stable.

Another remark concerning this framework is that condition (v) is trivially satisfied for any complete representation, for every increasing chain has a least upper bound.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Differently stated, the widening operator ∇ is not necessary in a complete representation to prove the existence of a least upper approximation. But even so, it can be very interesting to have such a widening operator to define *finite* and *tractable* frameworks, as we have seen in Section 2. Also, note that the use of widening operators over non-complete lattices was already present in Cousot and Halbwachs (1978), where the lattice of finitely represented conxex hulls is not complete.

Finally, it should be noted that our framework can be very easily generalized to cases where $(R, \preccurlyeq)$ is only a preorder,[1] in which case the meaning function need not be monotonic and the conditions imposed on the elementary widening operators must be:

$$\forall r, r' \in R: \begin{cases} \gamma(r) \sqsubseteq \gamma(r \nabla_i r') \\ \gamma(r') \sqsubseteq \gamma(r \nabla_i r'). \end{cases}$$

Pre-orders have been used for instance in Stransky (1990). However, the problem with such very general frameworks is that not much can be said about them, for they are essentially defined by the properties of the widening operator ∇. Moreover, pre-orders are not very easy to work with, for representations having the same meaning can 'oscillate' during the computation and one must be able to finitely compute the *equivalence* of representations (i.e. $\gamma(r) = \gamma(r')$) to detect the stabilization of increasing chains. As we noted earlier, this is not necessary here since stabilization is detected by the *equality* of representations (i.e. $r = r'$). Hence, our framework is perfectly suited to cases where R is implemented using very *complex data structures* for which the equivalence test is intractable or very costly, since *we require that equivalent representations be comparable only when they are 'similar enough'*. It is interesting to note that a comparable idea, which was only a heuristic at the time, was used in the design of the widening operator of Cousot and Halbwachs (1978) which preserves as much as possible the representations of convex hulls during iterative computations.

So let $(R, \preccurlyeq, \gamma, \nabla)$ be a representation of D, and $\Phi \in D \to D$ be a continuous function, that is, a monotonic function such that for every directed subset $C \subseteq D: \Phi(\bigsqcup C) = \bigsqcup \Phi(C)$. It is well known that the least fixed point of Φ is:

$$\phi = \mathrm{lfp}(\Phi) = \bigsqcup_{i \in \mathrm{N}} \Phi^i(\perp).$$

If the elements of D do not have a finite encoding, it may be impossible to compute this increasing chain. So let us suppose that one can define a safe approximation $\Phi^{\#}$ of Φ operating over the set of (supposedly finitely encoded) representations R, that is, a function $\Phi^{\#}$ such that:

$$\gamma \circ \Phi^{\#} \sqsupseteq \Phi \circ \gamma.$$

One can choose in particular[2] any function $\Phi^{\#}$ such that $\Phi^{\#} \succcurlyeq \alpha \circ \Phi \circ \gamma$, for by monotonicity of γ and property (iii):

$$\gamma \circ \Phi^{\#} \sqsupseteq (\gamma \circ \alpha) \circ \Phi \circ \gamma \sqsupseteq \Phi \circ \gamma$$

but this is not mandatory. The only thing we really need is *any* safe approximation

[1] A pre-order is a reflexive and transitive binary relation.
[2] As in Cousot (1981, p. 331), for instance.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

of Φ. Our problem is now: how can we compute a safe representation of ϕ using $\Phi^{\#}$ which is the only function that we can possibly compute? To this end, let us define the chain $(r_i)_{i \in \mathrm{N}}$ by:

$$\begin{cases} r_0 = \bot \\ r_{i+1} = r_i \nabla_i \Phi^{\#}(r_i). \end{cases}$$

Theorem 2

The chain $(r_i)_{i \in \mathrm{N}}$ is an increasing chain over R that has an upper bound r_ω which is a safe representation of the least fixed point of Φ, that is:

$$\gamma(r_\omega) \sqsupseteq \phi.$$

Proof

Let us first define the increasing chain $(\phi_i)_{i \in \mathrm{N}}$ by $\phi_0 = \bot$ and $\phi_{i+1} = \Phi(\phi_i)$. It is clear that $\gamma(r_0) \sqsupseteq \bot = \phi_0$. Suppose by induction that $\gamma(r_i) \sqsupseteq \phi_i$. Then by definition of the widening operator ∇:

$$\gamma(r_{i+1}) = \gamma(r_i \nabla_i \Phi^{\#}(r_i)) \sqsupseteq \gamma(\Phi^{\#}(r_i)).$$

But $\Phi^{\#}$ being a safe approximation of Φ, and by monotonicity of Φ:

$$\gamma(\Phi^{\#}(r_i)) \sqsupseteq \Phi(\gamma(r_i)) \sqsupseteq \Phi(\phi_i) = \phi_{i+1}$$

which proves that each r_i is a safe approximation of ϕ_i. But thanks to the first property of ∇, $r_{i+1} \succcurlyeq r_i$, and $(r_i)_{i \in \mathrm{N}}$ is an increasing chain which has an upper bound r_ω. Finally, by monotonicity of γ: $\forall i : \phi_i \sqsubseteq \gamma(r_i) \sqsubseteq \gamma(r_\omega)$, which implies that $\phi \sqsubseteq \gamma(r_\omega)$. □

Using a finite and tractable representation, one can therefore compute a non-trivial, safe and finitely represented approximation of the least fixed point ϕ of any continuous function Φ over D, even if the representation R does not have a maximum element (which would be of course a trivial safe approximation of any element of D).

However, as in Section 2, it is possible to compute an even better representation r'_ω of the least fixed point of Φ by defining a *narrowing operator* $\Delta = (\Delta_i)_{i \in \mathrm{N}}$ such that every elementary narrowing operator Δ_i satisfies:

$$\forall r, r' \in R, \quad \forall \phi \in D : \begin{cases} \phi \sqsubseteq \gamma(r) \\ \phi \sqsubseteq \gamma(r') \end{cases} \Rightarrow \begin{cases} r \Delta_i r' \preccurlyeq r \\ \phi \sqsubseteq \gamma(r \Delta_i r') \end{cases}$$

and computing the lower limit of the decreasing chain $(r'_i)_{i \in \mathrm{N}}$ defined as follows:

$$\begin{cases} r'_0 = r_\omega \\ r'_{i+1} = r'_i \Delta_i \Phi^{\#}(r'_i). \end{cases}$$

Note that, once again, the first condition imposed on Δ_i enforces the chain condition whereas the second condition enforces the safeness of the computation, that is, this condition ensures that the narrowing operator will not 'jump below' the least fixed point, as shown by the following theorem.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Theorem 3
When the decreasing chain $(r'_i)_{i \in \mathbf{N}}$ is eventually stable, its lower limit is a safe representation of the least fixed point of Φ.

Proof
We first note that $r'_0 = r_\omega$ being a safe representation of ϕ, we have:

$$\gamma(r'_0) \sqsupseteq \phi.$$

Now, suppose by induction that:

$$\gamma(r'_i) \sqsupseteq \phi$$

then obviously, by monotonicity of Φ:

$$\gamma(\Phi^{\#}(r'_i)) \sqsupseteq \Phi(\gamma(r'_i)) \sqsupseteq \Phi(\phi) = \phi$$

and therefore:
$$\gamma(r'_{i+1}) = \gamma(r'_i \, \Delta_i \, \Phi^{\#}(r'_i)) \sqsupseteq \phi$$

which shows that the lower limit $r'_\omega = r'_{i_0}$ for some $i_0 \in \mathbf{N}$ is a safe representation of ϕ. □

Note that, contrary to the classical widening/narrowing approach, we do not require that the meaning of the first element $r'_0 = r_\omega$ be a post fixed point of Φ, which, consequently, avoids comparing $\gamma(r_i)$ with $\Phi(\gamma(r_i))$ at each stage of the iterative computation of r_ω, as in Section 2. This property is very important if, as stated in the introduction, comparing the meaning of representations is very costly. However, if the elementary widening operator ∇_i satisfies the natural *stability condition*:

$$\forall r, r' \in R : \gamma(r') \sqsubseteq \gamma(r) \Rightarrow r \nabla_i r' = r$$

as does the widening operator ∇_{I} over the interval lattice $\mathbf{I}(\mathbf{Z})$, then $\gamma(r_\omega)$ will always be a post fixed point of Φ. Elementary widening operators satisfying this condition will be called *stable*. Note that complex widening operators, such as the ones that will be presented in the following sections, will not generally be stable. Intuitively, since we require that two representations r and r' be comparable only when they are 'similar enough', the stability test $\gamma(r') \sqsubseteq \gamma(r)$ will be approximated, and, therefore, redundant information r' added to a representation r will sometimes lead to a loss of precision.

Finally, note that widening and narrowing are not dual operations. However, for the sake of simplicity, we shall only focus on widening operators in the rest of this paper.

4 Dynamic partitioning

The aim of this section is to build generic representations based on the idea of 'non-redundancy'. We shall first discuss what we call *basic partitioning*, which is a technique that can be used to build representations of a concrete complete lattice $(L, \bot, \top, \sqsubseteq, \sqcup, \sqcap)$ using well chosen subsets of A when A is an upper approximation of L. We shall then discuss two other methods, called *basic functional partitioning* and *functional partitioning* used to build representations of functions $F: C \rightarrow B$ over an

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

infinite set C, using subsets of $A \times B$ when A is an upper approximation of $\mathbf{P}(C)$ and B is a lattice. We start by defining two notions of non-redundancy for the subsets of a lattice A.

Definition 4

A subset P of lattice A is said to be non-redundant if it does not contain $\perp$ and:

$$\forall a, a' \in P : a \leqslant a' \Rightarrow a = a'.$$

P is said to be strongly non-redundant if it does not contain $\perp$ and:

$$\forall a, a' \in P : a \neq a' \Rightarrow a \wedge a' = \perp.$$

Non-redundant subsets are often called *crowns* or *antichains*. The set of non-redundant and strongly non-redundant subsets of A will be noted respectively $\mathbf{P}_{\mathrm{nr}}(A)$ and $\mathbf{P}_{\mathrm{snr}}(A)$. ľwo elements of a non-redundant subset are either equal or not comparable, whereas they are equal or have 'nothing in common' if they belong to the same strongly non-redundant subset. Note that strong non-redundancy implies non-redundancy.

4.1 Basic partitioning

So let us suppose that A is an upper approximation of a complete lattice L and let (α, γ) denote the Galois connection between the two lattices. We wish to build a representation of L using the subsets of A. The most natural *meaning* of a subset P of A is of course:

$$\Gamma(P) = \bigsqcup_{a \in P} \gamma(a)$$

that is, the least upper bound of the set of concrete elements denoted by the abstract elements of P. Let us define the binary relation $\preccurlyeq$ over $\mathbf{P}(A)$ by:

$$P \preccurlyeq P' \Leftrightarrow \forall a \in P, \exists a' \in P' : a \leqslant a'.$$

This relation is similar to the preorder used to build the *lower powerdomain* (see Gunter and Scott, 1990, p. 653), sometimes called the *Hoare powerdomain*, or the *relational powerdomain* (see Schmidt, 1953, p. 295). The originality of our framework is that using well chosen subsets of $\mathbf{P}(A)$, we can turn this preorder into a partial order and avoid using principal ideals and complex power domains, as in Mycroft and Nielson (1983), for instance.

Theorem 5

$(\mathbf{P}_{\mathrm{nr}}(A), \preccurlyeq)$ and $(\mathbf{P}_{\mathrm{snr}}(A), \preccurlyeq)$ are partial orders and Γ is monotonic over $\mathbf{P}(A)$.

Proof

$\preccurlyeq$ is obviously a preorder. So let $P \preccurlyeq P'$, $P' \preccurlyeq P$, and $a \in P$. Then there exists $a' \in P'$ and $a'' \in P$ such that: $a \leqslant a' \leqslant a''$, and by non-redundancy: $a = a''$, which implies that: $a = a' \in P'$, and hence $P \subseteq P'$. But similarly $P' \subseteq P$, and thus $\preccurlyeq$ is a partial order. Finally, it is easy to show that the monotonicity of γ implies the monotonicity of Γ. □

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Under more restrictive conditions, we can show that $(\mathbf{P}_{\mathrm{snr}}(A), \preccurlyeq)$ is a complete partial order.

Theorem 6
If A is meet-continuous,[3] *then $(\mathbf{P}_{\mathrm{snr}}(A), \preccurlyeq)$ is a cpo.*

Proof
To show that $(\mathbf{P}_{\mathrm{snr}}(A), \preccurlyeq)$ is complete, let $(P_i)_{i\in\mathbf{N}}$ be an increasing chain. Using the diagonal argument and the definition of $\preccurlyeq$, one can build a (possibly infinite) set of increasing chains $\{C_j\}_{j\in J}$, $J \subseteq \mathbf{N}$, $C_j = (c_{ji})_{i\in\mathbf{N}}$, such that for all $i\in\mathbf{N}: P_i = \{c_{ji}\}_{j\in J} - \{\bot\}$. But A being a complete lattice, each increasing chain C_j has a limit $l_j \in A$. The only possible candidate to the upper limit of the chain $(P_i)_{i\in\mathbf{N}}$ is therefore $\{l_j\}_{j\in J}$. But then for all $j \neq j'$:

$$\begin{aligned} l_j \wedge l_{j'} &= \bigvee C_j \wedge \bigvee C_{j'} \\ &= \bigvee_{i} (c_{ji} \wedge \bigvee C_{j'}) \\ &= \bigvee_{i,i'} (c_{ji} \wedge c_{j'i'}) \\ &= \bigvee_{i,i'} \bot = \bot \end{aligned}$$

which shows that $\{l_j\}_{j\in J}$ is strongly non-redundant and is the least upper bound of the chain $(P_i)_i \in \mathbf{N}$. □

Under the light of this theorem, one might think that it is a good idea to limit oneself to the strongly non-redundant subsets of A, since they form a complete partial order, and appear to be 'less arbitrary' than the general non-redundant subsets of A. But it depends a lot on the 'shape' of the abstract lattice A. Intuitively, for strongly non-redundant subsets to be useful, every abstract element $a \in A$ should be the least upper bound of a set of *atoms*. This can be formalized by saying that A should be an algebraic atomic lattice with a strongly non-redundant *basis* $\underline{A}$ of atoms such that:

$$\forall a \in A : a = \bigvee (\underline{A} \cap \downarrow a)$$

where $\downarrow a = \{a' \in A : a' \leqslant a\}$ is the principal ideal generated by a. Standard examples of such lattices are $(\mathbf{P}(S), \subseteq)$ and the interval lattice $\mathbf{I}(\mathbf{Z})$, with bases $\{\{s\}\}_{s\in S}$ and $\{[i, i]\}_{i\in\mathbf{Z}}$, respectively. Counter-examples are complete total orders, for which every subset with more than two elements is necessarily redundant. More generally, one can prove the following theorem.

Theorem 7
Let $(A, \bot, \top, \leqslant, \vee, \wedge)$ be an upper approximation of $\mathbf{P}(S)$, such that γ is one-to-one, $\gamma(\bot) = \varnothing$, and:

$$[s] \neq [s'] \Rightarrow [s] \wedge [s'] = \bot \quad (\text{where } [s] = \alpha(\{s\})).$$

Then A is an algebraic atomic lattice, $\underline{A} = \{[s] : s \in S\}$ is a strongly non-redundant basis

[3] A complete lattice L is meet-continuous (see Gierz *et al.*, 1980, p. 30) if for every directed subset $D \subseteq L$ and every element $x \in L : x \wedge \bigvee D = \bigvee \{x \wedge d : d \in D\}$.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

of A, and $\{\gamma(\underline{a}):\underline{a}\in\underline{A}\}$ is a partition of S. Moreover, if γ is $\vee$-continuous, then A is meet-continuous.

Proof

We first note that γ being one-to-one, $\alpha\circ\gamma=\mathrm{Id}_A$, γ is $\wedge$-continuous and α is $\cup$-continuous (Cousot, 1978, theorem 4.2.7.0.3, p. 4.33). Suppose now that there exists $s\in S$ such that $[s]=\perp$. Then $[s]=\alpha(\{s\})\leqslant\perp$ and by definition of Galois connections, $\{s\}\subseteq\gamma(\perp)=\varnothing$ which is impossible. Thus $\underline{A}$ is strongly non-redundant. But:

$$\begin{aligned} x\in\underline{A}\cap\downarrow a &\Leftrightarrow \exists s\in S: x=\alpha(\{s\})\leqslant a\\ &\Leftrightarrow \exists s\in S: x=[s]\wedge\{s\}\subseteq\gamma(a)\\ &\Leftrightarrow \exists s\in\gamma(a): x=[s]. \end{aligned}$$

Therefore $\underline{A}\cap\downarrow a=\beta(a)=\{[s]:s\in\gamma(a)\}$. In order to show that $a=\bigvee\beta(a)$, we shall first show that $\gamma(a)=\bigcup_{\underline{a}\in\beta(a)}\gamma(\underline{a})$. But $\gamma\circ\alpha\supseteq\mathrm{Id}_{\mathbf{P}(S)}$ and thus:

$$\begin{aligned} \bigcup_{\underline{a}\in\beta(a)}\gamma(\underline{a}) &= \bigcup_{s\in\gamma(a)}(\gamma\circ\alpha)\{s\})\\ &\supseteq \bigcup_{s\in\gamma(a)}\{s\}=\gamma(a). \end{aligned}$$

Conversely, let $x\in\bigcup_{s\in\gamma(a)}\gamma([s])$. Then there exists $\{s\}\subseteq\gamma(a)$ such that $\{x\}\subseteq\gamma([s])$, and hence by monotonicity of α:

$$\alpha(\{x\})\leqslant[s]=\alpha(\{s\})\leqslant\alpha(\gamma(a))=a$$

which implies that $\{x\}\subseteq\gamma(a)$, that is, $x\in\gamma(a)$. Therefore:

$$\begin{aligned} a=(\alpha\circ\gamma)(a)&=\alpha(\bigcup_{\underline{a}\in\beta(a)}\gamma(\underline{a}))\\ &=\bigvee_{\underline{a}\in\beta(a)}(\alpha\circ\gamma)(\underline{a})=\bigvee\beta(a). \end{aligned}$$

Now, when γ is $\vee$-continuous, then for every $a\in A$ and every subset $X\subseteq A$:

$$\begin{aligned} \gamma(a\wedge\bigvee X)&=\gamma(a)\cap\bigcup\{\gamma(x):x\in X\}\\ &=\bigcup\{\gamma(a)\cap\gamma(x):x\in X\}\\ &=\bigcup\{\gamma(a\wedge x):x\in X\} \end{aligned}$$

and therefore, α being $\cup$-continuous:

$$\begin{aligned} a\wedge\bigvee X&=(\alpha\circ\gamma)(a\wedge\bigvee X)\\ &=\bigvee\{(\alpha\circ\gamma)(a\wedge x):x\in X\}\\ &=\bigvee\{a\wedge x:x\in X\} \end{aligned}$$

which shows that A is meet-continuous. Finally, $\gamma([s])\neq\gamma([s'])$ implies that $[s]\neq[s']$, and γ being $\wedge$-continuous, $\gamma([s])\cap\gamma([s'])=\gamma([s]\wedge[s'])=\gamma(\perp)=\varnothing$, which proves that $\{\gamma(\underline{a}):\underline{a}\in\underline{A}\}$ is a (set-theoretic) partition of S. □

What we need now in order to complete our framework is to define elementary widening operators ∇ such that: $P\preccurlyeq P\nabla P'$ and $\Gamma(P')\subseteq\Gamma(P\nabla P')$. There are, of

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

course, many ways to define these operators. When working with $\mathbf{P}_{\mathrm{nr}}(A)$, the most precise widening operator can be defined by:

$$P \nabla P' = (P \cup P') - \{a' \in P', \exists\, a \in P : a' < a\}.$$

In fact, this widening operator behaves rather like a join operator. More generally, at each step of the computation, one can choose (either deterministically or not) a subset P_0 of the current invariant P to coalesce. The idea of such a generalization is to replace P by:

$$(P \cup \{a_0\}) - \{a \in P : a < a_0\}$$

where a_0 is any element greater than $\bigvee P_0$. Of course, one might choose to define a very poor widening, which does not improve the expressible properties of the framework, by:

$$P \nabla P' = \{\bigvee (P \cup P')\}.$$

It is easy to see that these definitions turn $\mathbf{P}_{\mathrm{nr}}(A)$ into a representation framework abstracting the lattice L whenever the 'generalizations' are properly used. It is difficult to say more about these generalizations since widening operators are well known to be highly lattice-dependent. When working with $\mathbf{P}_{\mathrm{snr}}(A)$, widening operators are even more difficult to define in the general case, and we shall only develop an example in Section 5.1. Note that in practice, one will always work with the set of *finite* strongly non-redundant subsets of A which is generally not a cpo, so the completeness of $\mathbf{P}_{\mathrm{snr}}(A)$ will not help. Finally, note that the definition of the widening operator has a great influence over the quality of the result of the computation, as we shall see in Section 5.2. Therefore, the general idea one should follow in the definition of a widening operator $(\nabla_i)_{i \in \mathrm{N}}$ should be to use very precise elementary operators (i.e. join-like) at the beginning of an iteration sequence, and to generalize only after these operators have precisely defined the 'shape' of the least fixed point. However, as we shall see in Section 5.2, there are also cases where it can be a good idea to alternate join-like operators and generalizations.

4.2 Basic functional partitioning

A central problem in abstract interpretation is to find a safe approximation of a least fixed point F that belongs to a functional lattice $C \to B$, where $(B, \bot, \top, \sqsubseteq, \sqcup, \sqcap)$ is itself a lattice. For instance, for very simple, non-recursive programming languages, C is usually the finite set of lexical control points, and B the powerset of run-time memory states. But for more complicated, recursive languages, a control point is more naturally defined as a subpart of the run-time stack, and C is infinite. More generally, there are cases where it can be interesting to consider that control points are indeed *execution traces* and not only static control points. Finally, in the minimal function graph approach, C is the set of admissible inputs of program functions, and B is the lattice of possible outputs, the bottom value of B being used to denote nondetermination. In this paper, we shall refer to the elements of the possibly infinite set C as *control points*.

Very often however, there is no need to know the value of F for every control point, and it is sufficient to determine a safe approximation $F_i \sqsupseteq \sqcup \{F(c) : c \in C_i\}$, of the

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

values taken by F over each subset of a given *partition* $\{C_i\}_{i \in I}$ of C. This partition can be defined in a very natural way by assigning a *token* to each control point. This method has been proposed for instance in Sharir and Pnueli (1981) and Jones (1982), where tokens are used to group execution traces and coalesce the memory states associated with them.

If the set of tokens is finite, then the framework is said to be *partitioned* (see Cousot, 1981, p. 315) and the problem is equivalent to the resolution of a finite system of semantic equations. This is the case for instance in Bourdoncle (1990), where a token is assigned to each run-time stack. These tokens model the 'shape' of the stack (pointers, control stack ...) and generalize the tokens used in Sharir and Pnueli (1981) that only took into account the control part of the run-time stacks.

However, if the set of tokens is infinite (or very large) and one has no idea of a good way of defining a finite partition, then the original problem of finding a safe and finitely represented approximation of F remains to be solved. The idea is then to 'lift' F so that it operates on sets of control points, and to dynamically calculate a partition of C, instead of it being 'hard wired'.

We are going to study two general methods for doing this *dynamic partitioning*. For each of these methods, we suppose that there is a basic (and supposedly not satisfactory) way of *finitely* representing sets of control points, and we intend to build a *representation* from this initial approximation. We shall therefore suppose that (A, $\perp$, $\top$, $\leqslant$, $\vee$, $\wedge$) is an upper approximation of ($\mathbf{P}(C)$, $\emptyset$, C, $\subseteq$, $\cup$, $\cap$), and call (α, γ) the Galois connection between the two lattices. We shall also suppose that γ is one-to-one and that $\gamma(\perp) = \emptyset$, which implies that γ is $\wedge$-continuous. The elements of A will be called *abstract control points* and the elements of $T = \{[c] : c \in C\}$, where $[c] = \alpha(\{c\})$, will be called the *tokens*. Theorem 7 shows that whenever T is strongly non-redundant, then A is an algebraic atomic lattice, and T defines a partition of C, but this hypothesis will not be necessary. Our definition therefore generalizes the classical notion of token. Finally, the elements of B will be called *abstract values*.

The first representation that we shall define is based on the very naïve observation that every abstract control point $a \in A$ implicitly defines a (possibly infinite) set of control points $\gamma(a)$ that we shall informally call a 'region'. Therefore, an easy way to approximate a function from C into B is to 'cover' the region over which this function is different from $\perp$ by a finite subset of A, and to associate an abstract value b with each element a of this subset.

If the regions of such a representation $P \subseteq A \times B$ do not overlap, the natural meaning of P will map every control point c to the unique abstract value b associated with the element a by which its token $[c]$ is covered, or to $\perp$ if its token is not covered. However, if the regions overlap, the meaning of P can be defined in several ways. We shall study in this section the most natural idea which is to map every control point c to the *union* of the abstract values associated with the abstract control point a with which its token *intersects*. We will show that these representations can be constrained in order to form a partial order compatible with this meaning, and then explain how widening operators can be effectively designed. We shall then study, in the next section, a different and non standard meaning of overlapping representations that is better suited to generalization.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Definition 8

A subset P of $A \times B$ is said to be normalized if:

$$\forall \langle a, b \rangle \in P, \quad \forall \langle a', b' \rangle \in P : a = a' \Rightarrow b = b'.$$

For every normalized subset P, and $a, a_1, a_2 \in A$, we define:

$$\begin{aligned} A(P) &= \{a \in A : \exists\, b \in B : \langle a, b \rangle \in P\} \\ C(P) &= \bigcup \{\gamma(a) : \exists\, b \in B : \langle a, b \rangle \in P\} \\ P(a) &= \sqcup\, \{b \in B : \langle a, b \rangle \in P\} \\ P^{\natural}(a_1, a_2) &= \{b : \langle a, b \rangle \in P \wedge a_1 \leqslant a \leqslant a_2\}. \end{aligned}$$

The set $A(P)$ is called the *domain* of P, and the abstract value $P(a)$ is the *image* of a by P. The set $C(P)$ is the *concrete domain* of P, i.e. the 'region' of C covered by the domain of P. For a normalized subset P of $A \times B$, the image $P(a)$ of every element of the domain of P is the unique element b such that $\langle a, b \rangle \in P$. We call $\mathbf{P}(A, B)$ the set of normalized subsets P whose domains do not contain $\perp$, and $\mathbf{P}_{\mathrm{nr}}(A, B)$ (resp. $\mathbf{P}_{\mathrm{snr}}(A, B)$) the set of normalized subsets which have a non-redundant (resp. strongly non-redundant) domain. Obviously:

$$\mathbf{P}_{\mathrm{snr}}(A, B) \subseteq \mathbf{P}_{\mathrm{nr}}(A, B) \subseteq \mathbf{P}(A, B).$$

We then define the meaning $\Gamma(P)$ of a representation $P \in \mathbf{P}(A, B)$ by:

$$\Gamma(P)(c) = \mathcal{M}_P[c]$$

where the monotonic function $\mathcal{M}_P : A \rightarrow B$ is defined by:

$$\mathcal{M}_P(x) = \bigsqcup_{\substack{a \in A(P) \\ a \wedge x \neq \perp}} P(a).$$

When T is a strongly non-redundant basis of A, we obviously have:

$$\forall \tau \in T, \quad \forall a \in A : a \wedge \tau \neq \perp \Leftrightarrow \tau \leqslant a$$

and therefore: $\quad \Gamma(P)(c) = \sqcup\, \{b : \langle a, b \rangle \in P \wedge [c] \leqslant a\}$

which states that each control point c is mapped to the union of the abstract values b attached to the elements of $A(P)$ by which its token $[c]$ is 'covered', or to $\perp$ if its token is not covered. Note that if $A(P)$ is strongly non-redundant, then an element of the basis is at most covered by a single element in $A(P)$. It is worth mentioning at this stage that although any set of tokens can be chosen, it seems reasonable to impose that T be strongly non-redundant. To see the problem, let us choose $C = \mathbf{Z}$, $A = \mathbf{Z}_{\perp}$, and $[c] = c$. Then $\gamma(c) = \{\omega^-, \ldots, c\}$, the lattice $(A, \perp, \omega^+, \leqslant, \mathbf{max}, \mathbf{min})$ is *totally ordered*, and:

$$\forall a, a' \in A : a \neq \perp \wedge a' \neq \perp \Leftrightarrow a \wedge a' \neq \perp$$

which shows that $\Gamma(P)$ is constant over C and:

$$\Gamma(P)(c) = \sqcup\, \{b : \langle a, b \rangle \in P\}.$$

Intuitively, all the 'regions' defined by the abstract control points overlap, and therefore, there is no way to distinguish between the different abstract values $b \in B$ of the representation. We are now going to show that $\mathbf{P}_{\mathrm{nr}}(A, B)$ and $\mathbf{P}_{\mathrm{snr}}(A, B)$ can be turned into partial orders.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Theorem 9
$\mathbf{P}_{\mathrm{nr}}(A, B)$ and $\mathbf{P}_{\mathrm{snr}}(A, B)$ are partial orders for the binary relation:

$$P \preccurlyeq P' \Leftrightarrow \forall \langle a, b \rangle \in P, \quad \exists \langle a', b' \rangle \in P' : a \preccurlyeq a' \wedge b \sqsubseteq b'$$

and the meaning function Γ is monotonic over $\mathbf{P}_{\mathrm{nr}}(A, B)$ and $\mathbf{P}_{\mathrm{snr}}(A, B)$.

The proof is straightforward. Note that every function F in $C \rightarrow B$ can always be finitely abstracted by $\{\langle \top, \top \rangle\}$ and, when T is non-redundant, it can also be safely abstracted by $\{\langle [c], F(c) \rangle : c \in C\}$. Therefore, defining a widening operator over $\mathbf{P}_{\mathrm{nr}}(A, B)$ will turn $\mathbf{P}_{\mathrm{nr}}(A, B)$ into a representation of $C \rightarrow B$. Elementary widening operators can be defined as follows:

- We first define the domain of $P \nabla P'$ by:

$$A(P \nabla P') = A(P) \nabla_b A(P')$$

where ∇_b is any basic partitioning widening operator defined in Section 4.1.

- Then, for every a in the domain of $P \nabla P'$, we define the image of a by:

$$(P \nabla P')(a) = b$$

where $b \in B$ is any abstract value such that:

$$b \sqsupseteq \mathcal{M}_P(a) \sqcup \mathcal{M}_{P'}(a).$$

To prove that $P \preccurlyeq P \nabla P'$, we remark that by definition $A(P) \preccurlyeq A(P \nabla P')$, and therefore:

$$\forall \langle a, b \rangle \in P, \quad \exists a' \in A(P \nabla P') : a \preccurlyeq a'$$

hence: $b = P(a) \sqsubseteq \mathcal{M}_P(a) \sqsubseteq \mathcal{M}_P(a') \sqsubseteq \mathcal{M}_P(a') \sqcup \mathcal{M}_{P'}(a') \sqsubseteq (P \nabla P')(a')$

and thus: $\exists \langle a', b' \rangle \in P \nabla P' : a \preccurlyeq a' \wedge b \sqsubseteq b'$.

Let us now prove that $\Gamma(P') \sqsubseteq \Gamma(P \nabla P')$. So let $\langle a', b' \rangle \in P'$ and $x \in A$ be such that $x \wedge a' \neq \bot$. By construction, we have:

$$C(P') = \bigcup_{a \in A(P')} \gamma(a) \subseteq \bigcup_{a \in A(P \nabla P')} \gamma(a) = C(P \nabla P')$$

and therefore, there exists a pair $\langle a, b \rangle \in P \nabla P'$ such that $a \wedge x \wedge a' \neq \bot$, otherwise, γ being $\wedge$-continuous:

$$\begin{aligned}
\gamma(x \wedge a') &= \gamma(x \wedge a') \cap \gamma(a') \\
&\subseteq \gamma(x \wedge a') \cap \bigcup_{a \in A(P')} \gamma(a) \\
&\subseteq \bigcup_{a \in A(P \nabla P')} (\gamma(x \wedge a') \cap \gamma(a)) \\
&\subseteq \bigcup_{a \in A(P \nabla P')} \gamma(a \wedge x \wedge a') = \varnothing
\end{aligned}$$

and thus, γ being one-to-one, $x \wedge a' = \bot$, which is absurd. Consequently, $a \wedge a' \neq \bot$, and therefore:

$$b \sqsupseteq \mathcal{M}_P(a) \sqcup \mathcal{M}_{P'}(a) \sqsupseteq \mathcal{M}_{P'}(a \wedge a') \sqsupseteq b'$$

where shows that $\mathcal{M}_{P'}(x) = \bigsqcup_{\substack{(a', b') \in P' \\ x \wedge a' \neq \bot}} (b') \sqsubseteq \bigsqcup_{\substack{(a, b) \in P \nabla P' \\ x \wedge a \neq \bot}} (b) = \mathcal{M}_{P \nabla P'}(x)$

and hence: $\Gamma(P') \sqsubseteq \Gamma(P \nabla P')$.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Provided that condition (v) of Definition 1 is satisfied, $(\mathbf{P}_{\mathrm{nr}}(A,B), \preccurlyeq, \Gamma, \nabla)$ is thus a representation of the functional lattice $C \rightarrow B$.

4.3 Functional partitioning

The main interest of basic functional partitioning is that it is indeed very natural and easy to understand: the value mapped to a control point c by the meaning of a representation P is defined as the union of the abstract values b associated with the abstract control points a which have 'something in common' with its token $[c]$, i.e. $[c] \wedge a \neq \perp$. But basic functional partitioning has several shortcomings.

Firstly, as we noted earlier, basic functional partitioning is reasonably applicable only when the lattice A is an algebraic atomic lattice with a strongly non-redundant basis. This can be very annoying when approximating higher order functions for instance, since abstract functional lattices do not generally have a strongly non-redundant basis.

Secondly, there are cases where the ordering $\preccurlyeq$ over $\mathbf{P}_{\mathrm{nr}}(A,B)$ is not appropriate, and one would like abstract control points of representations to be maintained during iterative computations. This happens to be the case for interprocedural abstract interpretation since abstract control points naturally correspond to function calls in the fixed point computation algorithm, and the abstract call graph, which is generally needed to determine the set of recursive procedures, is therefore defined in terms of abstract control points. Of course, this goal can be easily achieved by slightly modifying the definition of $\preccurlyeq$ as follows:

$$\forall \langle a,b \rangle \in P, \quad \exists \langle a',b' \rangle \in P' : a = a' \wedge b \sqsubseteq b'.$$

However, this ordering has a major drawback with respect to the definition of widening operators in that, intuitively, the only way to generalize a representation P without losing too much information is to 'pave' C using strongly non-redundant subsets of A, because every extra element $\langle a',b' \rangle$ added to P leads to a loss of information for every token 'covered' by a'. This is illustrated by the left part of Figure 1 in the case of the interval lattice $\mathbf{I}(\mathbf{Z}^2)$. But this pavement can turn out to be

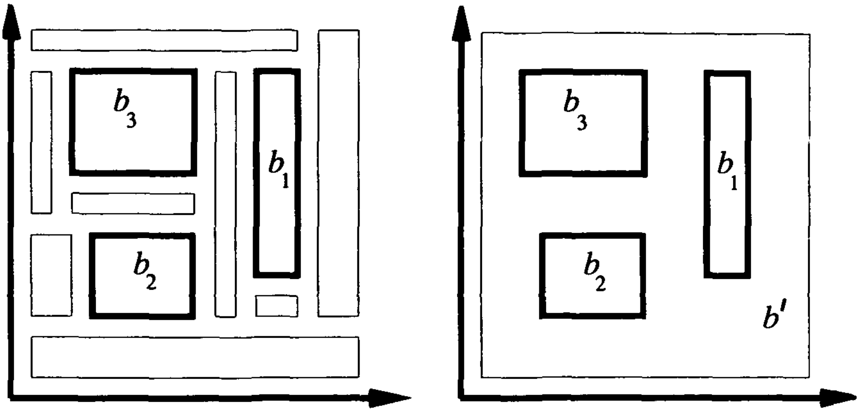

Fig. 1. Basic functional partitioning versus functional partitioning.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

much too complicated when working with sophisticated lattices such as the linear inequalities lattice, for instance. Worse, it can even be impossible to define such a pavement for lattices that do not have a strongly non-redundant basis. What we would like to do is therefore to map small regions $\{a_i\}_{i\in I}$ of C to a given set of values $\{b_i\}_{i\in I}$, while defining some kind of *default value* b' for a larger and possibly overlapping area a', as illustrated by the right part of Figure 1. This can be achieved by generalizing the definition of the meaning function Γ to every element P of $\mathbf{P}(A,B)$ by:

$$\Gamma(P)(c) = \mathcal{M}_P[c]$$

where $\mathcal{M}_P : A \to B$ is defined by:

$$\mathcal{M}_P(x) = \bigsqcup_{\substack{a\in A(P)\\ a\,\wedge\, x\,\neq\,\bot}} \mathcal{D}_P(a \wedge x, a) \quad \text{and} \quad \mathcal{D}_P(u,v) = \bigsqcap P^{\natural}(u,v).$$

Given two abstract control points u and v, $\mathcal{D}_P(u,v)$ is equal to the greatest lower bound of the abstract values b associated with the abstract control points a that belong to the (possibly empty) convex subset $\{x : u \leqslant x \leqslant v\}$. It is easy to see that this function is increasing in its first argument and decreasing in its second argument. The function $\mathcal{M}_P$ is therefore monotonic, and maps every abstract control point $x \in A$ to the union of the abstract values b associated with the *minimum* elements $a \in A(P)$ such that $x \wedge a \neq \bot$. The meaning of a representation in $\mathbf{P}_{\text{nr}}(A,B)$ is thus identical to the meaning defined in the previous section, since every element of a non-redundant domain $A(P)$ is minimum. The meaning of $\mathcal{M}_P$ for the representation $P = \{\langle a,b\rangle, \langle a', b'\rangle, \langle a'',b''\rangle\}$ and three particular values $a = [2,13]$, $a' = [10,18]$ and $a'' = [6,21]$, is illustrated in Figure 2 in the case of the interval lattice $\mathbf{I}(\mathbf{Z})$. The function $\mathcal{M}_P$ maps

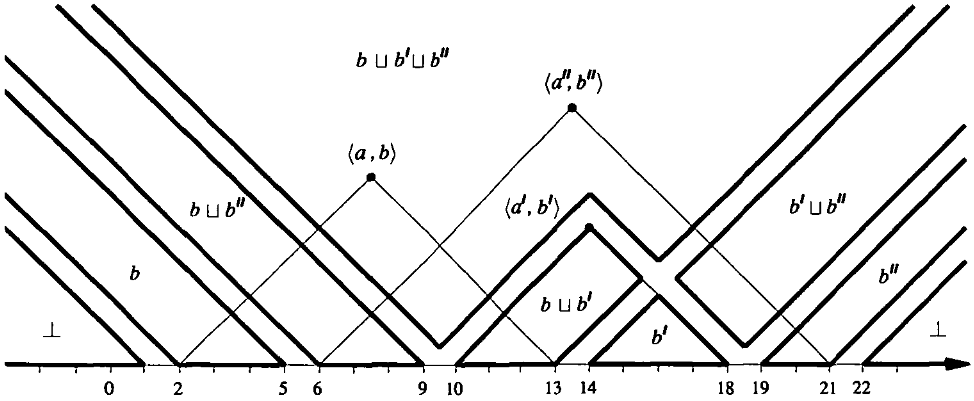

Fig. 2. Meaning of $\mathcal{M}_P$ for $P = \{\langle a,b\rangle, \langle a',b'\rangle, \langle a'',b''\rangle\}$.

an abstract control point, represented by a point on the plane using the usual encoding of intervals:

$$[x,y] \mapsto \langle (x+y)/2, (y-x)/2\rangle$$

to an abstract value in B. The function $\Gamma(P) : \mathbf{Z} \to B$ maps every integer k to $\mathcal{M}_P[k]$,

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

where the token $[k] = [k, k]$ is the one-point interval. Note how the intervals below a' are 'protected' from the value b'' associated with the abstract control point a''.

The problem however with this new meaning function Γ is that it is not monotonic over $\mathbf{P}(A, B)$. Intuitively, non-monotonicity arises when an element $\langle a', b'\rangle$ is added to a representation P and a' 'masks' a region previously mapped to a value greater than b'. This would be the case for instance in Figure 2 if $\langle a', b'\rangle$ was added to $\{\langle a, b\rangle, \langle a'', b''\rangle\}$ and $b' \sqsubset b''$. In order to avoid this situation, one can require that b' be greater than $\mathcal{S}_P(a')$, where the *smallest safe value* $\mathcal{S}_P(x)$ is defined by:

$$\mathcal{S}_P(x) = \bigsqcup_{\substack{a\in A(P)\\ \bot < x \leqslant a}} \mathcal{D}_P(x, a).$$

Intuitively, this condition ensures that the new value b' is at least the union of every value that a' could mask, i.e. the values associated with the minimum elements in $A(P)$ that are above a'. This intuition can be formalized by defining the relation $\preccurlyeq$ over $\mathbf{P}(A, B)$ as follows:

$$P \preccurlyeq P' \Leftrightarrow \begin{cases} \forall \langle a, b\rangle \in P, \quad \exists \langle a', b'\rangle \in P' : a = a' \wedge b \sqsubseteq b' \\ \forall \langle a', b'\rangle \in P' : a' \notin A(P) \Rightarrow b' \sqsupseteq \mathcal{S}_P(a'). \end{cases}$$

Note that since $\mathcal{S}_P(a) = P(a)$ for every a in the domain of P, this condition could also be written as follows:

$$P \preccurlyeq P' \Leftrightarrow \begin{cases} A(P) \subseteq A(P') \\ \forall \langle a', b'\rangle \in P' : b' \sqsupseteq \mathcal{S}_P(a'). \end{cases}$$

Theorem 10
$(\mathbf{P}(A, B), \preccurlyeq)$ *is a partial order, and* Γ *is monotonic over* $\mathbf{P}(A, B)$.

Proof
Let us first show that Γ is monotonic over $\mathbf{P}(A, B)$. So let us suppose that $P \preccurlyeq P'$. Then $A(P) \subseteq A(P')$, and for every $x \in A$ such that $x \neq \bot$:

$$\begin{cases} \mathcal{S}_{P'}(x) \sqsupseteq \displaystyle\bigsqcup_{\substack{a\in A(P)\\ \bot < x \leqslant a}} \mathcal{D}_{P'}(a \wedge x, a) \\ \mathcal{M}_{P'}(x) \sqsupseteq \displaystyle\bigsqcup_{\substack{a\in A(P)\\ a \wedge x \neq \bot}} \mathcal{D}_{P'}(a \wedge x, a). \end{cases}$$

But for every $a \in A(P)$ such that either $x \wedge a \neq \bot$ or $\bot < x \leqslant a$:

$$\mathcal{D}_{P'}(a \wedge x, a) = \bigsqcap_{\substack{\langle a', b'\rangle \in P'\\ a \wedge x \leqslant a' \leqslant a}} (b').$$

Let us now suppose that there exists $\langle a', b'\rangle \in P'$ such that $a \wedge x \leqslant a' \leqslant a$. Then by hypothesis:

$$b' \sqsupseteq \mathcal{S}_P(a') = \bigsqcup_{\substack{a''\in A(P)\\ a' \leqslant a''}} \mathcal{D}_P(a', a'') \sqsupseteq \bigsqcup_{\substack{a''\in A(P)\\ a' \leqslant a'' \leqslant a}} \mathcal{D}_P(a', a'').$$

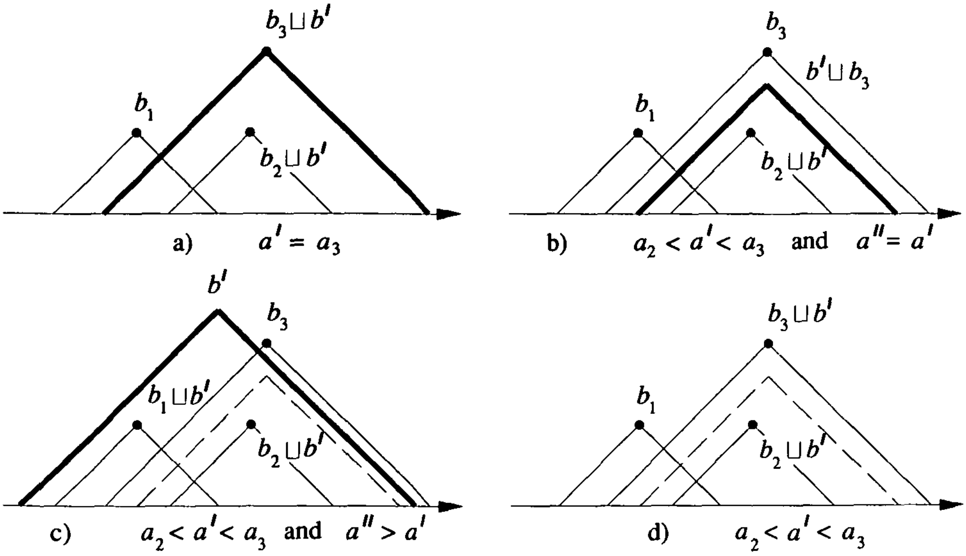

Fig. 3. Widening in a functional partitioning framework.

But $a \wedge x \leqslant a' \leqslant a'' \leqslant a$ implies that $\mathcal{D}_P(a \wedge x, a) \sqsubseteq \mathcal{D}_P(a', a'')$, and since $a \in A(P)$, the set $\{a'' \in A(P) : a' \leqslant a'' \leqslant a\}$ is non-empty and thus:

$$b' \sqsupseteq \mathcal{D}_P(a \wedge x, a)$$

which implies that:
$$\mathcal{D}_{P'}(a \wedge x, a) \sqsupseteq \mathcal{D}_P(a \wedge x, a)$$

and therefore:
$$\begin{cases} \mathcal{S}_{P'}(x) \sqsupseteq \mathcal{S}_P(x) \\ \mathcal{M}_{P'}(x) \sqsupseteq \mathcal{M}_P(x). \end{cases}$$

Consequently, since $\mathcal{S}_Q(\bot) = \mathcal{M}_Q(\bot) = \bot$ for every representation Q, then $\mathcal{M}_P \sqsubseteq \mathcal{M}_{P'}$, $\mathcal{S}_P \sqsubseteq \mathcal{S}_{P'}$, and Γ is monotonic over $\mathbf{P}(A, B)$. Finally, $\preccurlyeq$ being trivially reflexive and antisymmetric, let us prove that it is also transitive. So let $P \preccurlyeq P' \preccurlyeq P''$. Then for every $\langle a'', b'' \rangle \in P''$ such that $a'' \notin A(P)$, either $a'' \in A(P')$, and thus $b'' \sqsupseteq b' \sqsupseteq \mathcal{S}_P(a'')$, or else:

$$b'' \sqsupseteq \mathcal{S}_{P'}(a'') \sqsupseteq \mathcal{S}_P(a'')$$

which proves that $P \preccurlyeq P''$. □

We have proven that $(\mathbf{P}(A, B), \preccurlyeq)$ is a partial order, but we must note that the meaning function is not strictly monotonic, i.e. one can find two distinct representations such that $P_1 \prec P_2$ and $\Gamma(P_1) = \Gamma(P_2)$. This holds for instance whenever $b_1 \sqsubset b_2$ for:

$$P_i = \{\langle [1,1], b \rangle, \langle [2,3], b' \rangle, \langle [1,3], b_i \rangle\} \quad (i \in \{1,2\})$$

since:
$$\Gamma(P_1) = \Gamma(P_2) = \{1 \mapsto b, 2 \mapsto b', 3 \mapsto b'\}.$$

This problem can be solved by considering well chosen subsets of $\mathbf{P}(A, B)$, but we shall not study this problem here. We have thus defined a very flexible framework such that abstract control points are added and never removed during a given computation. Moreover, the information associated with each control point is only

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

allowed to increase. Finally, we have a very easy criterion to check whether or not a pair $\langle a,b\rangle \in A\times B$ can be safely added to a representation, which is very important if one wishes to be able to generalize at some point of the computation. What we need now is to define elementary widening operators, i.e. operators such that $P \preccurlyeq P \nabla P'$ and $\Gamma(P') \sqsubseteq \Gamma(P\nabla P')$.

In the rest of this section, we shall only consider finite representations, for they have the greatest practical interest, and for the sake of simplicity, we start by defining $P'' = P\nabla P'$ for a singleton $P' = \{\langle a',b'\rangle\}$. There are basically three cases in the definition of P''. Each one is illustrated in Figure 3, where we have taken $A = \mathbf{I}(\mathbf{Z})$, and $P = \{\langle a_1,b_1\rangle, \langle a_2,b_2\rangle, \langle a_3,b_3\rangle\}$:

- If $a' \in A(P)$, then for every $\langle a,b\rangle \in P$ such that $a \leqslant a'$, the replacement of b by any element greater than $b' \sqcup b$ ensures that the meaning of P'' is greater than the meaning of $\{\langle a',b'\rangle\}$, and at the same time that $P \preccurlyeq P''$ (Figure 3a).
- If $a' \notin A(P)$. Suppose one wishes to add this new abstract control point to the domain of the current representation P. Obviously $\langle a',b'\rangle$ cannot simply be added to P. But adding any pair $\langle a'',b''\rangle$ such that $a'' \geqslant a'$ and $b'' \sqsupseteq b' \sqcup \mathscr{S}_P(a'')$ will ensure that $P \preccurlyeq P''$. However, as in the previous case, every $\langle a,b\rangle \in P$ such that $a \leqslant a''$ may 'mask' the value b' to several elements of the basis. Hence, each value b must be replaced by an element greater than $b' \sqcup b$ to ensure that the meaning of P'' is greater than the meaning of $\{\langle a',b'\rangle\}$ (see Figures 3b, c).
- There are cases however where $a' \notin A(P)$ but one does not want to add a' to the domain of P. In fact, this will almost always be the case when working with finite representations. There are then two subcases to consider. If $\gamma(a') \subseteq C(P)$, every control point c represented by a' is already represented by at least one element a such that $\langle a,b\rangle \in P$ and $[c] \leqslant a$, and replacing b by $b \sqcup b'$ will ensure that the meaning of P'' is greater than the meaning of $\{\langle a',b'\rangle\}$ – provided of course that, as in the previous cases, every b'' such that $\langle a'',b''\rangle \in P$ and $a'' \leqslant a$ be replaced by an element greater than $b' \sqcup b''$ (Figure 3d). But if $\gamma(a') \nsubseteq C(P)$, then the region defined by a' contains 'new' control points, and there is no way to avoid the addition of a' to $A(P)$. The previous case must thus be applied, choosing for instance $a'' = \top$.

  Note that, in practice, the test $\gamma(a') \subseteq C(P)$ will always be approximated, and a given abstract control point a' will *not* be added to the domain of P *only* if there exists $\langle a,b\rangle \in P$ such that $a' \leqslant a$. Such an approximation will thus generally imply the non-stability of the widening operator (cf. Section 3).

This definition shows that functional partitioning is well suited to generalization processes, for it enables one to easily generalize without losing too much information. In order to complete our framework, we now define $P\nabla Q$ for any finite representation $Q \in \mathbf{P}(A,B)$ by arbitrarily numbering the elements $\langle a_i,b_i\rangle$, $i \in [1,k]$ of Q, and adding them one at a time to P, i.e.:

$$P\nabla Q = ((P\nabla Q_1)\ldots)\nabla Q_k$$

where $Q_i = \{\langle a_i,b_i\rangle\}$. This definition trivially implies that

$$P\nabla Q \succcurlyeq ((P\nabla Q_1)\ldots)\nabla Q_{k-1} \succcurlyeq \ldots \succcurlyeq P$$

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

and thanks to the next theorem:

$$\begin{aligned}\Gamma(P \nabla Q) &= \Gamma(((P \nabla Q_1) \ldots) \nabla Q_k)\\ &\sqsupseteq \Gamma(Q_1) \sqcup \ldots \sqcup \Gamma(Q_k)\\ &\sqsupseteq \Gamma(Q_1 \cup \ldots \cup Q_k)\\ &= \Gamma(Q)\end{aligned}$$

which shows that condition (iv) of Definition 1 is satisfied.

Theorem 11
For every $Q_1, Q_2 \in \mathbf{P}(A, B)$ such that $Q_1 \cup Q_2 \in \mathbf{P}(A, B)$:

$$\Gamma(Q_1 \cup Q_2) \sqsubseteq \Gamma(Q_1) \sqcup \Gamma(Q_2).$$

Proof
We first remark that for every $u, v \in A$:

$$(Q_1 \cup Q_2)^{\natural}(u, v) = Q_1^{\natural}(u, v) \cup Q_2^{\natural}(u, v)$$

thus: $$\mathscr{D}_{Q_1 \cup Q_2}(u, v) = \mathscr{D}_{Q_1}(u, v) \sqcap \mathscr{D}_{Q_2}(u, v) \sqsubseteq \mathscr{D}_{Q_1}(u, v), \quad \mathscr{D}_{Q_2}(u, v)$$

and therefore:

$$\begin{aligned}\mathscr{M}_{Q_1 \cup Q_2}(x) &= \bigsqcup_{\substack{a \in A(Q_1 \cup Q_2)\\ a \wedge x \neq \perp}} \mathscr{D}_{Q_1 \cup Q_2}(a \wedge x, a)\\ &= \bigsqcup_{\substack{a \in A(Q_1)\\ a \wedge x \neq \perp}} \mathscr{D}_{Q_1 \cup Q_2}(a \wedge x, a) \sqcup \bigsqcup_{\substack{a \in A(Q_2)\\ a \wedge x \neq \perp}} \mathscr{D}_{Q_1 \cup Q_2}(a \wedge x, a)\\ &\sqsubseteq \mathscr{M}_{Q_1}(x) \sqcup \mathscr{M}_{Q_2}(x)\end{aligned}$$

which proves that $\Gamma(Q_1 \cup Q_2) \sqsubseteq \Gamma(Q_1) \sqcup \Gamma(Q_2)$. □

5 Applications

We now present two possible applications of dynamic partitioning. The first one is simply the application of basic partitioning to the bi-dimensional interval lattice $\mathbf{I}(\mathbf{Z}^2)$. Its interest is rather academic, but we shall use this example to illustrate how widening operators can be effectively built. In the second example, we will show how a precise description of the input/output behaviour of a program function can be computed using functional partitioning. This example will exemplify the case where the shape of a program invariant cannot be predicted and has to be considered as an output of the fixed point computation itself.

5.1 Multi-intervals

The aim of this section is to show how sets of bi-dimensional intervals can be used to represent sets of integer pairs. Following the method developed in Section 4.1, we can either use non-redundant subsets or strongly non-redundant subsets of $\mathbf{I}(\mathbf{Z}^2)$. Note that strongly non-redundant subsets of $\mathbf{I}(\mathbf{Z}^2)$ are always larger than non-

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

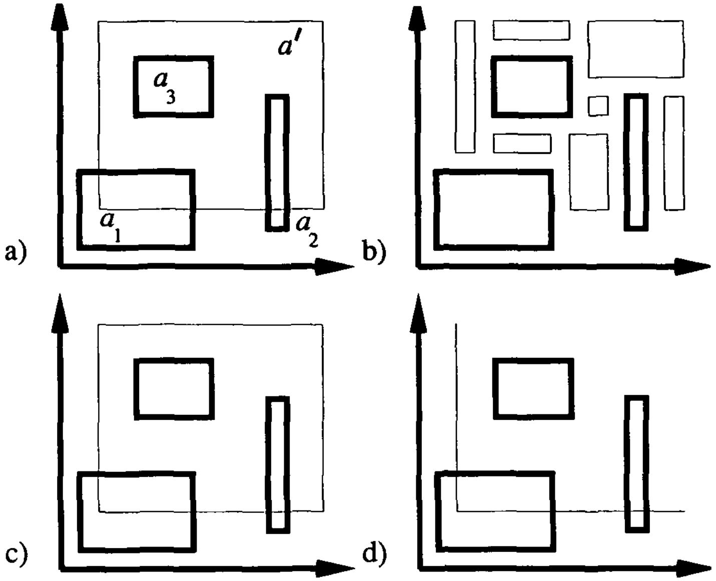

Fig. 4. Widenings over $\mathbf{P}_{nr}(\mathbf{I}(\mathbf{Z}^2))$.

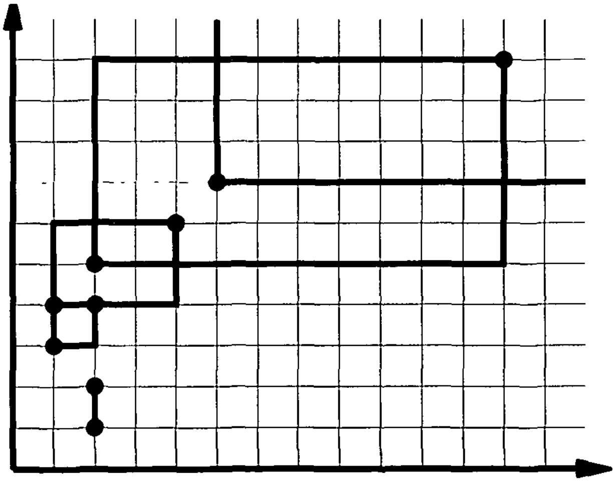

Fig. 5. Safe representation of lfp(Φ).

redundant subsets. We are going to illustrate the ideas that can be used to build elementary widening operators over such subsets. Figure 4a shows an element $P = \{a_1, a_2, a_3\}$ of $\mathbf{P}_{snr}(\mathbf{I}(\mathbf{Z}^2))$ plus an extra element $a' \in \mathbf{I}(\mathbf{Z}^2)$. We wish to calculate $P' = P \nabla \{a'\}$. Figure 4b illustrates how P' can be defined using a join-like operator. Note that such an operator might be very difficult to implement. So let us focus on $\mathbf{P}_{nr}(\mathbf{I}(\mathbf{Z}^2))$. We shall define two elementary operators. The first one ∇_j (Figure 4c) behaves like a join operator, and shall be used in the first steps of a computation. The second one ∇_g (Figure 4d) computes a generalization as follows. Using the widening operator ∇_I over $\mathbf{I}(\mathbf{Z}^2)$ defined in Section 2, one first computes $a'' = (\bigvee P) \nabla_I a'$. Intuitively, $\bigvee P$ is used as a reference to determine in 'which direction' a' is 'moving'.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press

Of course, different references can be used, such as the most recently added elements of P for instance. Finally, P' is calculated by removing the redundant elements of $P \cup \{a''\}$. We shall use these two elementary widening operators to compute a non-trivial, safe and finitely represented approximation of the least fixed point of $\Phi : \mathbf{P}(\mathbf{N}^2) \rightarrow \mathbf{P}(\mathbf{N}^2)$ defined by:

$$\Phi(M) = \{\langle 2,1\rangle, \langle 2,2\rangle\} \cup \{\langle \lfloor xy/2 \rfloor, x+y\rangle\}_{\langle x,y\rangle \in M}.$$

This least fixed point cannot be finitely represented in any usual lattice used for modelling sets of integer pairs, such as the linear inequalities lattice of Cousot and Halbwachs (1978), for instance. But one can very easily define a safe approximation $\Phi^{\#}$ of Φ by:

$$\Phi^{\#}(P) = \{\langle [2,2],[1,2]\rangle\} \cup \{\Phi_1^{\#}(X,Y)\}_{\langle X,Y\rangle \in P}$$

where $$\Phi_1^{\#}([i,s],[i',s']) = \langle [\lfloor ii'/2 \rfloor, \lfloor ss'/2 \rfloor], [i+i', s+s']\rangle.$$

Thus, using the framework of Section 3 with the widening operator:

$$\nabla = (\nabla_j^3 \nabla_g^{\omega}) = (\nabla_j, \nabla_j, \nabla_j, \nabla_g \ldots)$$

one can finitely compute the following non-trivial approximation displayed in Figure 5:

$$\{\langle [2,2],[1,2]\rangle, \langle [1,2],[3,4]\rangle, \langle [1,4],[4,6]\rangle, \langle [2,12],[5,10]\rangle, \langle [5,\omega^+],[7,\omega^+]\rangle\}.$$

5.2 *Minimal function graphs*

We now present an application of functional partitioning to interprocedural abstract interpretation, which in fact originally motivated this work. Let us suppose that one has a *program function* $\Phi : \mathbf{Z} \rightarrow \mathbf{Z}_\perp$ such as the ***Loop*** function introduced in Section 2 or MacCarthy's 91-function defined by:

```
fun Mc n = if (n > 100) then
                n−10
             else
                Mc(Mc(n+11)).
```

We wish to determine a safe approximation of the minimal function graph of Φ for a given set of input data specifications. We do not formally describe a minimal function graph semantics, but rather give an intuition about the way finite representations of minimal function graphs can be computed using functional partitioning. As hinted at the end of Section 2, we shall abstract minimal function graphs using representations in $\mathbf{P}(\mathbf{I}(\mathbf{Z}), \mathbf{I}(\mathbf{Z}))$, and input data specifications using representations in $\mathbf{P}_{\text{nr}}(\mathbf{I}(\mathbf{Z}))$. So let us suppose that we have an initial representation I_0 of the set of input data specifications. We can define the first representation of the minimal function graph of Φ with respect to the input data specification I_0 by:

$$P_0 = \{\langle i_0, \perp\rangle\}_{i_0 \in I_0}.$$

The meaning of a representation P is the one introduced in Section 4.3, that is:

$$\Gamma(P)(n) = \mathcal{M}_P[n,n]$$

where $$\mathscr{M}_P(x) = \bigvee_{\substack{\langle i, v\rangle \in P \\ x \wedge i \neq \bot}} \bigwedge \{v' : (x \wedge i) \leqslant i' \leqslant i\}_{\langle i', v'\rangle \in P}.$$

Note that, contrary to what is proposed by Jones and Mycroft (1986), we have not introduced a special value '!' to denote non-termination. Therefore, at the end of the computation, $\Gamma(P)(n) = \bot$ either means that Φ has never been called with n as argument, or that it has been called and looped. Note that this is not too important, since these two interpretations can be easily distinguished by looking at the domain of the representation. The approximate minimal function graph is therefore the limit of the increasing chain defined by:

$$P_{k+1} = P_k \nabla_k \Phi^{\#}(P_k)$$

where ∇_k is an elementary widening operator, and $\Phi^{\#}(P)$ is defined as follows:

(1) For every $\langle i, v\rangle$ in P, an updated value $v' \sqsupseteq v$ of v is computed by applying the definition of Φ to the set of values denoted by i and replacing the values of the recursive calls $\Phi(i')$ by $\mathscr{M}_P(i')$. The latter is the best approximation of $\Phi(i')$ that can be given using the current approximation of Φ.

(2) Then $\Phi^{\#}(P) = \{\langle i', \bot\rangle\}_{i' \in I'} \nabla_k \{\langle i, v'\rangle\}_{\langle i, v\rangle \in P}$, where I' is the set of *new* abstract control points over which Φ has been called in step 1.

In other words, we compute an updated approximation v' of the value v of Φ over each abstract control point i in the domain of the representation P, taking into account the fact that recursive calls have generated new abstract control points i' by inserting these intervals into the representation, and we then insert the new pair $\langle i, v'\rangle$ itself.

The insertion of $\langle i, v'\rangle$ can be done for instance in a fairly simple way by using the usual widening operator ∇_{I} over $\mathbf{I}(\mathbf{Z})$ defined in Section 2, and replacing $\langle i, v\rangle$ in the initial representation by $\langle i, v \nabla_{\mathrm{I}} v'\rangle$, in order to make sure that the increasing chain of abstract values $(v, v \nabla_{\mathrm{I}} v', \ldots)$ will be eventually stable. Of course, at the beginning of the iteration sequence, it is also safe to replace $\langle i, v\rangle$ by $\langle i, v \vee v'\rangle$.

The insertion of the new abstract control points i' into the representation is more subtle, and uses the elementary widening operator ∇ defined in Section 4.3. Obviously, it is generally unsafe to add directly $\langle i', \bot\rangle$ into the representation since, as discussed in Section 4.3, i' might 'mask' one of the intervals i in the domain of the representation and thus invalidate its meaning. The smallest pair that can be safely inserted is therefore $\langle i', \mathscr{S}_P(i')\rangle$. But abstract control points themselves need to be generalized to enforce a finite computation, and at some point of the iteration sequence, we will have to replace the interval i' by a greater one i'', e.g. the maximum element $\top$. This can be formalized by introducing three elementary widening operators defined as follows.

(∇_a) Add the pair $\langle i', \mathscr{S}_P(i')\rangle$. This is the most precise, join-like, widening operator.

(∇_b) When it is safe not to add i', i.e. when the region covered by i' is already covered by the domain of P, then do nothing, otherwise generalize by adding $\langle i'', \mathscr{S}_P(i'')\rangle$, where $i'' \geqslant i'$. A good choice can be for instance $i'' = (\bigvee A(P)) \vee i'$, i.e. the smallest interval representing all the values over which Φ has been computed so far, in which case $\mathscr{S}_P(i'') = \bot$.

(∇_c) Finally, to avoid adding an infinite number of abstract control points, one can use the widening operator over the intervals and add $\langle (\bigvee A(P))\, \nabla_{\mathrm{I}}\, i', \bot\rangle$.

Of course, the choice of the *sequence* of elementary widening operator is essential. The first elementary widening operator ∇_a will generally be used at the beginning of the computation, and ∇_c will systematically be used at the end. Moreover, it is often useful, after having generalized using ∇_c, to make a few more precise steps using ∇_a or ∇_b. The motivation behind this choice is that once the domain of the minimal function graph has been delimited, a few more precise steps are generally needed to determine the abstract control points that are useful to precisely describe this graph and allow these intervals to 'propagate' along recursive calls.

Finally, note that the insertion of the updated abstract values v' and the insertion of the new abstract control points i' can be freely mixed in practice, and newly generated control points can be added on the fly to the representation without problem.

The widening operator that we have described turns the functional partitioning framework into a *tractable* framework. So for instance, using the widening operator $(\nabla_c \nabla_a \nabla_c^{\omega})$, one can *automatically* compute, after four iterations, the following representation of the minimal function graph of ***Loop*** for the input data specification $\{[0,0]\}$:

$$\{\langle[0,0],[100,100]\rangle, \langle[0,\omega^+],[100,\omega^+]\rangle, \langle[1,1],[100,100]\rangle, \langle[1,100],[100,100]\rangle\}$$

which has the following meaning:

i	***Loop*** (i)
$0 \leqslant n \leqslant 100$	$[100,100]$
$100 < n$	$[100,\omega^+]$

This result is interesting in that it shows that the exact information: ***Loop*** $[0,100] = [100,100]$ has been obtained, as opposed to Section 2, and this has been achieved without the help of a narrowing operator. However, contrary to the result of Section 2, the approximate minimal function graph seems to indicate that the computation of ***Loop*** (0) might require computing ***Loop*** for values greater than 100, but starting this time from the input data specification $\{[0,100]\}$, and using the 'brute force' widening operator (∇_c^{ω}) we can compute the following representation:

$$\{\langle[0,100],[100,100]\rangle\}$$

which invalidates this interpretation. Similarly, using the widening operator $(\nabla_c \nabla_a^2 \nabla_c^{\omega})$ one can compute, after four iterations, the following representation of the minimal function graph of ***Mc*** for the input specification $\{[0,50]\}$:

$$\{\langle[0,50],[91,91]\rangle, \langle[0,\omega^+-10],[91,\omega^+]\rangle, \langle[11,111],[91,101]\rangle, \langle[11,61],[91,91]\rangle,$$
$$\langle[22,72],[91,91]\rangle, \langle[22,111],[91,101]\rangle, \langle[91,101],[91,91]\rangle\}.$$

This representation has the following meaning:

n	$\boldsymbol{Mc}(n)$
$n < 0$	$\bot$
$0 \leqslant n \leqslant 72$	$[91, 91]$
$73 \leqslant n \leqslant 90$	$[91, 101]$
$91 \leqslant n \leqslant 101$	$[91, 91]$
$102 \leqslant n \leqslant 111$	$[91, 101]$
$112 \leqslant n$	$[91, \omega^+ - 10]$

which is a good and safe approximation of the exact meaning of ***Mc***, i.e.:

$$\boldsymbol{Mc}(n) = \begin{cases} n - 10 & \textbf{if } n > 101 \\ 91 & \textbf{otherwise.} \end{cases}$$

It is interesting to compare this result to the one obtained in Bourdoncle (1990) using a method based on *static partitioning*. In this method, the representation of MacCarthy's 91 function would consist of three interval pairs, each pair being associated with a *syntactically different* call to ***Mc***, that is, the main call to ***Mc*** and the two recursive calls. This formally corresponds to having three mutually recursive functions ***Mc1***, ***Mc2*** and ***Mc3*** with the following definition:

```
if (n > 100) then
    n-10
else
    Mc3(Mc2(n+11))
```

and describing each of these functions by a pair of intervals representing all of the function's inputs and all of the function's outputs. The result obtained is the following:

$$\boldsymbol{Mc1}: \langle [0, 50], [91, \omega^+ - 20] \rangle$$
$$\boldsymbol{Mc2}: \langle [11, \omega^+], [91, \omega^+ - 10] \rangle$$
$$\boldsymbol{Mc3}: \langle [91, \omega^+ - 10], [91, \omega^+ - 20] \rangle.$$

This quite mediocre result can be explained by noting that the induction property:

$$\forall\, n \in [91, 101] : \boldsymbol{Mc}(n) = 91$$

has not been *inferred* by the framework because the number of interval pairs was fixed *in advance*. This phenomenon can be worked around by using an *ad-hoc* input data specification, namely $\{[0, 100]\}$, which gives the following, optimum, result:

$$\boldsymbol{Mc1}: \langle [0, 100], [91, 91] \rangle$$
$$\boldsymbol{Mc2}: \langle [11, 111], [91, 101] \rangle$$
$$\boldsymbol{Mc3}: \langle [91, 101], [91, 91] \rangle.$$

However, this 'trick' is not necessary when using the functional partitioning framework, since this framework infers the interesting program properties by itself, and automatically determines the number of interval pairs needed to describe the program invariant. However, it is worth mentioning that the widening operator has a major impact on the result's quality, and for instance, the 'brute force' widening operator would only compute, after two iterations, the following, mediocre but concise, representation:

$$\{\langle[0,50],[91,\omega^+-10]\rangle,\langle[0,\omega^+],[91,\omega^+-10]\rangle\}$$

with the obvious meaning:

$$\forall n \in [0,\omega^+]: \boldsymbol{Mc}(n) \in [91,\omega^+-10].$$

This example shows that the data-oriented approach of dynamic partitioning is much more versatile than the syntax-oriented approach of static partitioning, and generally gives better results. But on the other hand, static partitioning guarantees the size of the least fixed point's representation, and can lead to faster analyses. Finally, note that the two approaches can be easily mixed. For example, using the widening operator $(\nabla_c \nabla_a \nabla_c^{(\omega)})$ and the input data specification $\langle\{[0,50]\},\varnothing,\varnothing\rangle$, one can compute, after five iterations, the following representations:

$$\begin{aligned}
&\boldsymbol{Mc1}:\{\langle[0,50],[91,91]\rangle\}\\
&\boldsymbol{Mc2}:\{\langle[11,61],[91,91]\rangle,\langle[11,\omega^+],[91,\omega^+-10]\rangle,\langle[22,72],[91,91]\rangle,\\
&\qquad\qquad\qquad\qquad\qquad\langle[22,111],[91,101]\rangle\}\\
&\boldsymbol{Mc3}:\{\langle[91,101],[91,91]\rangle\}
\end{aligned}$$

which have the following, coalesced meaning, obtained by intersecting their individual meanings:

n	$\boldsymbol{Mc}(n)$
$n < 0$	$\perp$
$0 \leqslant n \leqslant 72$	$[91,91]$
$73 \leqslant n \leqslant 90$	$[91,101]$
$91 \leqslant n \leqslant 101$	$[91,91]$
$102 \leqslant n \leqslant 111$	$[91,101]$
$112 \leqslant n$	$[91,\omega^+-10]$

6 Conclusion

We have presented a technique that enables rich abstract interpretation frameworks to be built from simpler ones even in cases when one has no indication about what such frameworks should look like. We believe in particular that functional partitioning is of great interest to interprocedural abstract interpretation for it incrementally builds finite, non-trivial representations of minimal function graphs and monotonic functions. More generally, the representation framework can be used every time there is no canonical representation of abstract program properties and the equivalence test over these properties is intractable or very costly.

We have shown how widening operators can be built in dynamic partitioning frameworks, and exemplified their behaviour over a set of examples. However, this paper has not addressed a number of interesting problems such as the effective design of narrowing operators, the combination of forward and backward analyses, and the generalization of functional partitioning to higher order functions.

References

Bourdoncle, F. 1990. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In: *Proc. International Workshop PLILP'90*. Vol. 456 of *Lecture Notes in Computer Science*, Springer-Verlag.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction of approximations of fixpoints. In: *Proc. 4th ACM Symp. on POPL*, 238–252.

Cousot, P. and Halbwachs, N. 1978. Automatic discovery of linear constraints among variables of a program. In: *Proc. 5th ACM Symp. on POPL*, 84–97.

Cousot, P. 1978. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis. Analyse sémantique de programmes. PhD Thesis, Université Scientifique et Médicale de Grenoble.

Cousot, P. and Cousot, R. 1978. Static determination of dynamic properties of recursive procedures. In: *Formal Description of Programming Concepts*. North-Holland, 237–277.

Cousot, P. 1981. Semantic foundations of program analysis. In: Muchnick and Jones (eds), *Program Flow Analysis, Theory and Applications*. Prentice-Hall, 303–343.

Gierz, G., Hofmann, K. H., Keimel, K., Lawson, J. D., Mislove, M., Scott, D. S. 1980. *A Compendium of Continuous Lattices*. Springer-Verlag.

Gunter, C. A. and Scott, D. S. 1990. Semantic domains. In: *Handbook of Theoretical Computer Science* (Chapter 12). Elsevier, 635–673.

Jones, N. D. and Muchnick, S. 1982. A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In: *Proc. 9th ACM Symp. on POPL*.

Jones, N. D. and Mycroft, A. 1986. Data flow analysis of applicative programs using minimal function graphs. In: *Proc. 13th ACM Symp. on POPL*, 296–306.

Mycroft, A. and Nielson, F. 1983. Strong abstract interpretation using power domains (extended abstract). In: *Proc. 10th ICALP*. Vol. 154 of *Lectures Notes in Computer Science*. Springer-Verlag, 536–547.

Stransky, J. 1990. A lattice for abstract interpretation of dynamic (lisp-like) structures. L.I.X. internal report LIX/RR/90/03.

Sharir, M. and Pnueli, A. 1981. Two approaches to interprocedural data flow analysis. In: Muchnick and Jones (eds), *Program Flow Analysis, Theory and Applications*. Prentice-Hall, 189–233.

Schmidt, D. A. 1953. *Denotational Semantics*. Allyn and Bacon, Inc.

https://doi.org/10.1017/S0956796800000496 Published online by Cambridge University Press