# Introduction to Abstract Interpretation *

**Patrick Cousot**
LIENS, École Normale Supérieure
45, rue d'Ulm
75230 Paris cedex 05 (France)
cousot@dmi.ens.fr

**Radhia Cousot**
LIX, École Polytechnique
91128 Palaiseau cedex (France)
radhia@polytechnique.fr

March 19, 1998

Ces notes pour le cours « Interprétation abstraite » de l'édition 1998 de l' « École Chercheurs en Programmation » (École des Mines de Nantes, 23 mars – 3 avril 1998), sont extraites de :

P. Cousot & R. Cousot.
Abstract interpretation and application to logic programs.
*J. Logic Prog.*, 13(2–3):103–179, 1992.

**Abstract.** Abstract interpretation is a theory of semantics approximation which is used for the construction of semantics-based program analysis algorithms (sometimes called "data flow analysis"), the comparison of formal semantics (e.g., construction of a denotational semantics from an operational one), the design of proof methods, etc.

Automatic program analysers are used for determining statically conservative approximations of dynamic properties of programs. Such properties of the run-time behavior of programs are useful for debugging (e.g., type inference), code optimization (e.g., compile-time garbage collection, useless occur-check elimination), program transformation (e.g., partial evaluation, parallelization), and even program correctness proofs (e.g., termination proof).

After a few simple introductory examples, we recall the classical framework for abstract interpretation of programs. Starting from a standard operational semantics formalized as a transition system, classes of program properties are first encapsulated in collecting semantics expressed as fixpoints on partial orders representing concrete program properties. We consider invariance properties characterizing the descendant states of the initial states (corresponding to top/down or forward analyses), the ascendant states of the final states (corresponding to bottom/up or backward analyses) as well as a combination of the two. Then we choose specific approximate abstract properties to be gathered about program behaviors and express them as elements of a poset of abstract properties. The correspondence between concrete and abstract properties is established by a concretization and abstraction function that is a Galois connection formalizing the loss of information. We can then constructively derive the abstract program properties from the collecting semantics by a formal computation leading to a fixpoint expression in terms of abstract operators on the domain of abstract properties. The design of the abstract interpreter then involves the choice of a chaotic iteration strategy to solve this abstract fixpoint equation. We insist on the compositional design of this abstract interpreter, which is formalized by a series of propositions for designing Galois connections (such as Moore families, decomposition by partitioning, reduced product, down-set completion, etc.). Then we recall the convergence acceleration methods using widening and narrowing allowing for the use of very expressive infinite domains of abstract properties.

## 1. INTRODUCTION

Program manipulators (such as programmers who write, debug, and attempt to understand programs or computer programs which interpret, compile, or execute programs) reason upon or

* This work was supported in part by Esprit BRA 3124 *Sémantique* and the CNRS GDR $C^3$.

are constructed by relying on the syntax but mainly on the semantics of these programs. The *semantics* of a program describes the set of all possible behaviors of that program when executed for all possible input data. For logic programs, the input data are questions. The behaviors can be non-termination, termination with a run-time error, failure, or correct termination delivering one or more output answers.

For a given type of reasoning about programs, not all aspects and details about their possible behaviors during execution have to be considered. Each program manipulation is facilitated by reasoning upon a well-adapted semantics, abstracting away from irrelevant matters. For example, logical programs debugging often refers to a small-step operational semantics with backtracking. On the contrary, programs explanation often refers to the declarative aspect of a logic program providing the relation between questions and answers. Therefore, there is no universal general-purpose semantics of programs, and, in everyday life, more or less formal, more or less precise, special-purpose semantics are in current use. *Abstract interpretation* is a method for relating these semantics.

We will explain the abstract interpretation framework that we introduced in [6], [7], [9], [10], [13], [15] and illustrate it for logic programs. Thanks to examples, we will consider two essential utilizations of abstract interpretation: (a) The first utilization is the design of an abstract semantics in order to show off an underlying structure in a concrete, more detailed semantics. Hence, properties of programs are induced, without loss of indispensable information, from a concrete into a more abstract setting. A typical example consists in designing a proof method starting from a collecting semantics [8]. (b) The second utilization of abstract interpretation is the design of an abstract semantics in order to specify an automatic program analyser for the static determination of dynamic properties of programs. Here, properties of programs are approximated, with an inevitable loss of information, from a concrete to a less precise abstract setting. Such semantics-based sound but approximate information is indispensable to identify errors in a program, as performed by program debuggers and type checkers. Another use is in program transformers such as compilers, partial evaluators, and parallelizers, where the analysis determines the applicability of various transformations.

## 2. SIMPLE EXAMPLES OF ABSTRACT INTERPRETATION

As a first approximation, abstract interpretation can be understood as a nonstandard semantics, i.e., one in which the domain of values is replaced by a domain of descriptions of values, and in which the operators are given a corresponding nonstandard interpretation.

### *2.1. Rule of Signs*

For example, rather than using integers as concrete values, an abstract interpretation may use abstract values $-1$ and $+1$ to describe negative and positive integers, respectively [50]. Then by reinterpreting operations like addition or multiplication according to the "rule of signs" due to the ancient Greek mathematicians, the abstract interpretation may establish certain properties of a program such as "whenever this loop body is entered, variable $x$ is assigned a positive value (or perhaps is uninitialized)."

#### *2.1.1. The Rule of Signs Calculus*

For example, $(x \times x) + (y \times y)$ yields the value 25 when $x$ is 3 and $y$ is $-4$ and when $\times$ and $+$ are the usual arithmetical multiplication and addition. But when applying the "rule of signs":

$$\begin{array}{lll} +1 + +1 = +1 & \quad +1 \times +1 = +1 & \quad -1 \times +1 = -1 \\ -1 + -1 = -1 & \quad +1 \times -1 = -1 & \quad -1 \times -1 = +1 \end{array}$$

(where the abstract value $+1$ represents any positive integer, while $-1$ represents any negative integer) one concludes that the sign of $(3 \times 3) + (-4 \times -4)$ is always $+1$ since $(+1 \times +1) + (-1 \times -1) = (+1) + (+1) = +1$. However, this simple abstract calculus fails to prove that $x^2 + 2 \times x \times y + y^2$ is always positive.

Although very simple, this example shows that abstract interpretations may fail. To avoid errors due to such failures in a partial abstract calculus, we choose to use a total abstract calculus where an abstract value $\top$ is introduced to represent the fact that nothing is known about the result:

$$\begin{array}{cccc} +1 + -1 = \top & & & \\ -1 + +1 = \top & +1 + \top = \top & \top \times +1 = \top & -1 \times \top = \top \\ \top + +1 = \top & -1 + \top = \top & \top \times -1 = \top & \top \times \top = \top \\ \top + -1 = \top & \top + \top = \top & +1 \times \top = \top & \end{array}$$

Now, several abstract values can be used to approximate a given concrete value. For example, the concrete value 5 can be approximated by $+1$ or $\top$. A partial order relation $\preceq$ can be introduced to compare the precision of abstract values ([58], [38]). For example, $-1 \preceq \top$ and $+1 \preceq \top$ since $-1$ or $+1$ are more precise than $\top$, whereas $-1$ and $+1$ are not comparable since no one can always safely replace the other.

A concrete value may be approximated by several minimal values. For example, 0 can be approximated by minimal abstract values $-1$ or $+1$. In this case, the best choice may depend upon the expression to be analysed. For example, when analysing $0+x$ it is better to approximate 0 by $+1$ if $x$ is known to be positive and by $-1$ when $x$ is negative. In order to avoid having to do the choice during the abstract calculus or to explore all alternatives, it is always possible to enrich the abstract domain so that the set of upper approximations of any given concrete value has a best element [15]. For our example, this leads to the introduction of an abstract value 0:

$$\begin{array}{cccc} 0 + +1 = +1 & & 0 \times +1 = 0 & \\ 0 + -1 = -1 & +1 + 0 = +1 & 0 \times -1 = 0 & +1 \times 0 = 0 \\ 0 + \top = \top & -1 + 0 = -1 & 0 \times \top = 0 & -1 \times 0 = 0 \\ 0 + 0 = 0 & \top + 0 = \top & 0 \times 0 = 0 & \top \times 0 = 0 \end{array}$$

### 2.1.2. *Generalization to Interval Analysis*

In [9], this "rule of signs" idea was generalized to interval analysis, i.e., to properties of the form $l \leq x \leq u$ where $l, u \in \mathbb{Z} \cup \{-\infty, +\infty\}$, $\mathbb{Z}$ is the set of integers, and $l \leq u$. The main innovations were the idea of soundness proof by relating the abstract interpretations to an operational semantics and the use of infinite abstract domains, which led to very powerful analyses, as shown by the following results (where the comments have been generated automatically [1]):

```
function F(X : integer) : integer;
begin
  if X > 100 then begin
    F := X − 10
    { X ∈ [101, maxint] ∧ F ∈ [91, maxint - 10] }
  end else begin
    F := F(F(X + 11))
    { X ∈ [minint, 100] ∧ F = 91 }
  end;
end;
```

This analysis supersedes the most sophisticated methods based upon data flow analysis. Let us consider the following program given in [30]:

```
program AnOldLookAtOptimizingArrayBoundChecking;
  var
    i, j, k, l, m : integer;
    a : array[1..100] of real;
begin
```

```
  read(i, j);
  { i∈★[1, 99]★ ∧ j∈[-maxint-1, maxint] }
  k := i;
  { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 99] }
  l := 1;
  { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 99] ∧ l∈[1, 1]}
  while l <= i do begin
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] }
    m := i;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] ∧ m∈[1, 99] }
    while m <= j do begin
      { i∈[1, 99] ∧ j∈[1, maxint] ∧ k∈[1, maxint-1] ∧ l∈[1, 99] ∧ m∈[1, maxint-1] }
      k := k + m;
      { i∈[1, 99] ∧ j∈[1, maxint] ∧ k∈[2, maxint]# ∧ l∈[1, 99] ∧ m∈[1, maxint-1] }
      m := m + 1;
      { i∈[1, 99] ∧ j∈[1, maxint] ∧ k∈[2, maxint] ∧ l∈[1, 99] ∧ m∈[2, maxint] }
    end;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, maxint] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    a[l] := k;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, maxint] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    if k < 1000 then begin
      { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
      write(k);
      { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    end else begin
      { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1000, maxint] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
      k := i;
      { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 99] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    end;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    a[l + 1] := a[l] / 2;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[1, 99] ∧ m∈[1, maxint] }
    l := l + 1;
    { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[2, 100] ∧ m∈[1, maxint] }
  end;
  { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[2, 100] ∧ m∈[1, maxint] }
  write(a[i]);
  { i∈[1, 99] ∧ j∈[-maxint-1, maxint] ∧ k∈[1, 999] ∧ l∈[2, 100] ∧ m∈[1, maxint] }
end.
```

The invariants given in comments have been discovered automatically. They hold during any execution of the program without run-time error. If any one of these invariants is violated during execution, then a later run-time error is inevitable. To detect these run-time errors before they occur, it is shown automatically that only two bound checks (marked $\star$) are necessary upon initialization as well as an overflow check (marked $\#$) within the loop. This analysis seems well out of reach of the data flow analysis of [30] based upon the syntactical elimination, propagation, and combination of range checks.

### *2.2. Dimension Calculus*

Let us now consider a familiar example from elementary physics.

#### *2.2.1. The Dimension Calculus*

The dimension calculus uses the abstract values *length*, *surface*, *volume*, *time*, *speed*, *acceleration*, *mass*, *force*, ... , *nodimension*. The abstract version $\bar{op}$ of an operator $op$ is defined as follows:

$$\begin{array}{rcl} length \mathbin{\bar{+}} length &=& length \\ length \mathbin{\bar{\times}} length &=& surface \\ length \mathbin{\bar{/}} length &=& nodimension \\ length \mathbin{\bar{/}} time &=& speed \\ speed \mathbin{\bar{/}} time &=& acceleration \end{array} \qquad \begin{array}{rcl} mass \mathbin{\bar{\times}} acceleration &=& force \\ & \ldots & \\ x \mathbin{\bar{/}} y^{\overline{n+1}} &=& \bar{(}x \bar{/} y^{\bar{n}}\bar{)} \bar{/} y \\ y^{\bar{1}} &=& y \\ \bar{(}x\bar{)} &=& x \\ & \ldots & \end{array}$$

The correspondence between concrete values and abstract values can be formalized by an abstraction function $\alpha$ mapping units to dimensions:

$$\begin{array}{rcl} \alpha(meter) &=& length \\ \alpha(mile) &=& length \\ \alpha(acre) &=& surface \\ \alpha(second) &=& time \\ \alpha(minute) &=& time \\ \alpha(hour) &=& time \\ \alpha(kilogram) &=& mass \end{array} \qquad \begin{array}{rcl} \alpha(pound) &=& mass \\ \alpha(ton) &=& mass \\ \alpha(Newton) &=& force \\ \alpha(nounit) &=& nodimension \\ & \ldots & \\ \alpha(E_1 \ op \ E_2) &=& \alpha(E_1) \ \bar{op} \ \alpha(E_2) \\ \alpha((E)) &=& \bar{(}\alpha(E)\bar{)} \end{array}$$

The abstract interpretation of an expression can be done in two distinct steps: it begins with the derivation of an abstract expression from the concrete expression and goes on with the evaluation of the abstract expression using the definition of the abstract operators. In our example, the abstract expression is first obtained using the abstraction operator $\alpha$:

$$\begin{array}{rcl} \alpha(kg \times (m/s^2)) &=& \alpha(kg) \mathbin{\bar{\times}} \alpha((m/s^2)) \\ &=& mass \mathbin{\bar{\times}} \bar{(}\alpha(m \,/\, s^2)\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}\alpha(m) \mathbin{\bar{/}} \alpha(s^2)\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}length \mathbin{\bar{/}} \alpha(s)^{\bar{2}}\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}length \mathbin{\bar{/}} time^{\bar{2}}\bar{)} \end{array}$$

Since, in general, the abstraction function $\alpha$ is not computable, this first phase, which is usually done by hand, can be understood as the design of an abstract compiler. Then, the abstract expression can be evaluated using an abstract interpreter:

$$\begin{array}{rcl} mass \mathbin{\bar{\times}} \bar{(}length \mathbin{\bar{/}} time^{\bar{2}}\bar{)} &=& mass \mathbin{\bar{\times}} \bar{(}\bar{(}length \mathbin{\bar{/}} time\bar{)} \mathbin{\bar{/}} time\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}\bar{(}speed\bar{)} \mathbin{\bar{/}} time\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}speed \mathbin{\bar{/}} time\bar{)} \\ &=& mass \mathbin{\bar{\times}} \bar{(}acceleration\bar{)} \\ &=& mass \mathbin{\bar{\times}} acceleration \\ &=& force \end{array}$$

This second phase of abstract execution must always be finitely computable, hence must only involve the finite iterated application of computable abstract operations on finitely representable abstract values.

The main interest of this example is to illustrate the idea of proving the correctness of the abstract interpretation relatively to a semantics via an abstraction operator as introduced in [9] and [10]. The importance of this idea was that by relating abstract interpretations not to programming languages but to their operational semantics, one was able to define abstract interpretation independently of any programming language, thus obtaining a theory applicable to all programming languages. This can also be understood as meaning that abstract interpretations designed for a language can systematically be transferred to any other language. Moreover, by making clear the relationships between analysis and semantics [15], independently of any program property, a theory of discrete approximation emerged, which has a very broad scope since it is applicable from the design of semantics to that of low-level data flow analyses.

### 2.2.2. *Generalization to Type Checking and Type Inference*

Computer scientists would understand the dimension calculus as a type checking ensuring the correct use of units of measure. The idea of using a calculus for type-checking programs is due to Naur ([43], [44]) in the GIER ALGOL III compiler: "The basic method is a pseudo-evaluation of the expressions of the program. This proceeds like a run-time evaluation as far as the combining of operators and operands is concerned, but works with descriptions of the types and kinds of the operand instead of with values." Pseudo-evaluation is an abstract interpretation where the abstract operators are:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\text{integer} + \text{integer} &=& \text{integer} & \text{integer} \leq \text{integer} &=& \text{Boolean} \\
\text{integer} + \text{real} &=& \text{real} & \text{integer} \leq \text{real} &=& \text{Boolean} \\
\text{real} + \text{integer} &=& \text{real} & \text{real} \leq \text{integer} &=& \text{Boolean} \\
\text{real} + \text{real} &=& \text{real} & \text{real} \leq \text{real} &=& \text{Boolean}
\end{array}
$$

Errors were handled using an "error" ("undeclared" in [44]) abstract value. An error message was produced when it appeared for the first time in the abstract interpretation of an expression. Thereafter, "error" was accepted as abstract operand in order to prevent redundant error messages:

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
\text{integer} + \text{Boolean} &=& \text{error} & \text{error} + \text{integer} &=& \text{error} \\
\text{Boolean} + \text{integer} &=& \text{error} & \text{real} + \text{error} &=& \text{error} \\
\text{Boolean} + \text{Boolean} &=& \text{error} & \text{error} + \text{real} &=& \text{error} \\
\text{real} + \text{Boolean} &=& \text{error} & \text{Boolean} + \text{error} &=& \text{error} \\
\text{Boolean} + \text{real} &=& \text{error} & \text{error} + \text{Boolean} &=& \text{error} \\
\text{integer} + \text{error} &=& \text{error} & \text{error} + \text{error} &=& \text{error}
\end{array}
$$

In total, 25 abstract values were used, in fact much more since the number of dimensions of arrays and the number of parameters (not their type) of procedures and functions was taken into account.

## 2.3. *Casting Out of Nine*

Our last introductory example is well known by French pupils who use casting out of nine to check their additions and multiplications. To check the correctness of the multiplication $217 \times 38 = 8256$, one computes the respective rests $r_1 = (2 + 1 + 7) \bmod 9 = 1$, $r_2 = (3 + 8) \bmod 9 = 2$ and $r = (8 + 2 + 5 + 6) \bmod 9 = 3$ of the division by 9 of the sum of the digits of the first factor 217, of the second factor 38 and of the result 8256. Then one computes the rest $p = (r_1 \times r_2) \bmod 9 = (1 \times 2) \bmod 9 = 2$ of the division by 9 of the product $r_1 \times r_2$ of the rests. The disposition of the calculation on paper is shown in Figure 1. If $r \neq p$, then one concludes that the multiplication was done incorrectly. This is the case in our example. Whenever $r = p$, one cannot conclude that the operation is correct (although most pupils get very confident in their result; the unfortunate French name of "proof by nine" certainly enforcing this undue conviction).


**FIGURE 1.** Casting out of nine calculation.

### *2.3.1. The Casting Out of Nine Calculus*

Since casting out of nine is a rather simple abstract interpretation, we will design it formally so as to justify the above rule. To do this, we follow the systematic approach introduced in [6], [7], [10], and [15].

2.3.1.1. Syntax of Expressions. The syntax of expressions is given by the following grammar where $E$ is an expression, $P$ a product, $N$ a number, and $D$ a digit:

$$\begin{array}{lcl} E & ::= & P = N \\ P & ::= & N_1 \times N_2 \\ N & ::= & D \mid ND \\ D & ::= & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array}$$

2.3.1.2. Operational Semantics of Expressions. The operational semantics of expression $E$ is a boolean $\mathcal{E}[\![E]\!] \in \{\mathit{true}, \mathit{false}\}$, defined as follows:

$$\begin{array}{ccccc} \mathcal{E}[\![P = N]\!] & = & \mathit{true} & \text{if } \mathcal{E}[\![P]\!] \;=\; \mathcal{E}[\![N]\!] \\ \mathcal{E}[\![P = N]\!] & = & \mathit{false} & \text{if } \mathcal{E}[\![P]\!] \;\neq\; \mathcal{E}[\![N]\!] \\ \mathcal{E}[\![N_1 \times N_2]\!] & = & \mathcal{E}[\![N_1]\!] \;\times\; \mathcal{E}[\![N_2]\!] & \\ \mathcal{E}[\![ND]\!] & = & (10 \times \mathcal{E}[\![N]\!]) + \mathcal{E}[\![D]\!] & \\ \mathcal{E}[\![0]\!] & = & 0 & \\ \dots & & & \\ \mathcal{E}[\![9]\!] & = & 9 & \end{array}$$

2.3.1.3. Abstraction by Casting Out of Nine. The approximation consists in computing modulo nine ($[x]_9$ denotes the remainder upon division by 9 of integer $x \in \mathbb{Z}$):

$$\begin{array}{lcll} \alpha(X) & = & [\mathcal{E}[\![X]\!]]_9 & \text{if } X \text{ is } P,\ N, \text{ or } D \\ \alpha(P = N) & = & \mathit{error} & \text{if } [\mathcal{E}[\![P]\!]]_9 \;\neq\; [\mathcal{E}[\![N]\!]]_9 \\ \alpha(P = N) & = & \mathit{unknown} & \text{if } [\mathcal{E}[\![P]\!]]_9 \;=\; [\mathcal{E}[\![N]\!]]_9 \end{array}$$

The intuition behind this formal definition is that $[x]_9 \neq [y]_9$ implies $x \neq y$ so that whenever the abstract value *error* is found, the multiplication is incorrect.

2.3.1.4. Systematic Design of the Abstract Interpreter. The design of the abstract interpreter consists in expressing $\alpha(E)$ in an equivalent form involving only arithmetic modulo 9, i.e., operations on the abstract values *unknown*, *error*, 0, 1, ... , 8. Such abstract operations are effective since they involve a finite domain. We proceed by induction on the syntax of expressions. For the basis, we have:

$$\begin{array}{rcll} \alpha(0) & = & [\mathcal{E}[\![0]\!]]_9 & \text{by definition of } \alpha; \\ & = & [0]_9 & \text{by definition of } \mathcal{E}; \\ & = & 0 & \text{by definition of remainders.} \\ \dots & = & \dots & \dots \\ \alpha(9) & = & [\mathcal{E}[\![9]\!]]_9 & \text{by definition of } \alpha; \\ & = & [9]_9 & \text{by definition of } \mathcal{E}; \\ & = & 0 & \text{by definition of remainders.} \end{array}$$

Now, for the induction hypothesis, we assume that we have already expressed $\alpha(t_i)$ by composition of operators on abstract values for subterms $t_i,\ i \in [1, n]$. To do the same for term $f(t_1, \dots, t_n)$, we look for an abstract operator $\bar{f}$ such that we can prove $\alpha(f(t_1, \dots, t_n)) = \bar{f}(\alpha(t_1), \dots, \alpha(t_n))$ and insist upon the fact that $\bar{f}$ should be effectively computable using only abstract values:

$$
\begin{array}{lll}
\alpha(ND) & & \\
\quad = & [\mathcal{E}[\![ND]\!]]_9 & \text{by definition of } \alpha; \\
\quad = & [(10 \times \mathcal{E}[\![N]\!]) + \mathcal{E}[\![D]\!]]_9 & \text{by definition of } \mathcal{E}; \\
\quad = & [[(10 \times \mathcal{E}[\![N]\!])]_9 + [\mathcal{E}[\![D]\!]]_9]_9 & \text{since } [x+y]_9 = [[x]_9 + [y]_9]_9; \\
\quad = & [[[10]_9 \times [\mathcal{E}[\![N]\!]]_9]_9 + [\mathcal{E}[\![D]\!]]_9]_9 & \text{since } [x \times y]_9 = [[x]_9 \times [y]_9]_9; \\
\quad = & [[1 \times [\mathcal{E}[\![N]\!]]_9]_9 + \alpha(D)]_9 & \text{since } [10]_9 = 1 \text{ and by definition of } \alpha; \\
\quad = & [[[\mathcal{E}[\![N]\!]]_9]_9 + \alpha(D)]_9 & \text{since } 1 \times x = x; \\
\quad = & [[\mathcal{E}[\![N]\!]]_9 + \alpha(D)]_9 & \text{since } [[x]_9]_9 = [x]_9; \\
\quad = & [\alpha(N) + \alpha(D)]_9 & \text{by definition of } \alpha; \\
\quad = & (\alpha(N) \mathbin{\bar{+}} \alpha(D)) & \text{by letting } x \mathbin{\bar{+}} y \stackrel{\text{def}}{=} [x+y]_9.
\end{array}
$$

$$
\begin{array}{lll}
\alpha(N_1 \times N_2) & & \\
\quad = & [\mathcal{E}[\![N_1 \times N_2]\!]]_9 & \text{by definition of } \alpha; \\
\quad = & [\mathcal{E}[\![N_1]\!] \ \times \ \mathcal{E}[\![N_2]\!]]_9 & \text{by definition of } \mathcal{E}; \\
\quad = & [[\mathcal{E}[\![N_1]\!]]_9 \times [\mathcal{E}[\![N_2]\!]]_9]_9 & \text{since } [x \times y]_9 = [[x]_9 \times [y]_9]_9; \\
\quad = & [\alpha(N_1) \times \alpha(N_2)]_9 & \text{by definition of } \alpha; \\
\quad = & (\alpha(N_1) \mathbin{\bar{\times}} \alpha(N_2)) & \text{by letting } x \mathbin{\bar{\times}} y \stackrel{\text{def}}{=} [x \times y]_9.
\end{array}
$$

$$
\begin{array}{lll}
\alpha(P = N) & & \\
\quad = & \mathit{error} & \text{if } [\mathcal{E}[\![P]\!]]_9 \ \neq \ [\mathcal{E}[\![N]\!]]_9, \\
\quad = & \mathit{unknown} & \text{if } [\mathcal{E}[\![P]\!]]_9 \ = \ [\mathcal{E}[\![N]\!]]_9;
\end{array}
$$

whence, by definition of $\alpha$,

$$
\begin{array}{lll}
\quad = & \mathit{error} & \text{if } \alpha(P) \ \neq \ \alpha(N), \\
\quad = & \mathit{unknown} & \text{if } \alpha(P) \ = \ \alpha(N);
\end{array}
$$

and letting $x \mathrel{\bar{=}} y \stackrel{\text{def}}{=}$ if $x = y$ then *unknown* else *error*,

$$\quad = \quad \alpha(P) \mathrel{\bar{=}} \alpha(N).$$

2.3.1.5. Abstract Interpretation by Casting Out of Nine. The above design leads to an automatic semantic analyser that consists of a compiler and an interpreter, organized as follows:

1. The abstract compiler reads an expression $E$ and produces (a computer representation of) an abstract expression $\mathcal{C}[\![E]\!]$ defined as follows:

$$
\begin{array}{ccc}
\mathcal{C}[\![P = N]\!] & = & (\mathcal{C}[\![P]\!] \mathrel{\bar{=}} \mathcal{C}[\![N]\!]) \\
\mathcal{C}[\![N_1 \times N_2]\!] & = & (\mathcal{C}[\![N_1]\!] \mathbin{\bar{\times}} \mathcal{C}[\![N_2]\!]) \\
\mathcal{C}[\![ND]\!] & = & (\mathcal{C}[\![N]\!] \mathbin{\bar{+}} \mathcal{C}[\![D]\!]) \\
\mathcal{C}[\![0]\!] & = & 0 \\
\ldots & & \\
\mathcal{C}[\![8]\!] & = & 8 \\
\mathcal{C}[\![9]\!] & = & 0
\end{array}
$$

2. An abstract interpreter $\mathcal{I}$ is written to evaluate abstract expressions, as follows:

$$
\begin{array}{cccl}
\mathcal{I}[\![(v_1 \mathrel{\bar{=}} v_2)]\!] & = & \mathit{unknown} & \text{if } \mathcal{I}[\![v_1]\!] = \mathcal{I}[\![v_2]\!] \\
 & = & \mathit{error} & \text{if } \mathcal{I}[\![v_1]\!] \neq \mathcal{I}[\![v_2]\!] \\
\mathcal{I}[\![(v_1 \mathbin{\bar{\times}} v_2)]\!] & = & [\mathcal{I}[\![v_1]\!] \times \mathcal{I}[\![v_2]\!]]_9 & \\
\mathcal{I}[\![(v_1 \mathbin{\bar{+}} v_2)]\!] & = & [\mathcal{I}[\![v_1]\!] + \mathcal{I}[\![v_2]\!]]_9 & \\
\mathcal{I}[\![0]\!] & = & 0 & \\
\ldots & & & \\
\mathcal{I}[\![8]\!] & = & 8 &
\end{array}
$$

The correctness of our semantic analyser follows from its design since we have:

$$\alpha(E) = \mathcal{I}[\![\mathcal{C}[\![E]\!]]\!]$$

For example, the abstract interpretation of the concrete expression $E = 217 \times 38 = 8256$ first consists in compiling into:

$$\bar{E} \;=\; \mathcal{C}[\![E]\!] \;= ((((2 \mathbin{\bar{+}} 1) \mathbin{\bar{+}} 7) \mathbin{\bar{\times}} (3 \mathbin{\bar{+}} 8)) \mathbin{\bar{=}} (((8 \mathbin{\bar{+}} 2) \mathbin{\bar{+}} 5) \mathbin{\bar{+}} 6))$$

Then evaluation of the abstract expression $\bar{E}$ results into:

$$\begin{aligned}
\mathcal{I}[\![\bar{E}]\!] &= [[[2+1]_9+7]_9 \times [3+8]_9]_9 \mathbin{\bar{=}} [[[8+2]_9+5]_9+6]_9 \\
&= [1 \times 2]_9 \mathbin{\bar{=}} 3 \\
&= 2 \mathbin{\bar{=}} 3 \\
&= \mathit{error}
\end{aligned}$$

thus proving that the equality does not hold.

*2.3.2. Generalization to Congruence Analysis*

Abstract interpretations of integers modulo some given integer can be applied to the analysis of programs, such as the parity analysis considered in [15]. They have been generalized to the automatic discovery of invariants that are conjunctions of arithmetical congruences of the form $\alpha x \equiv \beta \pmod{\gamma}$ where $\alpha$, $\beta$, and $\gamma$ are integer constants automatically discovered during the analysis and $x$ denotes the value of an integer variable of the program [28] and then to the discovery of linear congruences of the form $\alpha_1 x_1 + \cdots + \alpha_n x_n \equiv \beta \pmod{\gamma}$ where $\alpha_1, \ldots, \alpha_n, \beta$, and $\gamma$ are integer constants automatically discovered during the analysis and $x_1, \ldots, x_n$ denote the values of integer variables of the program [29]. For example, this last analysis, automatically discovers the invariant given after the loop of the program below, which computes the integer root $x$ of $n \geq 0$:

```
x := 0; y := 1; z := 1;
while y <= n do begin
  x := x + 1; z := z + 2; y := y + z;
end;
```
$\{\ 2x - z + 1 \equiv 0 \pmod 0 \ \wedge\ x + y \equiv 1 \pmod 2\ \}$

## 3. PRINCIPLES OF ABSTRACT INTERPRETATION

The abstract interpretation framework that we introduced, illustrated and explained in a series of papers [9], [10], [12], [13], [11], [6], [21], [15], [16] and [7] was motivated by the desire to justify the specification of program analysers with respect to some formal semantics. The guiding idea is that this process is a discrete inducing or approximation of properties from the exact or concrete semantics onto an approximate or abstract semantics that explicitly exhibits an underlying particular structure implicitly present in the richer concrete structure associated to program executions. Hence, abstract interpretation has a constructive aspect, as opposed to a mere a posteriori justification, in that the abstract semantics can be derived systematically from the concrete one, with the hope that this process will be ultimately computer-aided. We think here, for example, to the partly automatic generation of program analysers. Therefore, the subject of abstract interpretation involves the study of program semantics, of program proof methods, and of program analyser's specification, realization, and experimentation with the underlying idea that these different descriptions, views, facets, or abstractions of run-time behaviors of programs are all linked together by an inducing or approximation, i.e., abstraction process. Clearly, this involves the deep understanding and creation of mathematical structures to describe program executions and the study of their relationships, which is a vast subject mainly remaining to be explored when considering, for a provocative example, what is known in algebra about numbers and the simplicity of this structure when compared to that of computer programs.

The classical framework summarized in [7] starts from an operational semantics describing, for example, small program execution steps using a transition system (for example, flowcharts in [10]) or execution traces (example 7.2.0.6 in [15]). Then a static or collecting semantics, often described using fixpoints on ordered structures, is designed that is minimal, sound, and relatively complete for the program properties of interest. Intuitively, the collecting semantics is the most precise of the semantics that can be conceived to describe a certain class of so-called concrete program properties without referring to other program properties out of the scope of interest. It can be used for example to design proof methods [17], [22], [18]. The design of program analysers is based on abstract semantics that are approximations of the collecting semantics. There, the main concern is the compromise to be found between the difficulty of the analysis conception, the flexibility, the precision, and the cost of the analyses. Everything is fixed by the choice of the abstract properties to be considered (which can be governed, for example, by computer representation considerations) and by their semantics that is their correspondence with concrete properties. The use of Galois connections to express this correspondence squares with an ideal situation where there is a best way to approximate any concrete property by an abstract one. These two interrelated choices entirely determine the abstract semantics, which can be derived from the concrete collecting semantics and described using fixpoints. Then, the practical problem of effectively computing these fixpoints must be grappled with. There, chaotic and asynchronous methods are useful. Convergence can be accelerated using widening and narrowing operators so as to cope with infinite domains of abstract properties or to avoid combinatorial explosions. Hence, the approximation process is split up in the static design of an abstract semantics expressed as an equation and the iterative resolution of this equation. Independent designs also have to be combined.

We now enter into more details of this approach, which we illustrate using logic programs.

## 4. APPROXIMATION METHODS FOR ABSTRACT INTERPRETATION

We start with a few, hopefully well-known, mathematical facts.

### *4.1. Lattice and Fixpoint Theory*

Let be given sets $S$, $T$ and $U$. The *powerset* $\wp(S)$ is the set $\{X \mid X \subseteq S\}$ of all subsets of $S$. The *cartesian product* $S \times T$ is the set $\{\langle s, t\rangle \mid s \in S \wedge t \in T\}$ of all pairs with first component in $S$ and second component in $T$. A *binary relation* on $S \times T$ is a subset $\rho \in \wp(S \times T)$ of $S \times T$. A *pre-order* on a set $S$ is a binary relation $\sqsubseteq$ that is *reflexive* ($\forall x \in S : x \sqsubseteq x$, where $x \sqsubseteq x'$ stands for $\langle x, x'\rangle \in \sqsubseteq$) and *transitive* ($\forall x, y, z \in X : (x \sqsubseteq y \ \wedge\ y \sqsubseteq z) \Rightarrow x \sqsubseteq z$). We write $x \sqsubset y$ for $(x \sqsubseteq y \ \wedge\ x \neq y)$. A *partial order* on a set $S$ is a pre-order that is *antisymmetric* ($\forall x, y \in S : (x \sqsubseteq y \ \wedge\ y \sqsubseteq x) \Rightarrow x = y$).

Let $\sqsubseteq$ be a partial order on a set $S$. $u$ is an *upper bound* of a subset $X$ of $S$ if and only if $u$ is greater than or equal to all members of $X$ ($\forall x \in X : x \sqsubseteq u$). The *least upper bound* $u$ of a subset $X$ of $S$ is an upper bound of $X$ that is smaller than any other upper bound of $X$ ($\forall x \in X : x \sqsubseteq u \ \wedge\ \forall u' \in S : (\forall x \in X : x \sqsubseteq u') \Rightarrow (u \sqsubseteq u')$). A least upper bound is unique. If it exists, the least upper bound of $X$ is written $\sqcup X$. The *lower bounds* and *greatest lower bound* $\sqcap X$ of $X \subseteq S$ are *dual* (i.e. their definition is obtained from that of upper bounds and the least upper bound by replacing $\sqsubseteq$ by its dual $\sqsupseteq$).

A *poset* $P(\sqsubseteq)$ is a partial order $\sqsubseteq$ on a set $P$. A *complete lattice* $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ is a poset $L(\sqsubseteq)$ such that any subset $X$ of $L$ has a least upper bound $\sqcup X$ and a greatest lower bound $\sqcap X$. In particular, the *infimum* $\bot = \sqcup\emptyset = \sqcap L$ is the smallest element of $L$ whilst the *supremum* $\top = \sqcap\emptyset = \sqcup L$ is the greatest. A *linear order* $\sqsubseteq$ is a partial order such that any two elements of $P$ are comparable: $\forall x, y \in P : x \sqsubseteq y \ \vee\ y \sqsubseteq x$. An *increasing chain* is a subset $X$ of $P$ such that $\sqsubseteq$ is a linear order on $X$. A *complete partial order*, for short *cpo*, is a poset such that every increasing chain has a least upper bound. A *strict* cpo has an infimum.

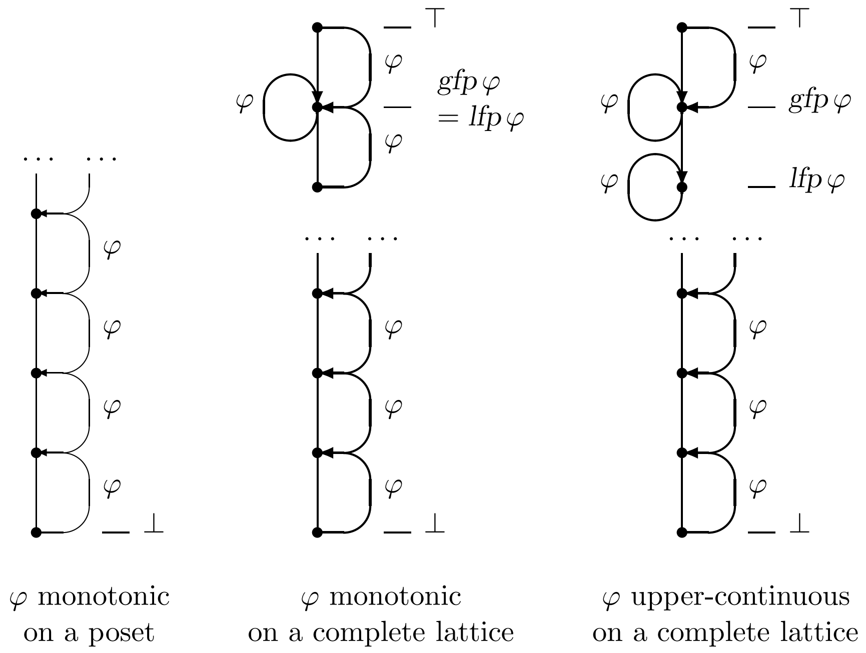



**FIGURE 2.** Fixpoints.

We write $\varphi \in S \rightarrowtail T$ to mean that $\varphi$ is a *partial function* of $S$ into $T$, i.e., a relation $\varphi \in \wp(S \times T)$ such that $\langle s, t\rangle \in \varphi$ only if $s \in S$ and $t \in T$ and, for every $s \in S$, there exists at most one $t \in T$, written $\varphi s$, $\varphi[\![s]\!]$, $\varphi[s]$, or $\varphi(s)$, satisfying $\langle s, t\rangle \in \varphi$. We say that $\varphi(s)$ is *well-defined* when the definition of $\varphi$ implies the existence of $\varphi(s)$. We write $\varphi \in S \mapsto T$ to mean that $\varphi$ is a *total function* of $S$ into $T$ i.e. $\varphi(s)$ is well-defined for all $s$ in $S$ ($\forall s \in S : \exists t \in T : \langle s, t\rangle \in \varphi$). As usual function composition $\circ$ is defined by $\varphi \circ \psi(s) = \varphi(\psi(s))$. The *image* of $X \subseteq S$ by $\varphi \in S \mapsto T$ is $\varphi^*(X) = \{\varphi(x) \mid x \in X\}$. Let $P(\sqsubseteq, \sqcup)$ be a poset with least upper bound $\sqcup$ and $Q(\preceq, \vee)$ be a poset with least upper bound $\vee$. $P(\sqsubseteq) \xmapsto{m} Q(\preceq)$ denotes the set of total functions $\varphi \in P \mapsto Q$ that are *monotone*, i.e., order morphisms: $\forall x \in P : \forall y \in Q : x \sqsubseteq y \Rightarrow \varphi x \preceq \varphi y$. $P(\sqsubseteq, \sqcup) \xmapsto{c} Q(\vee)$ denotes the set of total functions $\varphi \in P \mapsto Q$ that are *upper-continuous*, i.e., which preserve existing least upper bounds of increasing chains: if $X \subseteq P$ is an increasing chain for $\sqsubseteq$ and $\sqcup X$ exists then $\varphi(\sqcup X) = \vee\varphi^*(X)$. $P(\sqcup) \xmapsto{a} Q(\vee)$ denotes the set of total functions $\varphi \in P \mapsto Q$ that are *additive*, i.e., complete join-morphisms preserving least upper bounds of arbitrary subsets, when they exist: if $X \subseteq P$ and $\sqcup X$ exists then $\varphi(\sqcup X) = \vee\varphi^*(X)$. When the above notions are restricted to sets equipotent with the set $\mathbb{N}$ of natural numbers, they are qualified by the attribute $\omega$ as in *$\omega$-chain*, *$\omega$-cpo*, *$\omega$-continuity*, etc.

A *fixpoint* $x \in P$ of $\varphi \in P \mapsto P$ is such that $\varphi x = x$. We write $\varphi^{=}$ for the set $\{x \in P \mid \varphi x = x\}$ of fixpoints of $\varphi$. The *least fixpoint* $lfp\,\varphi$ of $\varphi$ is the unique $x \in \varphi^{=}$ such that $\forall y \in \varphi^{=} : x \sqsubseteq y$. The dual notion is that of *greatest fixpoint* $gfp\,\varphi$. By Tarski's fixpoint theorem [52], the fixpoints of a monotone mapping $\varphi \in L(\sqsubseteq) \xmapsto{m} L(\sqsubseteq)$ on a complete lattice $L(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ form a complete lattice $\varphi^{=}$ for $\sqsubseteq$ with infimum $lfp\,\varphi = \sqcap\varphi^{\sqsubseteq}$ and supremum $gfp\,\varphi = \sqcup\varphi^{\sqsupseteq}$ where $\varphi^{\sqsubseteq} = \{x \in L \mid \varphi x \sqsubseteq x\}$ is the set of *postfixpoints* and $\varphi^{\sqsupseteq} = \{x \in L \mid \varphi x \sqsupseteq x\}$ is the set of *prefixpoints* of $\varphi$. Moreover, if $\varphi$ is $\omega$-upper-continuous (hence, in particular, additive), $lfp\,\varphi = \sqcup_{n\geq 0}\varphi^n(\perp)$ where $\varphi^0(x) = \perp$ and $\varphi^{n+1}(x) = \varphi(\varphi^n(x))$ for all $x \in L$. This is illustrated in Figure 2 (where a poset $P$ is represented by its Hasse diagram so that its elements are figured by points, a point being above and linked by a line to another if it corresponds to a strictly greater element and a function $\varphi$ is represented by its sagittal graph using arrows linking all each point for $x \in P$ to the point corresponding to $\varphi(x)$).

### *4.2. Approximation of Fixpoint Semantics by Simplification Using Galois Connections*

Two fixpoint approximation methods were considered in [10]. One is static in that it can be understood as the simplification of the equation involved in the concrete semantics into an approximate abstract equation which solution provides the abstract semantics. Galois connections are used to formalize this discrete approximation process. The second is dynamic in that it takes place during the iterative resolution of the abstract equation (or system of equations). This separation introduces additional flexibility allowing for both expressiveness and efficiency.

#### *4.2.1. Approximation of Concrete Program Properties by Abstract Properties*

We assume that the concrete program properties are described by elements of a given set $P^\flat$. Let $\preceq^\flat$ be a partial order relation on $P^\flat$ defining the relative precision of concrete properties: ${p^\flat}_1 \preceq^\flat {p^\flat}_2$ means that ${p^\flat}_1$ and ${p^\flat}_2$ are comparable properties of the program, ${p^\flat}_1$ being more precise than ${p^\flat}_2$, the relative precision being left unquantified. The abstract program properties are assumed to be represented by elements of a poset $P^\sharp(\preceq^\sharp)$ where the partial order relation $\preceq^\sharp$ defines the relative precision of abstract properties.

*Example 1 (Rule of signs).* For a trivial example, we can chose $P^\flat = \{$false, $<0$, $=0$, $>0$, $\leq 0$, $\neq 0$, $\geq 0$, true$\}$ with the intended meaning that these properties refer to the possible values $x$ of some program variable and therefore false $\stackrel{\text{def}}{=} \emptyset$, $<0 \stackrel{\text{def}}{=} \{x \in \mathbb{Z} \mid x < 0\}$, $=0 \stackrel{\text{def}}{=} \{0\}$, ... , true $\stackrel{\text{def}}{=} \mathbb{Z}$. For example, $=0 \preceq^\flat \geq 0$ since "is equal to zero" is more precise than "is positive or zero" (but it would be difficult to say of how much!). Hence for this example $\preceq^\flat$ is the subset ordering $\subseteq$. A possible approximation of $P^\flat$ would be $P^\sharp = \{f^\sharp, -1, 0, +1, t^\sharp\}$ where strict inequalities are ignored. $\square$

The semantics of the abstract properties is given by a concretization function $\gamma \in P^\sharp \mapsto P^\flat$: $\gamma(p^\sharp)$ is the concrete property corresponding to the abstract description $p^\sharp \in P^\sharp$. The notion of approximation is formalized by an abstraction function $\alpha \in P^\flat \mapsto P^\sharp$ giving the best abstract approximation $\alpha(p^\flat)$ of concrete properties $p^\flat \in P^\flat$.

*Example 2 (Rule of signs, continued).* The concretization function for our trivial example is given in Figure 3, (where posets $P^\flat(\preceq^\flat)$ and $P^\sharp(\preceq^\sharp)$ are represented by their Hasse diagrams and $\gamma$ by its sagittal graph). For example, $+1$ means $\geq 0$ that is "belonging to the set of zero or positive integers". In Figure 3, we would have:

| $p^\flat$ | false | $<0$ | $=0$ | $>0$ | $\leq 0$ | $\neq 0$ | $\geq 0$ | true |
|---|---|---|---|---|---|---|---|---|
| $\alpha(p^\flat)$ | $f^\sharp$ | $-1$ | $0$ | $+1$ | $-1$ | $\top$ | $+1$ | $t^\sharp$ |

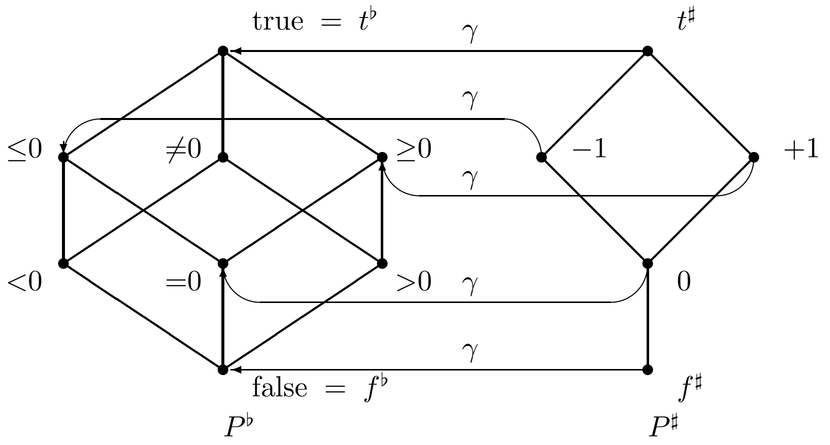



**FIGURE 3.** Concretization function.

with the obvious meaning that, e.g. "is strictly positive" can be approximate from above by "is zero or positive" for subset approximation ordering $\subseteq$. $\square$

If ${p^\sharp}_1 = \alpha(p^\flat)$ and ${p^\sharp}_1 \preceq^\sharp {p^\sharp}_2$ then ${p^\sharp}_2$ is also a correct, although less precise abstract approximation of the concrete property $p^\flat$. Hence, the soundness of approximations, i.e., the fact that $p^\sharp$ is a valid approximation of the information given by $p^\flat$ can be expressed by $\alpha(p^\flat) \preceq^\sharp p^\sharp$. If ${p^\flat}_1 = \gamma(p^\sharp)$ and ${p^\flat}_2 \preceq^\flat {p^\flat}_1$ then $p^\sharp$ is also a correct approximation of the concrete property ${p^\flat}_2$ although this concrete property ${p^\flat}_2$ provides more accurate information about program executions than ${p^\flat}_1$. So the soundness of approximations, that is the fact that $p^\sharp$ is a valid approximation of the information given by $p^\flat$, can also be expressed by $p^\flat \preceq^\flat \gamma(p^\sharp)$. When these two soundness conditions are equivalent, we have got a Galois connection. We now examine more precisely the motivations for and consequences of this hypothesis. This requires the study of mathematical properties of Galois connections.

### *4.2.2. Galois Connections*

Given posets $P^\flat(\preceq^\flat)$ and $P^\sharp(\preceq^\sharp)$, a *Galois connection* is a pair of maps such that:

$$\begin{array}{rcl} \alpha & \in & P^\flat \mapsto P^\sharp \\ \gamma & \in & P^\sharp \mapsto P^\flat \\ \multicolumn{3}{c}{\forall p^\flat \in P^\flat : \forall p^\sharp \in P^\sharp : \alpha(p^\flat) \preceq^\sharp p^\sharp \Leftrightarrow p^\flat \preceq^\flat \gamma(p^\sharp)} \end{array} \tag{1}$$

in which case we write:

$$P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$$

Galois connections have numerous properties, which are recalled in [15] (particularly theorems 5.3.0.5 and 5.3.0.7), where the references to the mathematical literature are also found. For example, $\gamma \circ \alpha$ is *extensive*:

$$\forall p^\flat \in P^\flat : p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat) \tag{2}$$

since $\alpha(p^\flat) \preceq^\sharp \alpha(p^\flat)$ by reflexivity, hence $p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat)$ by (1) with $p^\sharp = \alpha(p^\flat)$. This can be interpreted by the fact that the loss of information in the abstraction process is sound. The same way, $\alpha \circ \gamma$ is *reductive*:

$$\forall p^\sharp \in P^\sharp : \alpha \circ \gamma(p^\sharp) \preceq^\sharp p^\sharp \tag{3}$$

since $\gamma(p^\sharp) \preceq^\flat \gamma(p^\sharp)$ by reflexivity, hence $\alpha \circ \gamma(p^\sharp) \preceq^\sharp p^\sharp$ by (1) with $p^\flat = \gamma(p^\sharp)$. This can be interpreted by the fact that the concretization process introduces no loss of information. From an abstract point of view, $\alpha(p^\flat)$ is as precise as possible.

It follows that $\alpha$ is monotone [since ${p^\flat}_1 \preceq^\flat {p^\flat}_2$ implies ${p^\flat}_1 \preceq^\flat \gamma \circ \alpha({p^\flat}_2)$ by (2) and transitivity whence $\alpha({p^\flat}_1) \preceq^\sharp \alpha({p^\flat}_2)$ by (1)] and so is $\gamma$ [since ${p^\sharp}_1 \preceq^\sharp {p^\sharp}_2$ implies $\alpha \circ \gamma({p^\sharp}_1) \preceq^\sharp {p^\sharp}_2$ by (3) and transitivity whence $\gamma({p^\sharp}_1) \preceq^\flat \gamma({p^\sharp}_2)$ by (1)]:

$$\begin{array}{rcl} \alpha & \in & P^\flat(\preceq^\flat) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp) \\ \gamma & \in & P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\flat(\preceq^\flat) \end{array} \tag{4}$$

Monotony can be interpreted as the fact that the abstraction and concretization process preserves the soundness of the approximation.

(2), (3), and (4) imply (1), hence can be chosen as an equivalent definition of Galois connections:

$$\begin{array}{rcl} P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp) & \Longleftrightarrow & [\alpha \in P^\flat(\preceq^\flat) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)] \;\wedge\; [\gamma \in P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\flat(\preceq^\flat)] \;\wedge \\ & & [\forall p^\flat \in P^\flat : p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat)] \;\wedge\; [\forall p^\sharp \in P^\sharp : \alpha \circ \gamma(p^\sharp) \preceq^\sharp p^\sharp] \end{array} \tag{5}$$

Observe that $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$ if and only if $P^\sharp({\preceq^\sharp}^{-1}) \xleftrightarrows[\gamma]{\alpha} P^\flat({\preceq^\flat}^{-1})$ where the inverse $\leq^{-1}$ of the partial order $\leq$ is $\geq$. It follows that the duality principle on posets stating that any theorem is true for all posets, then so is its dual obtained by substituting $\geq$, $>$, $\top$, $\bot$, $\vee$, $\wedge$, etc. respectively for $\leq$, $<$, $\bot$, $\top$, $\wedge$, $\vee$, etc. can be extended to Galois connections by exchanging $\alpha$ and $\gamma$.

For all $p^\flat \in P^\flat$ and $p^\sharp \in P^\sharp$, we have $\alpha \circ \gamma(p^\sharp) \preceq^\sharp p^\sharp$ by (3) whence by monotony $\gamma \circ \alpha \circ \gamma(p^\sharp) \preceq^\sharp \gamma(p^\sharp)$. Moreover, $\gamma(p^\sharp) \preceq^\sharp \gamma \circ \alpha \circ \gamma(p^\sharp)$ by (2) when $p^\flat$ is $\gamma(p^\sharp)$. By antisymmetry, we conclude that:

$$\forall p^\sharp \in P^\sharp : \gamma \circ \alpha \circ \gamma(p^\sharp) = \gamma(p^\sharp) \tag{6}$$

The same way, for all $p^\flat \in P^\flat$ we have $\alpha \circ \gamma(\alpha(p^\flat)) \preceq^\sharp \alpha(p^\flat)$ by letting $p^\sharp = \alpha(p^\flat)$ in (3). Moreover, (2) implies that for all $p^\flat \in P^\flat$ we have $p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat)$ whence by monotony $\alpha(p^\flat) \preceq^\flat \alpha \circ \gamma \circ \alpha(p^\flat)$. By antisymmetry, we conclude:

$$\forall p^\flat \in P^\flat : \alpha \circ \gamma \circ \alpha(p^\flat) = \alpha(p^\flat) \tag{7}$$

An immediate consequence is that a Galois connection defines closure operators, as follows (a *lower closure operator* is monotone, reductive, and idempotent, whereas an *upper closure operator* is monotone, extensive, and idempotent):

$$P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp) \implies \begin{cases} \alpha \circ \gamma \text{ is a lower closure operator,} \\ \gamma \circ \alpha \text{ is an upper closure operator.} \end{cases} \tag{8}$$

Idempotence, i.e., $\rho \circ \rho = \rho$, can be interpreted as the fact that all information is lost at once in the abstract interpretation process so that two successive abstractions with the same abstraction function are equivalent to a single one. Another consequence is that one can reason upon the abstract interpretation using only $P^\flat$ and the image of $P^\flat$ by the closure operator $\gamma \circ \alpha$ (instead of $P^\sharp$). This equivalent approach is considered in [15]. In particular, the use of *Moore families*, i.e., containing $t^\flat$ and closed under arbitrary $\wedge^\flat$, is justified by the following:

*Proposition 3 (Moore family). If $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$ and $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ is a complete lattice then $\gamma^*(P^\sharp)$ is a Moore family.*

PROOF. If $p^\flat \in \gamma^*(P^\sharp)$ then $\exists p^\sharp \in P^\sharp : p^\flat = \gamma(p^\sharp) \preceq^\flat t^\flat$, hence by monotony and (6) $p^\flat = \gamma(p^\sharp) = \gamma \circ \alpha \circ \gamma(p^\sharp) \preceq^\flat \gamma \circ \alpha(t^\flat)$, proving that $\gamma \circ \alpha(t^\flat) = t^\flat$ is the supremum of $\gamma^*(P^\sharp)$.

Assume that $X \subseteq \gamma^*(P^\sharp)$. If $p^\flat \in X$, then $\exists p^\sharp \in P^\sharp$ such that $p^\flat = \gamma(p^\sharp)$. Then $\wedge^\flat X$ exists in a complete lattice and satisfies $\wedge^\flat X \preceq^\flat p^\flat$ so that by monotony and (6) $\gamma \circ \alpha(\wedge^\flat X) \preceq^\flat \gamma \circ \alpha(p^\flat) = \gamma \circ \alpha \circ \gamma(p^\sharp) = \gamma(p^\sharp) = p^\flat$ proving that $\gamma \circ \alpha(\wedge^\flat X)$ is a lower bound of $X$ so that $\gamma \circ \alpha(\wedge^\flat X) \preceq^\flat \wedge^\flat X$. But $\gamma \circ \alpha$ is extensive by proposition 8 so that by antisymmetry $\gamma \circ \alpha(\wedge^\flat X) = \wedge^\flat X$ proving that $\wedge^\flat X \in \gamma^*(P^\sharp)$. □

In a Galois connection, one function uniquely determines the other:

*Proposition 4. If $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha_1]{\gamma_1} P^\sharp(\preceq^\sharp)$ and $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha_2]{\gamma_2} P^\sharp(\preceq^\sharp)$, then $(\alpha_1 = \alpha_2)$ if and only if $(\gamma_1 = \gamma_2)$.*

PROOF. Assume that $\alpha_1 = \alpha_2$. For all $p^\sharp \in P^\sharp$, $\alpha_2 \circ \gamma_2(p^\sharp) \preceq^\sharp p^\sharp$ by (3), hence $\alpha_1 \circ \gamma_2(p^\sharp) \preceq^\sharp p^\sharp$ by hypothesis and therefore $\gamma_2(p^\sharp) \preceq^\flat \gamma_1(p^\sharp)$ by (1). The same way, $\alpha_1 \circ \gamma_1(p^\sharp) \preceq^\sharp p^\sharp$ by (3) hence $\alpha_2 \circ \gamma_1(p^\sharp) \preceq^\sharp p^\sharp$ by hypothesis and therefore $\gamma_1(p^\sharp) \preceq^\flat \gamma_2(p^\sharp)$ by (1). By antisymmetry, we conclude that $\gamma_1(p^\sharp) = \gamma_2(p^\sharp)$. The reciprocal follows from the duality principle. □

The practical consequence of this fact is that we can perform an abstract interpretation by defining the abstraction or, indifferently, the concretization function, since the adjoined function is uniquely determined as follows:

*Proposition 5.* *If* $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$, *then, for all* $p^\flat \in P^\flat$, $\alpha(p^\flat)$ *is equal to the greatest lower bound* $\bigwedge^\sharp\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$ *of the inverse image by* $\gamma$ *of the set of upper bounds of* $p^\flat$. *For all* $p^\sharp \in P^\sharp$ *we have* $\gamma(p^\sharp) = \bigvee^\flat\{p^\flat \mid \alpha(p^\flat) \preceq^\sharp p^\sharp\}$.

PROOF. If $p^\flat \preceq^\flat \gamma(p^\sharp)$ then $\alpha(p^\flat) \preceq^\sharp p^\sharp$ by (1) so that $\alpha(p^\flat)$ is a lower bound of $\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$. Moreover $p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat)$ by (2) so that $\alpha(p^\flat)$ belongs to $\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$. It follows that $\alpha(p^\flat)$ is the greatest lower bound of $\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$ since for any other lower bound $\ell$, we must have $\ell \preceq^\sharp \alpha(p^\flat)$. The dual result holds for $\gamma$. □

Another important property of Galois connections is the preservation of bounds:

*Proposition 6.* *If* $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$, *then* $\alpha \in P^\flat(\vee^\flat) \xmapsto{a} P^\sharp(\vee^\sharp)$ *preserves least upper bounds and* $\gamma \in P^\sharp(\wedge^\sharp) \xmapsto{a} P^\flat(\wedge^\flat)$ *preserves greatest lower bounds.*

PROOF. Assume that $X$ is a subset of $P^\flat$ such that $\bigvee^\flat X$ exists. For all $x \in X$ we have $x \preceq^\flat \bigvee^\flat X$ by definition of least upper bounds so that $\alpha(x) \preceq^\sharp \alpha(\bigvee^\flat X)$ by monotony, proving that $\alpha(\bigvee^\flat X)$ is an upper bound of the $\alpha(x)$. Let $m$ be another upper bound of all $\alpha(x)$, $x \in X$. We have $\alpha(x) \preceq^\sharp m$, whence $x \preceq^\flat \gamma(m)$ by (1) so that $\bigvee^\flat X \preceq^\flat \gamma(m)$ by definition of least upper bounds. By monotony and (3) it follows that $\alpha(\bigvee^\flat X) \preceq^\sharp \alpha \circ \gamma(m) \preceq^\sharp m$, proving that $\alpha(\bigvee^\flat X)$ is the least upper bound of $\{\alpha(x) \mid x \in X\}$. By the duality principle, it follows that $\forall X \subseteq P^\sharp : \gamma(\bigwedge^\sharp X) = \bigwedge^\flat\{\gamma(x) \mid x \in X\}$ when $\bigwedge^\sharp X$ exists. □

Whenever we have defined an abstraction function that is a complete join morphism or a concretization function that is a complete meet morphism, then this definition entirely determines a unique Galois connection, provided that the bounds allowing for the definition of the adjoined function exist (which is the case, for example, when considering complete lattices):

*Proposition 7.* *Let* $P^\flat(\preceq^\flat)$ *and* $P^\sharp(\preceq^\sharp)$ *be posets. If* $\alpha \in P^\flat(\bigvee^\flat) \xmapsto{a} P^\sharp(\bigvee^\flat)$ *and* $\bigvee^\flat\{p^\flat \mid \alpha(p^\flat) \preceq^\sharp p^\sharp\}$ *exists for all* $p^\sharp \in P^\sharp$, *then* $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$ *where* $\forall p^\sharp \in P^\sharp : \gamma(p^\sharp) = \bigvee^\flat\{p^\flat \mid \alpha(p^\flat) \preceq^\sharp p^\sharp\}$. *If* $\gamma \in P^\sharp(\bigwedge^\sharp) \xmapsto{a} P^\flat(\bigwedge^\flat)$ *and* $\bigwedge^\sharp\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$ *exists for all* $p^\flat \in P^\flat$, *then* $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$ *where* $\forall p^\flat \in P^\flat : \alpha(p^\flat) = \bigwedge^\sharp\{p^\sharp \mid p^\flat \preceq^\flat \gamma(p^\sharp)\}$.

PROOF. If $\alpha(p^\flat) \preceq^\sharp p^\sharp$ then $p^\flat \in \{{p^\flat}' \mid \alpha({p^\flat}') \preceq^\sharp p^\sharp\}$, whence $p^\flat \preceq^\flat \bigvee^\flat\{{p^\flat}' \mid \alpha({p^\flat}') \preceq^\sharp p^\sharp\} = \gamma(p^\sharp)$ by definition of least upper bounds and of $\gamma$. Reciprocally, if $p^\flat \preceq^\flat \gamma(p^\sharp)$ then by definition of $\gamma$ and monotony $\alpha(p^\flat) \preceq^\sharp \alpha(\bigvee^\flat\{{p^\flat}' \mid \alpha({p^\flat}') \preceq^\sharp p^\sharp\})$, which is equal to $\bigvee^\sharp\{\alpha({p^\flat}') \mid \alpha({p^\flat}') \preceq^\sharp p^\sharp\}$ since $\alpha$ preserves least upper bounds, proving that $\alpha(p^\flat) \preceq^\sharp p^\sharp$ by definition of least upper bounds and transitivity. A dual result holds for $\gamma$. □

By eliminating the "useless" abstract properties in the abstract domain $P^\sharp$ that are not the abstraction of some concrete property, we obtain an abstraction onto $P^\sharp$, a situation that can be characterized as follows:

*Proposition 8.* *If* $P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$, *then* $\alpha$ *is onto if and only if* $\gamma$ *is one-to-one if and only if* $\forall p^\sharp \in P^\sharp : \alpha \circ \gamma(p^\sharp) = p^\sharp$ *(in which case the Galois connection is said to be a* Galois surjection*).* $\alpha$ *is one-to-one if and only if* $\gamma$ *is onto if and only if* $\forall p^\flat \in P^\flat : \gamma \circ \alpha(p^\flat) = p^\flat$ *(in which case the Galois connection is said to be a* Galois injection*).*

PROOF. By (7) we have $\alpha \circ \gamma \circ \alpha(p^\flat) = \alpha(p^\flat)$ hence $\gamma \circ \alpha(p^\flat) = p^\flat$ for all $p^\flat \in P^\flat$ if $\alpha$ is one-to-one. In this case $\gamma$ is onto since $p^\flat = \gamma(p^\sharp)$ by choosing $p^\sharp = \alpha(p^\flat)$. The same way, $\gamma \circ \alpha \circ \gamma(p^\sharp) = \gamma(p^\sharp)$ by (6), whence $p^\sharp = \alpha \circ \gamma(p^\sharp)$ if $\gamma$ is one-to-one. In this case, it follows that $\alpha$ is onto since $p^\sharp = \alpha(p^\flat)$ by choosing $p^\flat = \gamma(p^\sharp)$. We conclude by application of the duality principle. □

This leads to the definition of *Galois surjections*:

$$P^\flat(\preceq^\flat) \xtwoheadleftrightarrow[\alpha]{\gamma} P^\sharp(\preceq^\sharp) \stackrel{\text{def}}{=} (P^\flat(\preceq^\flat) \xleftrightarrows[\alpha]{\gamma} P^\sharp(\preceq^\sharp)) \wedge (\forall p^\sharp \in P^\sharp : \alpha \circ \gamma(p^\sharp) = p^\sharp)) \qquad (9)$$

with $\longrightarrow$ denoting 'into' and $\twoheadrightarrow$ 'onto'. Galois surjections induce the order structure from concrete onto abstract properties:

*Proposition 9.* *If* $P^\flat(\preceq^\flat) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$ *and* $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ *is a complete lattice, then so is* $P^\sharp(\preceq^\sharp)$.

PROOF. Given any subset $X$ of $P^\sharp$, $p^\flat = \bigvee^\flat\{\gamma(p^\sharp) \mid p^\sharp \in X\}$ exists in the complete lattice $P^\flat$. Given $p^\sharp \in X$ we have $\gamma(p^\sharp) \preceq^\flat p^\flat$, whence by monotony and Galois surjection characteristic property $p^\sharp = \alpha \circ \gamma(p^\sharp) \preceq^\sharp \alpha(p^\flat)$ proving that $\alpha(p^\flat)$ is an upper bound of $X$.

Let $\ell$ be another upper bound of $X$. For all $p^\sharp \in X$, we have $p^\sharp \preceq^\sharp \ell$ and $\gamma(p^\sharp) \preceq^\flat \gamma(\ell)$ by monotony, whence $p^\flat \preceq^\flat \gamma(\ell)$ by definition of least upper bounds. By the Galois surjection characteristic property and monotony, $\alpha(p^\flat) \preceq^\sharp \alpha \circ \gamma(\ell) = \ell$ proving that $\alpha(p^\flat) = \vee^\sharp X$.

The proof that $\alpha(\bigwedge^\flat\{\gamma(p^\sharp) \mid p^\sharp \in X\})$ is the greatest lower bound of $X \subseteq P^\sharp$ is dual. □

As observed in theorem 10.1.0.2 of [15], each abstract property $p^\sharp$ can be improved by its lower closure $\alpha \circ \gamma(p^\sharp)$. This leads to a systematic way of obtaining Galois surjections from Galois connections by identification of the abstract properties $p^\sharp$, which meaning $\gamma(p^\sharp)$ cannot be distinguished at the concrete level, into an equivalence class:

*Proposition 10 (Reduction).* *If* $P^\flat(\preceq^\flat) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, *then* ${p^\sharp}_1 \equiv {p^\sharp}_2 \overset{\text{def}}{=} \gamma({p^\sharp}_1) = \gamma({p^\sharp}_2)$ *is an equivalence relation such that* $P^\flat(\preceq^\flat) \overset{\gamma_\equiv}{\underset{\alpha_\equiv}{\leftrightarrows}} P^\sharp/_\equiv(\preceq^\sharp_\equiv)$ *where* $\alpha_\equiv(p^\flat) \overset{\text{def}}{=} \{p^\sharp \mid p^\sharp \equiv \alpha(p^\flat)\}$, $\gamma_\equiv(X)$ *is* $\gamma(p^\sharp)$ *such that* $p^\sharp \in X$ *and* $X \preceq^\sharp_\equiv Y \overset{\text{def}}{=} \exists {p^\sharp}_1 \in X : \exists {p^\sharp}_2 \in Y : {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2$.

PROOF. We have $[\alpha_\equiv(p^\flat) \preceq^\sharp_\equiv X] \Leftrightarrow [\exists {p^\sharp}_1 \in \alpha_\equiv(p^\flat) : \exists {p^\sharp}_2 \in X : {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2] \Leftrightarrow [\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : {p^\sharp}_1 \equiv \alpha(p^\flat) \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2] \Leftrightarrow [\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : \gamma({p^\sharp}_1) = \gamma \circ \alpha(p^\flat) \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2]$, which implies $[\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : \alpha \circ \gamma \circ \alpha(p^\flat) = \alpha \circ \gamma({p^\sharp}_1) \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2]$, and, therefore, by (7) and (3), we have $[\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : \alpha(p^\flat) \preceq^\sharp {p^\sharp}_1 \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2]$, which implies $[\exists {p^\sharp}_2 \in X : \alpha(p^\flat) \preceq^\sharp {p^\sharp}_2]$ $\Leftrightarrow [\exists {p^\sharp}_2 \in X : p^\flat \preceq^\flat \gamma({p^\sharp}_2)] \Leftrightarrow [p^\flat \preceq^\flat \gamma_\equiv(X)]$. Reciprocally, $[\exists {p^\sharp}_2 \in X : \alpha(p^\flat) \preceq^\sharp {p^\sharp}_2]$ implies $[\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : {p^\sharp}_1 = \alpha(p^\flat) \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2]$, whence $[\exists {p^\sharp}_1 : \exists {p^\sharp}_2 \in X : {p^\sharp}_1 \equiv \alpha(p^\flat) \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2]$. □

From a practical point of view, this proposition corresponds to the use of a normal form for abstract properties with the same meaning. We use the following notation for the reduction:

$$P^\flat(\preceq^\flat) \overset{\gamma}{\underset{\alpha}{\leftrightarrows\!\!\!\!=}} P^\sharp(\preceq^\sharp) \quad \overset{\text{def}}{=} \quad P^\flat(\preceq^\flat) \overset{\gamma_\equiv}{\underset{\alpha_\equiv}{\leftrightarrows}} P^\sharp/_\equiv(\preceq^\sharp_\equiv) \tag{10}$$

### 4.2.3. *The Compositional Design of Galois Connections*

We now study systematic ways of defining Galois connections so as to specify program analysers by successive refinements.

4.2.3.1. COMPOSITION OF GALOIS CONNECTIONS. The composition of Galois connections is a Galois connection. This fundamental property is the basis for designing program analysers by composition of successive approximations:

$$\left(P^\flat(\preceq^\flat) \overset{\gamma_1}{\underset{\alpha_1}{\leftrightarrows}} P^\natural(\preceq^\natural) \wedge P^\natural(\preceq^\natural) \overset{\gamma_2}{\underset{\alpha_2}{\leftrightarrows}} P^\sharp(\preceq^\sharp)\right) \Rightarrow P^\flat(\preceq^\flat) \overset{\gamma_1 \circ \gamma_2}{\underset{\alpha_2 \circ \alpha_1}{\leftrightarrows}} P^\sharp(\preceq^\sharp) \tag{11}$$

For example, in [10], $\langle \alpha_1, \gamma_1 \rangle$ decomposes global invariants on program counters and values of variables into local invariants upon the values of the variables attached to program points, then $\langle \alpha_2, \gamma_2 \rangle$ decomposes the relational local invariants into attribute independent ones, and then $\langle \alpha_3, \gamma_3 \rangle$ approximates the set of possible values of each variable by an abstract value such as its sign, parity, interval of values, etc.

4.2.3.2. PARTITIONING. One first standard way of obtaining Galois connections is illustrated by the decomposition of a global invariant into local invariants attached to program points [15], example 6.2.0.2.

*Example 11 (Local invariants).* More precisely, consider invariance properties represented as a set $p^\flat$ of states belonging to $S$ so that $P^\flat = \wp(S)$. If states are pairs $\langle c, m\rangle$ where $c \in C$ is a control state (a control point for imperative sequential programs) and $m \in M$ is an environment delivering the values of variables then $p^\flat$ can be abstracted as a vector $p^\sharp = \alpha(p^\flat)$ such that $p^\sharp[c] = \{m \mid \langle c, m\rangle \in p^\flat\}$ for all $c \in C$. The meaning of such an abstract value $p^\sharp$ is $\gamma(p^\sharp) = \{\langle c, m\rangle \mid c \in C \wedge m \in p^\sharp[c]\}$. $\alpha$ is a bijection with inverse $\gamma$ so that the concrete and abstract representations of an invariant are isomorphic: an invariant can be represented globally as a predicate on the control and memory states or locally as a set of invariants on the memory states attached to each program point. □

This can be easily generalized as follows:

*Proposition 12 (Partitioning). Let $P^\flat(\preceq^\flat, f^\flat, t^\flat, \bigvee^\flat, \bigwedge^\flat)$ be a complete lattice that is (infinitely) distributive for intersection*[1]*, i.e., the join operation is (completely) distributive on meets so that $x \wedge^\flat \bigvee^\flat X = \bigvee^\flat \{x \wedge^\flat y \mid y \in X\}$ for all $x \in P^\flat$ and any (infinite) set $X \subseteq P^\flat$. Let $L$ be a non-empty finite (respectively infinite) set of so-called* labels *and $\delta \in L \mapsto P^\flat$ be a* partition *of $P^\flat$ (satisfying the* cover property *$t^\flat = \bigvee^\flat_{\ell \in L} \delta(\ell)$ and the* disjointness property *$\forall \ell, \ell' \in L : \ell \neq \ell' \Rightarrow \delta[\ell] \wedge^\flat \delta[\ell'] = f^\flat$). Define $P^\sharp = \prod_{\ell \in L}\{p^\flat \wedge^\flat \delta(\ell) \mid p^\flat \in P^\flat\}$ with the componentwise ordering ${p^\sharp}_1 \preceq^\sharp {p^\sharp}_2$ if and only if $\forall \ell \in L : {p^\sharp}_1[\ell] \preceq^\flat {p^\sharp}_2[\ell]$. Let $\alpha(p^\flat)[\ell] = p^\flat \wedge^\flat \delta(\ell)$ and $\gamma(p^\sharp) = \bigvee^\flat_{\ell' \in L} p^\sharp[\ell']$ for all $\ell \in L$, $p^\flat \in P^\flat$ and $p^\sharp \in P^\sharp$. Then the* partitioning *is such that $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$ whereas the* reduced partitioning *satisfies $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\rightarrow}} P^\sharp(\preceq^\sharp)$.*

PROOF. For all $p^\flat \in P^\flat$ and $p^\sharp \in P^\sharp$, if $\alpha(p^\flat) \preceq^\sharp p^\sharp$, then for all $\ell \in L$ we have $p^\flat \wedge^\flat \delta(\ell) = \alpha(p^\flat)[\ell] \preceq^\flat p^\sharp[\ell] \preceq^\flat \bigvee^\flat_{\ell' \in L} p^\sharp[\ell'] = \gamma(p^\sharp)$ by definition of $\alpha$, of the componentwise ordering, of least upper bounds $\bigvee^\flat$ and of $\gamma$. So $p^\flat$ is equal to $p^\flat \wedge^\flat t^\flat$ by definition of the supremum $t^\flat$, hence to $p^\flat \wedge^\flat \bigvee^\flat_{\ell \in L} \delta(\ell)$ by the cover property, so to $\bigvee^\flat_{\ell \in L} p^\flat \wedge^\flat \delta(\ell)$ by $\wedge^\flat$–distributivity, which is upper $\preceq^\flat$–bounded by $\gamma(p^\sharp)$ by definition of least upper bounds.

Reciprocally, if $p^\flat \preceq^\flat \gamma(p^\sharp)$, then $p^\flat \preceq^\flat \bigvee^\flat_{\ell' \in L} p^\sharp[\ell']$ by definition of the concretization $\gamma$, whence for all $\ell \in L$, $\alpha(p^\flat)[\ell] = p^\flat \wedge^\flat \delta[\ell] \preceq^\flat \left(\bigvee^\flat_{\ell' \in L} p^\sharp[\ell']\right) \wedge^\flat \delta[\ell]$ by definition of the abstraction $\alpha$ and of greatest lower bounds. But $\left(\bigvee^\flat_{\ell' \in L} p^\sharp[\ell']\right) \wedge^\flat \delta[\ell]$ is equal to $\bigvee^\flat_{\ell' \in L} (p^\sharp[\ell'] \wedge^\flat \delta[\ell])$ by distributivity. Moreover, $p^\sharp[\ell'] \preceq^\flat \delta[\ell']$ and $\delta[\ell] \wedge^\flat \delta[\ell'] = f^\flat$ so that $p^\sharp[\ell'] \wedge^\flat \delta[\ell'] = f^\flat$ by definition of greatest lower bounds and of the infimum when $\ell \neq \ell'$. It follows that $\bigvee^\flat_{\ell' \in L} (p^\sharp[\ell'] \wedge^\flat \delta[\ell])$ $= p^\sharp[\ell] \wedge^\flat \delta[\ell] = p^\sharp[\ell]$ since $p^\sharp[\ell] \preceq^\flat \delta[\ell]$ by definition of $P^\sharp$. By transitivity and definition of the pointwise ordering $\preceq^\sharp$, we conclude that $\alpha(p^\flat) \preceq^\sharp p^\sharp$. The reduction follows from proposition 10. □

This Galois connection enables us to decompose an equation into a system of equations, one for each label. For logical programs, the choice of labels can vary considerably. For example, one can choose a single one for the whole program, one for each predicate, one for each clause (after head unification), two for each clause (after call and before exit), one before and after each atom of a clause [51], [47] or one before and/or after a call [4] in an AND/OR tree, bi-labels corresponding to pairs of the previous choices or even paths to the calls in the computations within AND/OR trees [61], [59]. The choice of the best decomposition obviously depends upon the kind and quality of the information that is to be gathered about programs and of the acceptable memory and computation costs.

4.2.3.3. REDUCED PRODUCT. If several independent abstract interpretations $P^{\sharp^i}(\preceq^{\sharp^i})$, $i \in \Delta$ have been designed with respect to a concrete domain $P^\flat(\preceq^\flat)$ of program properties using Galois connections $P^\flat(\preceq^\flat) \underset{\alpha^i}{\overset{\gamma^i}{\leftrightarrows}} P^{\sharp^i}(\preceq^{\sharp^i}), i \in \Delta$, then there are many ways to combine them to perform all abstract interpretations simultaneously. Several such combinations of abstract

[1] Such lattices are called *Brouwerian*.

interpretations have been suggested in sections 9 and 10 of [15]. We will use the following ones:

*Proposition 13 (Reduced product). Let* $P^\flat(\preceq^\flat)$ *and* $P^{\sharp^i}(\preceq^{\sharp^i})$ *be posets for all* $i$ *in the index set* $\Delta$ *such that for all* $i$ *in* $\Delta$, $P^\flat(\preceq^\flat) \xleftarrow[\alpha^i]{\gamma^i} P^{\sharp^i}(\preceq^{\sharp^i})$. *Define* $P^\sharp = \prod_{i\in\Delta} P^{\sharp^i}$, ${p^\sharp}_1 \preceq^\sharp {p^\sharp}_2$ *if and only if* $\forall i \in \Delta : {p^\sharp}_1[i] \preceq^{\sharp^i} {p^\sharp}_2[i]$, $\alpha \in P^\flat \mapsto P^\sharp$ *such that* $\alpha(p^\flat) = \prod_{i\in\Delta} \alpha^i(p^\flat)$ *and* $\gamma \in P^\sharp \rightarrowtail P^\flat$ *such that* $\gamma(p^\sharp) = \bigwedge^\flat_{i\in\Delta} \gamma^i(p^\sharp[i])$.

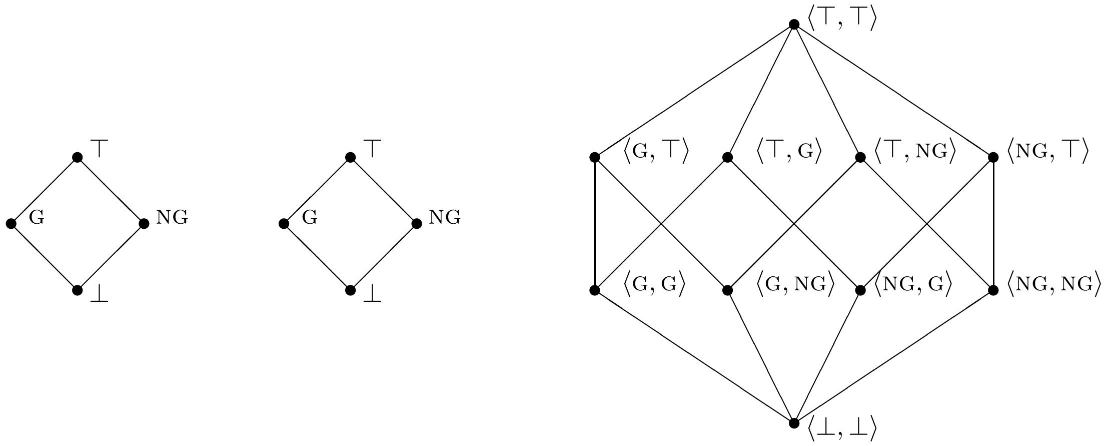
*If* $\Delta$ *is finite and* $P^\flat(\preceq^\flat, \wedge^\flat)$ *is a meet-semi-lattice or* $\Delta$ *is infinite and* $P^\flat(\preceq^\flat, \wedge^\flat)$ *is a complete meet-semi-lattice (hence a complete lattice), then the* product *is such that* $P^\flat(\preceq^\flat) \xleftarrow[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$, *whereas the* reduced product *satisfies* $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$.

PROOF. By definition of $\alpha$ and $\preceq^\sharp$, $\alpha(p^\flat) \preceq^\sharp p^\sharp$ is equivalent to $\forall i \in \Delta : \alpha^i(p^\flat) \preceq^{\sharp^i} p^\sharp[i]$ or, by definition (1) of Galois connections, to $\forall i \in \Delta : p^\flat \preceq^\flat \gamma^i(p^\sharp[i])$. By definition of greatest lower bounds, which exist by the lattice hypothesis, this is equivalent to $p^\flat \preceq^\flat \bigwedge^\flat_{i\in\Delta} \gamma^i(p^\sharp[i])$, i.e., to $p^\flat \preceq^\flat \gamma(p^\sharp)$ by definition of $\gamma$. The reduction follows from proposition 10. □

This combination of abstract interpretations can be qualified of attribute independent. A classical example consists in analysing the possible values of the variables of a program by analysing independently the possible values of each variable in the program, as, for example, in [9]. The information obtained by the combination of the analyses is essentially the same as the one obtained by performing the analyses separately. However, the separate analyses can be mutually improved using proposition 10. For example, the reduced product of sign and parity analysis would exclude the case when a variable is both zero and odd, a situation that may not be recognizable by separate analyses (for example the conjunction of $\{0\}$ `x := 1` $\{+\}$ and $\{\text{odd}\}$ `x := 1` $\{\text{even}\}$ would be $\{\langle 0, \text{odd}\rangle\}$ `x := 1` $\{\langle +, \text{even}\rangle\}$ which reduces to $\{\langle\bot,\bot\rangle\}$ `x := 1` $\{\langle +, \text{even}\rangle\}$ whereas for the reduced product we would have $\{\langle\bot,\bot\rangle\}$ `x := 1` $\{\langle\bot,\bot\rangle\}$).

*Example 14 (Attribute independent groundness analysis).* In groundness analysis, the lattice $P^{\sharp^i}, i = 1, 2$ represents the set of terms to which some logical variable $X^i, i = 1, 2$ can be bounded during execution of a logic program, $\bot$ corresponding to the empty set, G corresponding to the set of ground terms, NG corresponding to the set of terms containing at least one free variable, and $\top$ corresponding to all possible terms. Their reduced product $P^\sharp = \prod_{i\in\{1,2\}} P^{\sharp^i}$ can be used to represent the possible values of the pair of variables $\langle X^1, X^2\rangle$ (which implies that all abstract pairs of values containing $\bot$ are semantically equivalent hence reduced to $\langle\bot,\bot\rangle$, as shown in Figure 4). The analysis is attribute independent in that no relationship can be expressed between the groundness of $X^1$ and that of $X^2$ (such as $X^1$ is ground if and only if $X^2$ is not ground). □

4.2.3.4. DOWN-SET COMPLETION. A method was given in paragraph 9.2 of [15] to provide a disjunctive concrete interpretation of sets of abstract properties. It was used to show that merge over all paths data flow analyses can always be expressed in fixpoint form. This construction is of general use to enrich an abstract interpretation. The intuitive idea is that the abstraction $\alpha$ loses no information about meets (proposition 3), whereas joins are preserved by losing information (proposition 6). For example, in the rule of signs, $\alpha(\{n \in \mathbb{N} \mid n > 0\} \cup \{n \in \mathbb{N} \mid n < 0\}) = \alpha(\{n \in \mathbb{N} \mid n > 0\}) \sqcup \alpha(\{n \mid n < 0\}) = + \sqcup - = \top$, thus losing the information that 0 is impossible. This situation can be improved by moving to the more expressive abstract domain $\wp(P^\sharp)$ and considering sets of abstract values in $P^\sharp$ the meaning of which is the disjunction of the meaning of the individual abstract values in the set. This corresponds to a case analysis. For example, $\{-, +\}$ expresses a non-zero value since $\gamma(\{-,+\}) = \gamma(-) \cup \gamma(+) = \{n \in \mathbb{N} \mid n \neq 0\}$. Now, several sets of abstract values can have the same concrete meaning such as, for example, $\{\top\}$, $\{\top, -\}$, and $\{\top, -, 0, +, \bot\}$. Therefore, a reduction is necessary to reduce the size of the




${P^\sharp}^1$ ${P^\sharp}^2$ Reduced product $P^\sharp = \prod_{i\in\{1,2\}} {P^\sharp}^i$

**FIGURE 4.** Lattice of attribute independent groundness analysis.

abstract lattice, hence that of its computer representation. Proposition 10 can be used for that purpose, but in this case this can be done, at least partially, in a syntactic way, by considering down closed sets only, which contain all abstract values which can be approximated by an element of the set. Following theorems 9.2.0.2 to 9.2.0.4 of [15], this intuitive idea can be formalized as follows:

*Proposition 15 (Down-set completion). Let* $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ *be a complete lattice that is* completely distributive *(i.e.* $\bigwedge^\flat\{\bigvee^\flat\{x_{ij} \mid j \in J_i\} \mid i \in I\} = \bigvee^\flat\{\bigwedge^\flat\{x_{i\varphi(i)} \mid i \in I\} \mid \varphi \in \prod_{k\in I} J_k\}$ *for all* $J_i, i \in I$*) and assume* $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$*. Let* $\downarrow^{\preceq} X = \{y \mid \exists x \in X : y \preceq x\}$ *be the* down closure *of* $X$ *for* $\preceq$*. Define:*

$$\mathfrak{D}^{\preceq^\sharp}(P^\sharp) = {\downarrow^{\preceq^\sharp}}^{*}\,(\wp(P^\sharp))$$

$$\gamma^{\mathfrak{d}} \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp) \mapsto P^\flat \qquad\qquad \alpha^{\mathfrak{d}} \in P^\flat \mapsto \mathfrak{D}^{\preceq^\sharp}(P^\sharp)$$

$$\gamma^{\mathfrak{d}}(X) \stackrel{\text{def}}{=} \bigvee\nolimits^\flat \gamma^*(X) \qquad\qquad \alpha^{\mathfrak{d}}(p^\flat) \stackrel{\text{def}}{=} \cap\{X \mid p^\flat \preceq^\flat \gamma^{\mathfrak{d}}(X)\}$$

*then* $P^\flat\,(\preceq^\flat) \underset{\alpha^{\mathfrak{d}}}{\overset{\gamma^{\mathfrak{d}}}{\leftrightarrows}} \mathfrak{D}^{\preceq^\sharp}(P^\sharp)\,(\subseteq)$ *and* $P^\flat\,(\preceq^\flat) \underset{\alpha^{\mathfrak{d}}}{\overset{\gamma^{\mathfrak{d}}}{\leftrightarrows\!\!\!\!=}} \mathfrak{D}^{\preceq^\sharp}(P^\sharp)\,(\subseteq)$*.*

PROOF. We show that $\gamma^{\mathfrak{d}}$ is a complete meet morphism so that the conclusion follows from proposition 7.

We prove the preliminary lemma stating that $\{\bigwedge^\flat_{i\in I} \varphi[i] \mid \varphi \in \prod_{k\in I}\{p^\flat \mid \exists p^\sharp \in X_k : p^\flat \preceq^\flat \gamma(p^\sharp)\}\} = \{p^\flat \mid \exists p^\sharp \in \bigcap_{i\in I} X_i : p^\flat \preceq^\flat \gamma(p^\sharp)\}$ when $X_k \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp))$ for all $k \in I$. So let $\varphi$ be any element of $\prod_{k\in I}\{p^\flat \mid \exists p^\sharp \in X_k : p^\flat \preceq^\flat \gamma(p^\sharp)\}$ and $k \in I$. Then $\bigwedge^\flat_{i\in I} \varphi[i] \preceq^\flat \varphi[k] \in \{p^\flat \mid \exists p^\sharp \in X_k : p^\flat \preceq^\flat \gamma(p^\sharp)\}$, proving by transitivity that $\exists {p^\sharp}_k \in X_k : \bigwedge^\flat_{i\in I} \varphi[i] \preceq^\flat \gamma({p^\sharp}_k)$. Hence $\bigwedge^\flat_{i\in I} \varphi[i] \preceq^\flat \bigwedge^\flat_{k\in I} \gamma({p^\sharp}_k)$. By proposition 3, $\gamma^*(P^\sharp)$ is a Moore family so there exists some $p^\sharp \in P^\sharp$ such that $\bigwedge^\flat_{k\in I} \gamma({p^\sharp}_k) = \gamma(p^\sharp)$, whence by (6) $\bigwedge^\flat_{i\in I} \varphi[i] \preceq^\flat \gamma(p^\sharp) = \gamma \circ \alpha \circ \gamma(p^\sharp)$. But for all $k \in I$, we have $\gamma(p^\sharp) = \bigwedge^\flat_{k\in I} \gamma({p^\sharp}_k) \preceq^\flat \gamma({p^\sharp}_k)$, whence by monotony and (3), $\alpha \circ \gamma(p^\sharp) \preceq^\sharp \alpha \circ \gamma({p^\sharp}_k) \preceq^\sharp {p^\sharp}_k \in X_k$. But $X_k$ is down closed for $\preceq^\sharp$, whence $\alpha \circ \gamma(p^\sharp) \in X_k$. We conclude that $\alpha \circ \gamma(p^\sharp) \in \bigcap_{k\in I} X_k$ so that $\bigwedge^\flat_{i\in I} \varphi[i] \in \{p^\flat \mid \exists p^\sharp \in \bigcap_{i\in I} X_i : p^\flat \preceq^\flat \gamma(p^\sharp)\}$ proving inclusion in one direction. Reciprocally, assume that $\exists p^\sharp \in \bigcap_{i\in I} X_i : p^\flat \preceq^\flat \gamma(p^\sharp)$. Define $\varphi[i] = p^\flat$ for all $i \in I$. Then $\varphi \in \prod_{k\in I}\{p^\flat \mid \exists p^\sharp \in X_k : p^\flat \preceq^\flat \gamma(p^\sharp)\}$ and $p^\flat = \bigwedge^\flat_{i\in I} \varphi[i]$ proving the inverse inclusion.

To prove that $\gamma^{\mathfrak{d}}$ is a complete meet morphism, we observe that by definition $\gamma^{\mathfrak{d}}(\bigcap_{i\in I} X_i)$ is equal to $\bigvee^\flat \gamma^*(\bigcap_{i\in I} X_i) = \bigvee^\flat \downarrow^{\preceq^\flat}(\{\gamma(p^\sharp) \mid p^\sharp \in \bigcap_{i\in I} X_i\}) = \bigvee^\flat\{p^\flat \mid \exists p^\sharp \in \bigcap_{i\in I} X_i : p^\flat \preceq^\flat \gamma(p^\sharp)\}$, i.e., by the previous lemma, to $\bigvee^\flat\{\bigwedge^\flat_{i\in I} \varphi[i] \mid \varphi \in \prod_{k\in I}\{p^\flat \mid \exists p^\sharp \in X_k : p^\flat \preceq^\flat \gamma(p^\sharp)\}\}$, which by complete distributivity is $\bigwedge^\flat\{\bigvee^\flat\{p^\flat \mid \exists p^\sharp \in X_i : p^\flat \preceq^\flat \gamma(p^\sharp)\} \mid i \in I\}$, which by definition of $\gamma^{\eth}$ is equal to $\bigwedge^{\flat}_{i\in I}\bigvee^{\flat}{\downarrow}^{\preceq^{\flat}}(\{\gamma(p^\sharp) \mid p^\sharp \in X_i\}) = \bigwedge^{\flat}_{i\in I}\bigvee^{\flat}{\downarrow}^{\preceq^{\flat}}(\gamma^*(X_i)).\ = \bigwedge^{\flat}_{i\in I}\bigvee^{\flat}\gamma^*(X_i).\ = \bigwedge^{\flat}_{i\in I}\gamma^{\eth}(X_i)$. The reduction follows from proposition 10. □

*Example 16 (Rule of signs, continued).* Assume that $P^\flat = \wp(\mathbb{Z})$ and $P^\sharp$ is $\{\bot, -, 0, +, \top\}$ with the obvious meaning $\gamma(\bot) = \emptyset$, $\gamma(-) = \{x \in \mathbb{Z} \mid x < 0\}$, $\gamma(0) = \{0\}$, $\gamma(+) = \{x \in \mathbb{Z} \mid x > 0\}$ and $\gamma(\top) = \mathbb{Z}$. Using proposition 15, define $\alpha(X)$ as the least $s \in P^\sharp$ such that $X \subseteq \gamma(s)$. The down-set completion of $P^\sharp$ contains the elements: false $\stackrel{\text{def}}{=} \emptyset \equiv \{\bot\}$; $<0 \stackrel{\text{def}}{=} \{-, \bot\}$; $=0 \stackrel{\text{def}}{=} \{0, \bot\}$; $>0 \stackrel{\text{def}}{=} \{+, \bot\}$; $\leq 0 \stackrel{\text{def}}{=} \{-, 0, \bot\}$; $\neq 0 \stackrel{\text{def}}{=} \{-, +, \bot\}$; $\geq 0 \stackrel{\text{def}}{=} \{+, 0, \bot\}$ and true $\stackrel{\text{def}}{=} \{\top, +, -, 0, \bot\}$ ordered by subset inclusion so that we obtain the lattice that is shown in Figure 5. □

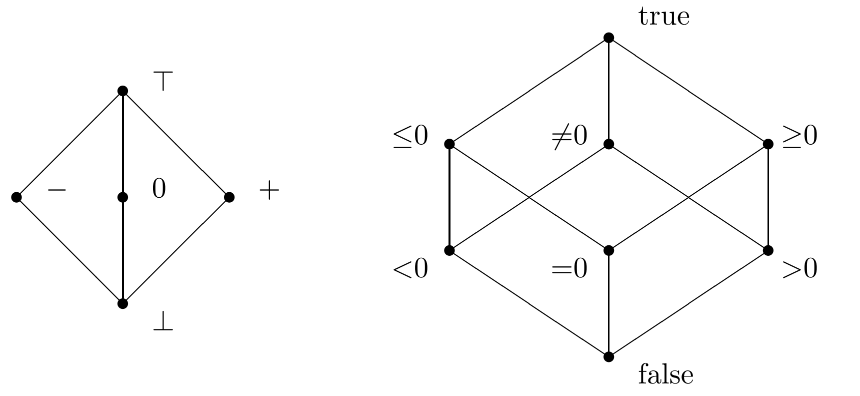
Another equivalent way to define the down-set completion consists in considering Hoare's lower powerdomain that is subsets of $P^\sharp$ pre-ordered by $X \precsim X'$ if and only if $\forall p^\sharp \in X : \exists p^{\sharp\prime} \in X' : p^\sharp \preceq^\sharp p^{\sharp\prime}$. Let $\approx$ be the corresponding equivalence relation defined by $X \approx X' \stackrel{\text{def}}{=} (X \precsim X') \wedge (X' \precsim X)$. The equivalence class of $X \subseteq P^\sharp$ is $[X]_\approx \stackrel{\text{def}}{=} \{X' \subseteq P^\sharp \mid X' \approx X\}$. $\wp(P^\sharp)/_\approx$ is the set of all equivalence classes $[X]_\approx$ for all $X \subseteq P^\sharp$. It is partially ordered by $[X]_\approx \precsim [X']_\approx$ if and only if there exist $Y$, $Y' \subseteq P^\sharp$ such that $(Y \approx X) \wedge (Y' \approx X') \wedge (Y \precsim Y')$. The fact that $\wp(P^\sharp)/_\approx(\precsim)$ is the down-set completion of $P^\sharp$ follows from the following:

*Proposition 17.* *$\wp(P^\sharp)/_\approx(\precsim)$ is order-isomorphic with $\mathfrak{D}^{\preceq^\sharp}(P^\sharp)(\subseteq)$.*

PROOF. Define $\alpha \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp) \mapsto \wp(P^\sharp)/_\approx$ by $\alpha(X) \stackrel{\text{def}}{=} [X]_\approx$ and $\gamma \in \wp(P^\sharp)/_\approx \mapsto \mathfrak{D}^{\preceq^\sharp}(P^\sharp)$ by $\gamma([X]_\approx) \stackrel{\text{def}}{=} {\downarrow}^{\preceq^\sharp}X$. For all $X \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp)$, we have $\gamma \circ \alpha(X) = {\downarrow}^{\preceq^\sharp}X = X$ since $X$ is down-closed. Moreover, $\alpha \circ \gamma([X]_\approx) = [{\downarrow}^{\preceq^\sharp}X]_\approx = [X]_\approx$ since for all $p^\sharp \in X$, we have $p^\sharp \in {\downarrow}^{\preceq^\sharp}X$, whence $X \precsim {\downarrow}^{\preceq^\sharp}X$ since $\preceq^\sharp$ is reflexive and by definition of the down closure, for all $p^\sharp \in {\downarrow}^{\preceq^\sharp}X$ there exists $p^{\sharp\prime} \in X$ such that $p^\sharp \preceq^\sharp p^{\sharp\prime}$, whence ${\downarrow}^{\preceq^\sharp}X \precsim X$. It follows that $\alpha$ is an isomorphism with inverse $\gamma$.

If $X$, $X' \subseteq P^\sharp$, and $X \subseteq X'$, then for all $p^\flat \in X$, $p^{\flat\prime} \in X'$ and $p^\flat \preceq^\sharp p^{\flat\prime}$ so that $X \precsim X'$ and therefore $[X]_\approx \precsim [X']_\approx$. Reciprocally, if $X$, $X' \subseteq P^\sharp$ and $[X]_\approx \precsim [X']_\approx$, then there exist $\bar{X}$, $\bar{X}' \subseteq P^\sharp$ such that $\bar{X} \approx X$, $\bar{X}' \approx X'$ and $\bar{X} \precsim \bar{X}'$ so that by definition of $\approx$ and transitivity, $X \precsim X'$ so that $\forall p^\flat \in X : \exists p^{\flat\prime} \in X : p^\flat \preceq^\sharp p^{\flat\prime}$ whence ${\downarrow}^{\preceq^\sharp}X \subseteq {\downarrow}^{\preceq^\sharp}X'$ proving that $\alpha(X) \precsim \alpha(X')$ which implies $\gamma \circ \alpha(X) \subseteq \gamma \circ \alpha(X')$ that is $X \subseteq X'$. We conclude that $\alpha$ is an order-isomorphism, that is $X \subseteq X'$ if and only if $\alpha(X) \precsim \alpha(X')$. □

The situation observed in example 16 where the down-set completion of $P^\sharp$ is the set of subsets of the atoms $\{-, 0, +\}$ of $P^\sharp$ is in fact more general. An element $a$ of a lattice $L(\leq)$




$P^\sharp$ Down-set completion of $P^\sharp$

**FIGURE 5.** Lattices of signs.

with infimum $\perp$ is an *atom* if it covers $\perp$, that is $\perp < x \leq a \Rightarrow x = a$. $L$ is *atomistic* if and only if every element of $L$ is a join of atoms, and hence of the atoms which it contains. We write $\mathfrak{A}(\mathfrak{L})$ for the set of atoms of $L$. Two abstract interpretations are *equivalent* if and only if any concrete property is approximated in the same way in both interpretations. More formally, if $P^\flat(\preceq^\flat) \xleftrightarrow[\alpha_1]{\gamma_1} {P^\sharp}_1({\preceq^\sharp}_1)$ and $P^\flat(\preceq^\flat) \xleftrightarrow[\alpha_2]{\gamma_2} {P^\sharp}_2({\preceq^\sharp}_2)$ then $\forall p^\flat \in P^\flat : \gamma_1 \circ \alpha_1(p^\flat) = \gamma_2 \circ \alpha_2(p^\flat)$.

*Proposition 18 (Representation of the down-set completion using atoms). Let $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ be a completely distributive complete lattice and $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ be an atomistic complete lattice such that $P^\flat(\preceq^\flat) \xleftrightarrow[\alpha]{\gamma} P^\sharp(\preceq^\sharp)$. Define:*

$$\begin{array}{rlrl} \gamma^{\mathfrak{a}} \in \wp(\mathfrak{A}(\mathfrak{P}^\sharp)) \mapsto \mathfrak{D}^{\preceq^\sharp}(P^\sharp) & & \alpha^{\mathfrak{a}} \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp) \mapsto \wp(\mathfrak{A}(\mathfrak{P}^\sharp)) \\ \gamma^{\mathfrak{a}}(X) \stackrel{\text{def}}{=} \{\bigvee^\sharp S \mid S \subseteq X\} & & \alpha^{\mathfrak{a}}(X) \stackrel{\text{def}}{=} X \cap \mathfrak{A}(\mathfrak{P}^\sharp) \end{array}$$

*Then $\mathfrak{D}^{\preceq^\sharp}(P^\sharp)(\subseteq) \xleftrightarrow[\alpha^{\mathfrak{a}}]{\gamma^{\mathfrak{a}}} \wp(\mathfrak{A}(\mathfrak{P}^\sharp))(\subseteq)$. If, moreover, $\gamma$ is* join atomistic*, that is to say:*

$$\forall X \subseteq P^\sharp : \bigvee^\flat \gamma^*(X) \;=\; \bigvee^\flat \gamma^* \left((\downarrow^{\preceq^\sharp} X) \cap \mathfrak{A}(\mathfrak{P}^\sharp)\right)$$

*then the two abstract interpretations are equivalent in that $\gamma^{\mathfrak{d}} \circ \alpha^{\mathfrak{d}} = \gamma^{\mathfrak{d}} \circ \gamma^{\mathfrak{a}} \circ \alpha^{\mathfrak{a}} \circ \alpha^{\mathfrak{d}}$.*

PROOF. If $X \subseteq \mathfrak{A}(\mathfrak{P}^\sharp)$, then for all $x \in X$ we have $\{x\} \subseteq X$ and $\vee^\sharp\{x\} = x$ proving that $X \subseteq \{\bigvee^\sharp S \mid S \subseteq X\} \cap \mathfrak{A}(\mathfrak{P}^\sharp) = \alpha^{\mathfrak{a}} \circ \gamma^{\mathfrak{a}}(X)$. Moreover, if $S \subseteq X$ and $\bigvee^\sharp S \in \mathfrak{A}(\mathfrak{P}^\sharp)$ then $S$ cannot be empty since the infimum $\vee^\sharp\emptyset = f^\sharp$ does not belong to $\mathfrak{A}(\mathfrak{P}^\sharp)$. Moreover, $S$ cannot contain two distinct atoms $x_1$ and $x_2$ since we would have $f^\sharp \prec^\sharp x_1 \preceq^\sharp (x_1 \vee^\sharp x_2) \preceq^\sharp \bigvee^\sharp S$ hence $x_1 = (x_1 \vee^\sharp x_2) = \bigvee^\sharp S$ since $x_1$ and $\bigvee^\sharp S$ are atoms and therefore the contradiction $f^\sharp \prec^\sharp x_1 \prec^\sharp x_2$ since $x_1$ and $x_2$ are distinct atoms. It follows that $\bigvee^\sharp S = \vee^\sharp\{x\} = x$ where $x \in X$ proving that $\{\bigvee^\sharp S \mid S \subseteq X\} \cap \mathfrak{A}(\mathfrak{P}^\sharp) \subseteq X$ hence by antisymmetry that $\alpha^{\mathfrak{a}} \circ \gamma^{\mathfrak{a}}$ is the identity.

Assume now that $X \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp)$ and $x \in X$. Let $S$ be $\{a \in \mathfrak{A}(\mathfrak{P}^\sharp) \mid a \preceq^\sharp x\}$. We have $S \subseteq X$ since $X$ is down closed and $x = \bigvee^\sharp S$ since $P^\sharp(\preceq^\sharp, \vee^\sharp)$ is an atomistic complete lattice proving that $X \subseteq \{\bigvee^\sharp S \mid S \subseteq X \cap \mathfrak{A}(\mathfrak{P}^\sharp)\} = \gamma^{\mathfrak{a}} \circ \alpha^{\mathfrak{a}}(X)$.

We conclude that $\mathfrak{D}^{\preceq^\sharp}(P^\sharp)(\subseteq) \xleftrightarrow[\alpha^{\mathfrak{a}}]{\gamma^{\mathfrak{a}}} \wp(\mathfrak{A}(\mathfrak{P}^\sharp))(\subseteq)$.

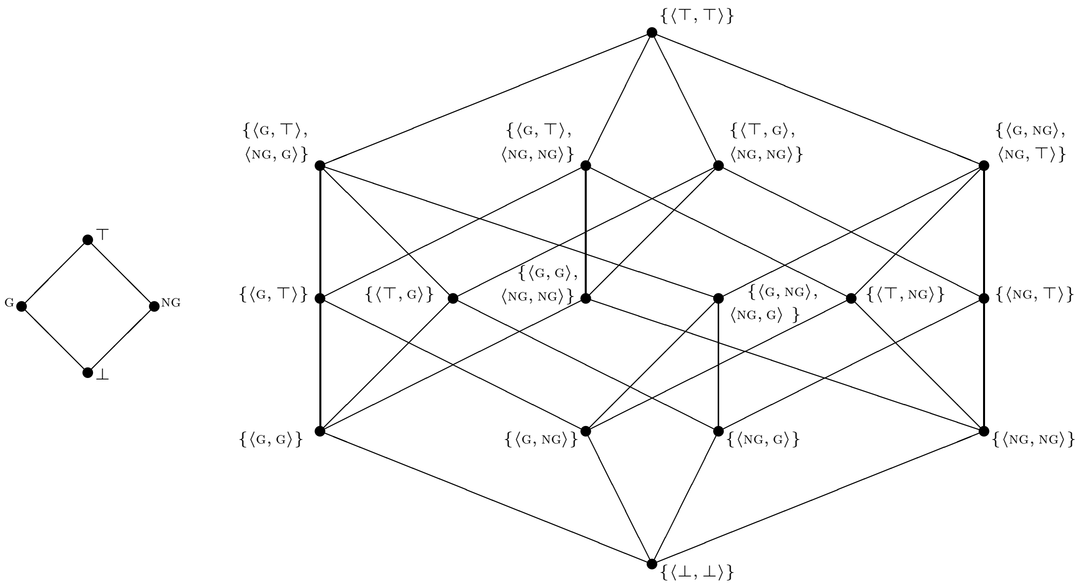
Finally, if $X \in \mathfrak{D}^{\preceq^\sharp}(P^\sharp)$ then $\gamma^{\mathfrak{d}} \circ \gamma^{\mathfrak{a}} \circ \alpha^{\mathfrak{a}}(X) = \bigvee^\flat \gamma^*(\{\bigvee^\sharp S \mid S \subseteq X \cap \mathfrak{A}(\mathfrak{P}^\sharp)\}) = \bigvee^\flat \gamma^*(\{a \in \mathfrak{A}(\mathfrak{P}^\sharp) \mid \exists S \subseteq X \cap \mathfrak{A}(\mathfrak{P}^\sharp) : a \preceq^\sharp \bigvee^\sharp S\})$ since $\gamma$ is join-atomistic. Since $P^\sharp$ is atomistic, this is equal to $\bigvee^\flat \gamma^*(\{S \mid S \subseteq X \cap \mathfrak{A}(\mathfrak{P}^\sharp)\}) = \bigvee^\flat \gamma^*(X \cap \mathfrak{A}(\mathfrak{P}^\sharp)) = \bigvee^\flat \gamma^* \left((\downarrow^{\preceq^\sharp} X) \cap \mathfrak{A}(\mathfrak{P}^\sharp)\right)$ since $X$ is down closed, which is equal to $\bigvee^\flat \gamma^*(X)$ that is to $\gamma^{\mathfrak{d}}(X)$. □

4.2.3.5. TRANSFORMING AN ATTRIBUTE INDEPENDENT ANALYSIS INTO A RELATIONAL ANALYSIS. If we have obtained independent analyses ${P^\sharp}^i({\preceq^\sharp}^i), i \in \Delta$, then the down-set completion of their reduced product provides a relational analysis.

*Example 19 (Relational groundness analysis).* By considering the down-set completion of the reduced product for groundness given in example 14, one can express that $X^1$ is ground if and only if $X^2$ is not ground by the element $\{\langle \text{G}, \text{NG}\rangle, \langle \text{NG}, \text{G}\rangle\}$, as shown in Figure 6. □

However the lattice which is obtained can be very large. If we consider the down-set completion of the reduced product of $n$ rules of signs lattices shown in example 16 then the longest strictly increasing chain in the down-set completion has length $3^n + 1$ hence this lattice is very large, so the corresponding program analyses might be very expensive.

Various other forms of relational combinations can be considered. For example, the set of monotone maps in ${P^\sharp}_1 \mapsto {P^\sharp}_2$ is considered in section 10.2 of [15], so as to obtain relational properties the complexity of which is included between those of the reduced product and down-set completion of the reduced product. By restricting to complete join morphisms, that is




$P^{\sharp^i},\ i = 1, 2$ Down-set completion of the reduced product $P^\sharp = \prod_{i\in\{1,2\}} P^{\sharp^i}$

**FIGURE 6.** Lattice of relational groundness analysis.

equivalently to Galois connections between ${P^\sharp}_1$ and ${P^\sharp}_2$, one obtains, up to an isomorphism, the tensor product considered in [45]. However tensor products cannot represent all relations that can be specified by elements of the down-set completion of the reduced product.

4.2.3.6. Transforming a Relational Analysis into an Attribute Independent Analysis. As shown by [33], relational analyses can be very expensive. A radical method to reduce the analysis cost is to transform the relational analysis into an attribute independent analysis. We now explain a systematic way to do so. In order to formalize the notion of relational analysis, let us consider a set $\Delta$ of program attributes, the properties of each attribute $i \in \Delta$ being described by elements of a given set $P^{\sharp^i}$ of properties. A relational property $X = \{\vec{p^\sharp}_j \mid j \in J\} \in \wp(\prod_{i\in\Delta} P^{\sharp^i})$ represents the disjunction for $j \in J$ of the conjunction for $i \in \Delta$ of the meanings of $\vec{p^\sharp}_j[i]$: $\gamma(X) = \bigvee_{j\in J} \bigwedge_{i\in\Delta} \gamma^i_j(\vec{p^\sharp}_j[i])$. Such a relational property can be approximated by a vector of attribute independent properties, as follows:

*Proposition 20. Let $P^{\sharp^i}(\preceq^{\sharp^i}, f^{\sharp^i}, t^{\sharp^i}, \wedge^{\sharp^i}, \vee^{\sharp^i})$ be complete lattices for $i \in \Delta$. Define:*

$$\alpha^{\mathrm{i}} \in \wp(\prod_{i\in\Delta} P^{\sharp^i}) \mapsto \prod_{i\in\Delta} P^{\sharp^i} \qquad\qquad \gamma^{\mathrm{i}} \in \prod_{i\in\Delta} P^{\sharp^i} \mapsto \wp(\prod_{i\in\Delta} P^{\sharp^i})$$

$$\alpha^{\mathrm{i}}(X) = \prod_{i\in\Delta} \vee^{\sharp^i}\{\vec{p^\sharp}[i] \mid \vec{p^\sharp} \in X\} \qquad\qquad \gamma^{\mathrm{i}}(\vec{p^\sharp}) = \{\vec{p^\sharp}\}$$

*Then $\wp(\prod_{i\in\Delta} P^{\sharp^i})(\vec{\precsim}) \underset{\alpha^{\mathrm{i}}}{\overset{\gamma^{\mathrm{i}}}{\leftrightarrows}} \prod_{i\in\Delta} P^{\sharp^i}(\vec{\preceq}^\sharp)$, where $X \mathrel{\vec{\precsim}} X' \stackrel{\text{def}}{=} \forall \vec{p^\sharp} \in X : \exists \vec{p^\sharp}' \in X' : \vec{p^\sharp} \mathrel{\vec{\preceq}^\sharp} \vec{p^\sharp}'$ and $\vec{p^\sharp} \mathrel{\vec{\preceq}^\sharp} \vec{p^\sharp}' \stackrel{\text{def}}{=} \forall i \in \Delta : \vec{p^\sharp}[i] \preceq^{\sharp^i} \vec{p^\sharp}'[i]$.*

PROOF. $\alpha^{\mathrm{i}}(X) \mathrel{\vec{\preceq}^\sharp} \vec{p^\sharp} \Leftrightarrow \prod_{i\in\Delta} \vee^{\sharp^i}\{\vec{p^\sharp}'[i] \mid \vec{p^\sharp}' \in X\} \mathrel{\vec{\preceq}^\sharp} \vec{p^\sharp} \Leftrightarrow \forall i \in \Delta : \forall \vec{p^\sharp}' \in X : \vec{p^\sharp}'[i] \preceq^{\sharp^i} \vec{p^\sharp}[i] \Leftrightarrow \forall \vec{p^\sharp}' \in X : \forall i \in \Delta : \vec{p^\sharp}'[i] \preceq^{\sharp^i} \vec{p^\sharp}[i] \Leftrightarrow \forall \vec{p^\sharp}' \in X : \vec{p^\sharp}' \mathrel{\vec{\preceq}^\sharp} \vec{p^\sharp} \Leftrightarrow X \mathrel{\vec{\precsim}} \{\vec{p^\sharp}\} \Leftrightarrow X \mathrel{\vec{\precsim}} \gamma^{\mathrm{i}}(\vec{p^\sharp})$. $\square$

In practice the attribute independent analysis is often not precise enough whilst the relational one is too expensive. The idea is then to consider some but not all relationships between attributes. Doing so a priori without knowing at all the program to be analysed, i.e., using the Galois connection approach, is then almost impossible. A better approach is to take decisions progressively during the analysis, as the relationships holding between attributes are discovered. This is the widening/narrowing approach discussed below. There a criterion is given to throw away the relationships considered uninteresting with respect to what is presently known about the program properties.

4.2.3.7. Lifting to Property Transformers. As observed in paragraph 7.1 of [15], Galois connections can be lifted from sets of properties to sets of monotone properties transformers:

*Proposition 21. If* $P^\flat(\preceq^\flat) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, *then*

$$P^\flat \overset{m}{\longmapsto} P^\flat(\preceq^\flat) \overset{\lambda\,\phi\bullet\gamma\circ\phi\circ\alpha}{\underset{\lambda\,\varphi\bullet\alpha\circ\varphi\circ\gamma}{\leftrightarrows}} P^\sharp \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)$$

*where the ordering on functions is pointwise that is* $\varphi \preceq \phi$ *if and only if* $\forall x : \varphi(x) \preceq \phi(x)$.

PROOF. If $\alpha \circ \varphi \circ \gamma \preceq^\sharp \phi$, then for all $x$ in $P^\sharp$, we have $\alpha \circ \varphi \circ \gamma(x) \preceq^\sharp \phi(x)$ by definition of pointwise orderings whence $\varphi \circ \gamma(x) \preceq^\flat \gamma \circ \phi(x)$ by (1). In particular when $x = \alpha(p^\flat)$ for any $p^\flat \in P^\flat$, we have $\varphi \circ \gamma \circ \alpha(p^\flat) \preceq^\flat \gamma \circ \phi \circ \alpha(p^\flat)$. But $p^\flat \preceq^\flat \gamma \circ \alpha(p^\flat)$ by (2) so that by monotony of $\varphi$ for $\preceq^\flat$ we have $\varphi(p^\flat) \preceq^\flat \varphi \circ \gamma \circ \alpha(p^\flat)$ proving by transitivity and definition of pointwise orderings that $\varphi \preceq^\flat \gamma \circ \phi \circ \alpha$. Reciprocally, if $\varphi \preceq^\flat \gamma \circ \phi \circ \alpha$ then $\forall x \in P^\flat : \varphi(x) \preceq^\flat \gamma \circ \phi \circ \alpha(x)$ whence $\alpha \circ \varphi \circ \gamma(p^\sharp) \preceq^\sharp \phi \circ \alpha \circ \gamma(p^\sharp)$ by (1) for $x = \gamma(p^\sharp)$. Moreover $\phi \circ \alpha \circ \gamma(p^\sharp) \preceq^\sharp \phi(p^\sharp)$ by (3) and monotony of $\phi$ for $\preceq^\sharp$. By transitivity and definition of pointwise ordering we conclude that $\alpha \circ \varphi \circ \gamma \preceq^\sharp \phi$. $\square$

For example, starting from an approximation of values, the repeated application of this property can be used to approximate functions, functionals, etc. In particular, it follows that the choice of an approximation of program properties uniquely determines the way of approximating fixpoints of properties transformers. This result is also the basis for extending abstract interpretation from first-order to higher-order functional languages.

### *4.2.4. Approximation of a Concrete Program Fixpoint Semantics by an Abstract Semantics*

We assume that the concrete semantics is defined as a least fixpoint $\mathit{lfp}\, F^\flat = \bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\bot^\flat)$ where $X = F^\flat(X)$ is the equation (or system of equations) associated to the program, $P^\flat(\sqsubseteq^\flat, \bot^\flat, \bigsqcup^\flat)$ is a poset of concrete program properties and $F^\flat \in P^\flat(\sqsubseteq^\flat) \overset{c}{\longmapsto} P^\flat(\sqsubseteq^\flat)$ is continuous. We assume that the least upper bound of the ${F^\flat}^n(\bot^\flat)$, $n \geq 0$ exists, for example because $P^\flat(\sqsubseteq^\flat,\bot^\flat,\bigsqcup^\flat)$ is a strict cpo.

*Example 22 (Semantics of logic programs).* Let `P` be a logic program (containing at least one constant), $U_{\mathtt{P}}$ be its Herbrand universe and *ground*(`P`) be the set of all ground instances of clauses in `P`. The poset $P^\flat(\sqsubseteq^\flat,\bot^\flat,\bigsqcup^\flat)$ of concrete properties is the complete lattice $\wp(U_{\mathtt{P}})(\subseteq, \emptyset, U_{\mathtt{P}}, \cap, \cup)$. $F^\flat$ is the immediate consequence operator $T_{\mathtt{P}}$ of van Emden and Kowalski [54] defined by:

$$\begin{array}{rcl} T_{\mathtt{P}} & \in & \wp(U_{\mathtt{P}}) \mapsto \wp(U_{\mathtt{P}}) \\ T_{\mathtt{P}}(X) & = & \{\mathtt{a}_0 \mid \mathtt{a}_0\text{->}\mathtt{a}_1, \ldots, \mathtt{a}_n \in \mathit{ground}(\mathtt{P}) \wedge \forall i \in [1, n] : a_i \in X\} \end{array} \tag{12}$$

Observe that $T_{\mathtt{P}}$ is a complete $\cup$-morphism. A postfixpoint $I \in {T_{\mathtt{P}}}^{\subseteq}$ of $T_{\mathtt{P}}$ is a model of `P`. The application of Tarski's fixpoint theorem [52] yields van Emden and Kowalski characterization theorem of the semantics of the logic program `P`, which is the least model of `P`, that is $\mathit{lfp}\, T_{\mathtt{P}} = \bigcup_{n\geq 0} {T_{\mathtt{P}}}^n(\emptyset)$ with ${T_{\mathtt{P}}}^n(\emptyset) \subseteq {T_{\mathtt{P}}}^{n+1}(\emptyset)$ for all $n > 0$. $\square$

Observe that two partial orderings are involved on $P^\flat$ (and $P^\sharp$). In general these orderings are distinct but they may coincide. $\sqsubseteq^\flat$ is called the *computational ordering*. It holds between iterates ${F^\flat}^n(\bot^\flat)$ during the fixpoint computation. $\preceq^\flat$ is called the *approximation ordering*. It specifies the relative precision of concrete program properties.

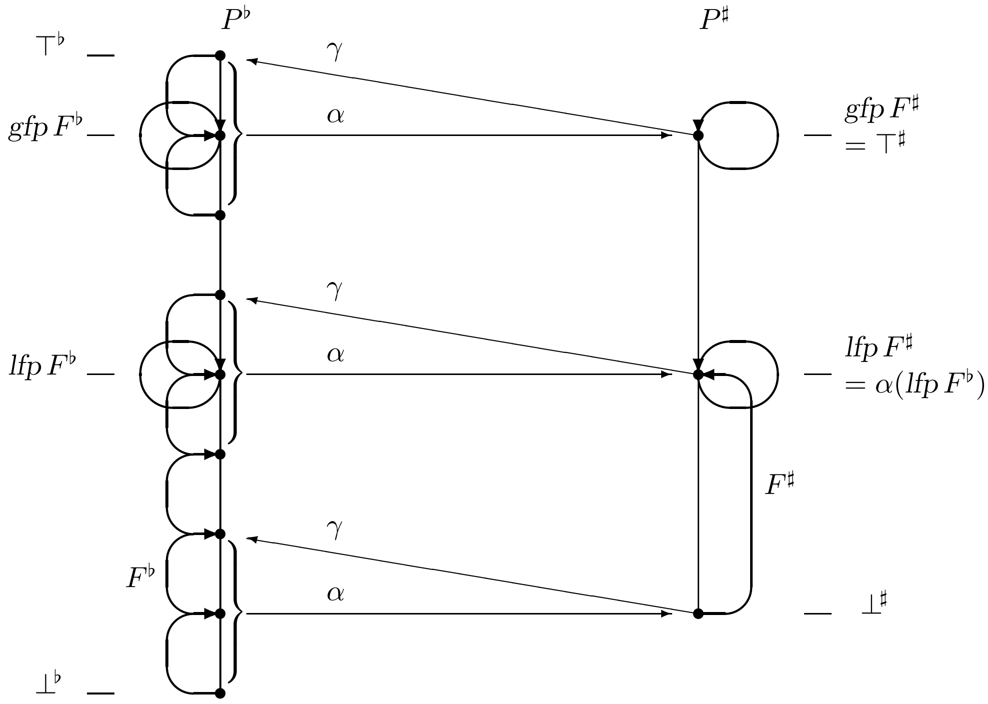



**FIGURE 7.** Fixpoint inducing using a Galois connection.

Let us now examine the problem of computing and then approximating from above for $\preceq^\sharp$ the abstract semantics $\alpha(lfp\,F^\flat)$ of the program.

4.2.4.1. Fixpoint Inducing Using Galois Connections. Assume that $P^\flat(\sqsubseteq^\flat, \bigsqcup^\flat)$ and $P^\sharp(\sqsubseteq^\sharp, \bigsqcup^\sharp)$ are posets, that $F^\flat \in P^\flat \mapsto P^\flat$ provides the concrete semantics $lfp\,F^\flat$ of a program and we are interested in its abstraction $\alpha(lfp\,F^\flat)$ where $P^\flat(\sqsubseteq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\sqsubseteq^\sharp)$. Assuming that $lfp\,F^\flat$ is obtained as the limit of the iteration sequence ${F^\flat}^0(\perp^\flat) = \perp^\flat$, ${F^\flat}^1(\perp^\flat) = F^\flat(\perp^\flat)$, $\ldots$ , ${F^\flat}^{n+1}(\perp^\flat) = F^\flat({F^\flat}^n(\perp^\flat))$, $\ldots$ , ${F^\flat}^\omega = \bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\perp^\flat) = lfp\,F^\flat$, it is natural to try to obtain $\alpha(lfp\,F^\flat)$ by computing the abstract image of this iteration sequence that is: $\alpha({F^\flat}^0(\perp^\flat)) = \alpha(\perp^\flat)$, $\alpha({F^\flat}^1(\perp^\flat)) = \alpha(F^\flat(\perp^\flat))$, $\ldots$ , $\alpha({F^\flat}^{n+1}(\perp^\flat)) = \alpha(F^\flat({F^\flat}^n(\perp^\flat)))$, $\ldots$ , $\alpha({F^\flat}^\omega) = \alpha(\bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\perp^\flat)) = \alpha(lfp\,F^\flat)$. Since, in general, the computation must be done entirely in the set $P^\sharp$ of abstract program properties, we would like to obtain this iteration sequence using an abstract infimum $\perp^\sharp$, an abstract operator $F^\sharp$ and an abstract least upper bound $\sqcup^\sharp$ on $P^\sharp$ in the form ${F^\sharp}^0(\perp^\sharp) = \perp^\sharp$, ${F^\sharp}^1(\perp^\sharp) = F^\sharp(\perp^\sharp)$, $\ldots$ , ${F^\sharp}^{n+1}(\perp^\sharp) = F^\sharp({F^\sharp}^n(\perp^\sharp))$, $\ldots$ , ${F^\sharp}^\omega = \bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\perp^\sharp)$. This is possible if ${F^\sharp}^n(\perp^\sharp) = \alpha({F^\flat}^n(\perp^\flat))$ for all $n = 0, 1, \ldots, \omega$. The situation is illustrated in Figure 7. When looking for hypotheses implying the desired property ${F^\sharp}^n(\perp^\sharp) = \alpha({F^\flat}^n(\perp^\flat))$ for all $n = 0, 1, \ldots, \omega$, it is interesting to favor inductive reasonings on $n$, as follows:

— For $n = 0$, $\alpha(\perp^\flat) = \perp^\sharp$ which yields the definition of $\perp^\sharp$;

— For all $n \geq 0$, $\alpha({F^\flat}^n(\perp^\flat)) = {F^\sharp}^n(\perp^\sharp)$ implies $\alpha({F^\flat}^{n+1}(\perp^\flat)) = {F^\sharp}^{n+1}(\perp^\sharp)$ that is by definition of the iteration sequences $\alpha(F^\flat({F^\flat}^n(\perp^\flat))) = F^\sharp({F^\sharp}^n(\perp^\sharp))$ that is using the induction hypothesis $\alpha(F^\flat({F^\flat}^n(\perp^\flat))) = F^\sharp(\alpha({F^\flat}^n(\perp^\flat)))$ which obviously holds when $\forall p^\flat \in P^\flat : \alpha(F^\flat(p^\flat)) = F^\sharp(\alpha(p^\flat))$ or $F^\sharp = \alpha \circ F^\flat \circ \gamma$ and $\forall p^\flat \in P^\flat : \gamma \circ \alpha(p^\flat) = p^\flat$;

— Lastly for $n = \omega$, we must have $\forall n \geq 0 : \alpha({F^\flat}^n(\perp^\flat)) = {F^\sharp}^n(\perp^\sharp)$ which implies $\alpha\left(\bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\perp^\flat)\right) = \bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\perp^\sharp)$. But this is true since $\bigsqcup^\sharp_{n\geq 0} \alpha({F^\flat}^n(\perp^\flat)) = \bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\perp^\sharp)$ by induction hypothesis and $\bigsqcup^\sharp_{n\geq 0} \alpha({F^\flat}^n(\perp^\flat)) = \alpha\left(\bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\perp^\flat)\right)$ since, by proposition 6, $\alpha$ is a complete join morphism, whence we conclude by transitivity.

Moreover if $p^\flat$ is a fixpoint of $F^\flat$ then $F^\flat(p^\flat) = p^\flat$ whence $\alpha(F^\flat(p^\flat)) = \alpha(p^\flat)$ and therefore $F^\sharp(\alpha(p^\flat)) = \alpha(p^\flat)$ since $\alpha \circ F^\flat = F^\sharp \circ \alpha$ proving that $\alpha(p^\flat)$ is a fixpoint of $F^\sharp$. In particular $\alpha(\bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\bot^\flat)) = \bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\bot^\sharp)$ is a fixpoint of $F^\sharp$.

Assume that $F^\sharp \in P^\sharp(\sqsubseteq^\sharp) \overset{m}{\longmapsto} P^\sharp(\sqsubseteq^\sharp)$ is monotone and $p^\sharp$ is a fixpoint of $F^\sharp$ such that $\bot^\sharp \sqsubseteq^\sharp p^\sharp$. Then ${F^\sharp}^0(\bot^\sharp) = \bot^\sharp \sqsubseteq^\sharp p^\sharp$. If ${F^\sharp}^n(\bot^\sharp) \sqsubseteq^\sharp p^\sharp$ then ${F^\sharp}^{n+1}(\bot^\sharp) = F^\sharp({F^\sharp}^n(\bot^\sharp)) \sqsubseteq^\sharp F^\sharp(p^\sharp) = p^\sharp$ by monotony and fixpoint property. If $\forall n \geq 0 : {F^\sharp}^n(\bot^\sharp) \sqsubseteq^\sharp p^\sharp$ then $\bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\bot^\sharp) \sqsubseteq^\sharp p^\sharp$ by definition of least upper bounds proving that $\bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\bot^\sharp)$ is the least fixpoint of $F^\sharp$ greater than or equal to $\bot^\sharp$.

In summary, we have just proved the following proposition ([15], theorem 7.1.0.4–(3)):

*Proposition 23 (Fixpoint inducing). If $P^\flat(\sqsubseteq^\flat, \bigsqcup^\flat)$ and $P^\sharp(\sqsubseteq^\sharp, \bigsqcup^\sharp)$ are posets, $P^\flat(\sqsubseteq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\sqsubseteq^\sharp)$, $F^\flat \in P^\flat \mapsto P^\sharp$ is such that $\mathit{lfp}\, F^\flat = \bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\bot^\flat)$, $\bot^\sharp = \alpha(\bot^\flat)$, $F^\sharp \in P^\sharp \mapsto P^\sharp$ is such that $\alpha \circ F^\flat = F^\sharp \circ \alpha$ (which is implied by $F^\sharp = \alpha \circ F^\flat \circ \gamma$ and $\forall p^\flat \in P^\flat : \gamma \circ \alpha(p^\flat) = p^\flat$), then $\alpha(\mathit{lfp}\, F^\flat) = \bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\bot^\sharp)$. $\bigsqcup^\sharp_{n\geq 0} {F^\sharp}^n(\bot^\sharp)$ is a fixpoint of $F^\sharp$. When $F^\sharp \in P^\sharp(\sqsubseteq^\sharp) \overset{m}{\longmapsto} P^\sharp(\sqsubseteq^\sharp)$ is monotone, it is the least fixpoint of $F^\sharp$ greater than or equal to $\bot^\sharp$.*

The fact that $\mathit{lfp}\, F^\flat = \bigsqcup^\flat_{n\geq 0} {F^\flat}^n(\bot^\flat)$ follows for example from $\omega$-upper-continuity on a complete partial order. It can be relaxed into monotony by using transfinite iteration sequences, as in [14]. Observe that no such hypothesis is necessary on the abstract domain $P^\sharp$ since it is induced from $P^\flat$ by the Galois connection.

4.2.4.2. Fixpoint Abstraction Using Galois Connections. In general, a fixpoint inducing is not computable so that one must be satisfied with an abstract approximation $p^\sharp$ from above of the concrete fixpoint $\alpha(\mathit{lfp}\, F^\flat)$ that is $\alpha(\mathit{lfp}\, F^\flat) \preceq^\sharp p^\sharp$ or equivalently $\mathit{lfp}\, F^\flat \preceq^\flat \gamma(p^\sharp)$. When a Galois connection has been established between concrete and abstract properties, any concrete fixpoint can be approximated by an abstract fixpoint using an approximation of the function as indicated in proposition 21. We obtain theorem 7.1.0.4 of [15]:

*Proposition 24 (Fixpoint abstraction). If $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ and $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ are complete lattices, $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$ and $F^\flat \in P^\flat(\preceq^\flat) \overset{m}{\longmapsto} P^\flat(\preceq^\flat)$, then $\alpha(\mathit{lfp}\, F^\flat) \preceq^\sharp \mathit{lfp}\, \alpha \circ F^\flat \circ \gamma$.*

PROOF. By Tarski's fixpoint theorem [52], the least fixpoints exist. So let $p^\sharp = \mathit{lfp}\, \alpha \circ F^\flat \circ \gamma$. We have $\alpha \circ F^\flat \circ \gamma(p^\sharp) = p^\sharp$ whence $F^\flat \circ \gamma(p^\sharp) \preceq^\flat \gamma(p^\sharp)$ by (1). It follows that $\gamma(p^\sharp)$ is a postfixpoint of $F^\flat$ whence $\mathit{lfp}\, F^\flat \preceq^\flat \gamma(p^\sharp)$ by Tarski's fixpoint theorem or equivalently $\alpha(\mathit{lfp}\, F^\flat) \preceq^\sharp p^\sharp = \mathit{lfp}\, \alpha \circ F^\flat \circ \gamma$. □

A consequence of this theorem is that the choice of the concrete semantics $F^\flat$ and of the abstraction $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$ of program properties entirely determines the abstract semantics $\mathit{lfp}\, \alpha \circ F^\flat \circ \gamma$. It follows that the abstract semantics can be constructively derived from the concrete semantics by a formal computation consisting in simplifying $\alpha \circ F^\flat \circ \gamma$ so as to express it using operators on abstract properties only (this has been illustrated on the casting out of nine introductory example). But for a few exceptions (such as [5] where $P^\flat$ is finite), this simplification is not mechanizable and must be done by hand. This simplification is facilitated by the observation that $\alpha \circ F^\flat \circ \gamma$ can be approximated from above by $F^\sharp$ such that $\forall p^\sharp \in P^\sharp : \alpha \circ F^\flat \circ \gamma(p^\sharp) \preceq^\sharp F^\sharp(p^\sharp)$:

*Proposition 25 (Fixpoint approximation). If $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ is a complete lattice, $F^\sharp$, $\bar{F}^\sharp \in P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)$, and $F^\sharp \preceq^\sharp \bar{F}^\sharp$, then $\mathit{lfp}\, F^\sharp \preceq^\sharp \mathit{lfp}\, \bar{F}^\sharp$.*

PROOF. We have $F^\sharp(\mathit{lfp}\, \bar{F}^\sharp) \preceq^\sharp \bar{F}^\sharp(\mathit{lfp}\, \bar{F}^\sharp) = \mathit{lfp}\, \bar{F}^\sharp$ whence $\mathit{lfp}\, F^\sharp \preceq^\sharp \mathit{lfp}\, \bar{F}^\sharp$ since $\mathit{lfp}\, F^\sharp = \bigwedge^\sharp\{X \mid F^\sharp(X) \preceq^\sharp X\}$ by Tarski's fixpoint theorem [52]. □

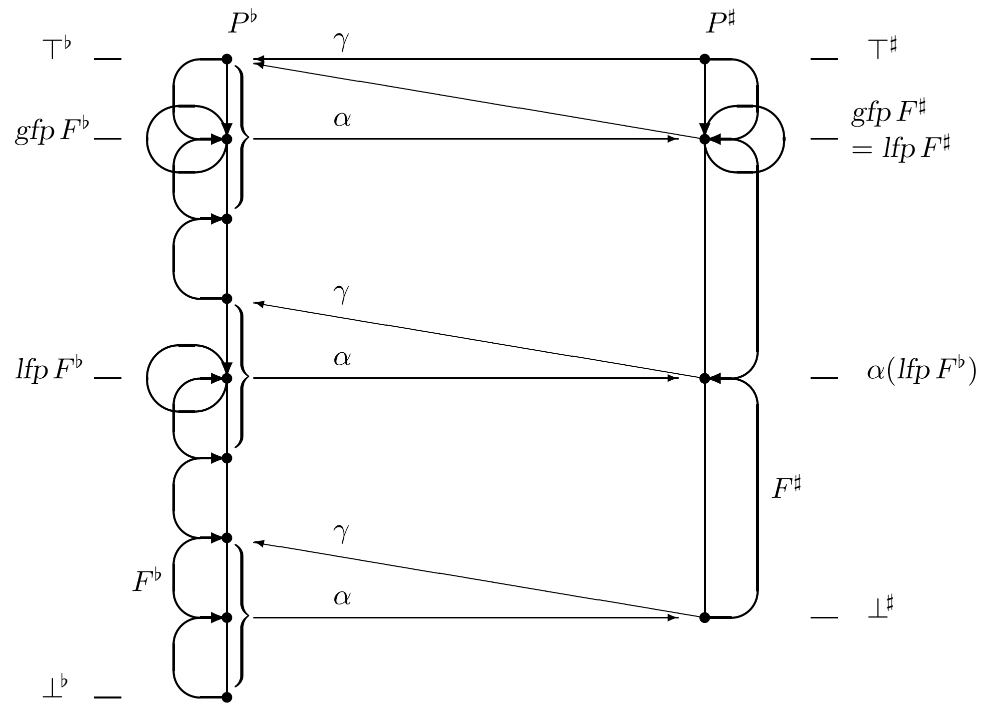



**FIGURE 8.** Fixpoint approximation using a Galois connection.

Apart from this, theorem 24 has numerous variants depending upon the hypotheses ensuring the existence of fixpoint (for example see theorem 7.1.0.5 of [15] which avoids the monotony hypothesis). The general idea is that the abstract iterates approximate from above the concrete iterates, as illustrated in Figure 8. We will use the following variant of proposition 24 which is based upon the ideas sketched above, where the computational and approximation orderings are distinguished:

*Proposition 26 (Fixpoint abstract approximation). If $P^\flat(\sqsubseteq^\flat, \perp^\flat, \sqcup^\flat)$ and $P^\sharp(\sqsubseteq^\sharp, \perp^\sharp, \sqcup^\sharp)$ are cpos, $F^\flat \in P^\flat(\sqsubseteq^\flat, \sqcup^\flat) \xmapsto{c} P^\flat(\sqsubseteq^\flat, \sqcup^\flat)$, $F^\flat \in P^\flat(\preceq^\flat) \xmapsto{m} P^\flat(\preceq^\flat)$, $F^\sharp \in P^\sharp(\sqsubseteq^\sharp, \sqcup^\sharp) \xmapsto{c} P^\sharp(\sqsubseteq^\sharp, \sqcup^\sharp)$, $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, $\alpha(\perp^\flat) \preceq^\sharp \perp^\sharp$, $\forall p^\sharp \in P^\sharp : \alpha \circ F^\flat \circ \gamma(p^\sharp) \preceq^\sharp F^\sharp(p^\sharp)$ and for any $\sqsubseteq^\flat$-increasing chain ${p^\flat}_i, i \in \mathbb{N}$ and any $\sqsubseteq^\sharp$-increasing chain ${p^\sharp}_i, i \in \mathbb{N}$ such that $\forall i \in \mathbb{N} : \alpha({p^\flat}_i) \preceq^\sharp {p^\sharp}_i$ we have $\alpha({\bigsqcup^\flat}_{i\in\mathbb{N}}\, {p^\flat}_i) \preceq^\sharp {\bigsqcup^\sharp}_{i\in\mathbb{N}}\, {p^\sharp}_i$, then $\alpha(lfp\, F^\flat) \preceq^\sharp lfp\, F^\sharp$.*

PROOF. Since $\alpha(\perp^\flat) \preceq^\sharp \perp^\sharp$, we have $\alpha({F^\flat}^0(\perp^\flat)) \preceq^\sharp {F^\sharp}^0(\perp^\sharp)$. If $n \geq 0$ and $\alpha({F^\flat}^n(\perp^\flat)) \preceq^\sharp {F^\sharp}^n(\perp^\sharp)$ by induction hypothesis, then ${F^\flat}^n(\perp^\flat) \preceq^\flat \gamma({F^\sharp}^n(\perp^\sharp))$ by (1) so that by monotony $\alpha \circ F^\flat \circ {F^\flat}^n(\perp^\flat) \preceq^\sharp \alpha \circ F^\flat \circ \gamma({F^\sharp}^n(\perp^\sharp)) \preceq^\sharp F^\sharp({F^\sharp}^n(\perp^\sharp))$ whence $\alpha({F^\flat}^{n+1}(\perp^\flat)) \preceq^\sharp {F^\sharp}^{n+1}(\perp^\sharp)$. We conclude by continuity and the last hypothesis that $\alpha(lfp\, F^\flat) = \alpha({\bigsqcup^\flat}_{i\in\mathbb{N}}\, {F^\flat}^n(\perp^\flat)) \preceq^\sharp {\bigsqcup^\sharp}_{i\in\mathbb{N}}\, {F^\sharp}^n(\perp^\sharp) = lfp\, F^\sharp$. $\square$

When the computational and approximation orderings coincide, this proposition amounts to the more simple:

*Proposition 27. If $P^\flat(\preceq^\flat, f^\flat, \vee^\flat)$ and $P^\sharp(\preceq^\sharp, f^\sharp, \vee^\sharp)$ are cpos, $F^\flat \in P^\flat(\preceq^\flat, \vee^\flat) \xmapsto{c} P^\flat(\preceq^\flat, \vee^\flat)$, $F^\sharp \in P^\sharp(\preceq^\sharp, \vee^\sharp) \xmapsto{c} P^\sharp(\preceq^\sharp, \vee^\sharp)$, $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, $\alpha(\perp^\flat) \preceq^\sharp \perp^\sharp$ and $\forall p^\sharp \in P^\sharp : \alpha \circ F^\flat \circ \gamma(p^\sharp) \preceq^\sharp F^\sharp(p^\sharp)$, then $\alpha(lfp\, F^\flat) \preceq^\sharp lfp\, F^\sharp$.*

PROOF. This is a corollary of proposition 26 since for any $\preceq^\flat$-increasing chain ${p^\flat}_i, i \in \mathbb{N}$ and any $\preceq^\sharp$-increasing chain ${p^\sharp}_i, i \in \mathbb{N}$ such that $\forall i \in \mathbb{N} : \alpha({p^\flat}_i) \preceq^\sharp {p^\sharp}_i$, we have $\alpha({\vee^\flat}_{i\in\mathbb{N}}\, {p^\flat}_i) = {\vee^\sharp}_{i\in\mathbb{N}}\, \alpha({p^\flat}_i) \preceq^\sharp {\vee^\sharp}_{i\in\mathbb{N}}\, {p^\sharp}_i$ by proposition 6 and definition of least upper bounds. $\square$

As usual continuity hypotheses can be avoided using monotony and transfinite iteration sequences.

4.2.4.3. Chaotic and Asynchronous Iterations. Using a decomposition by partitioning, a 'concrete' fixpoint equation $X = F^\flat(X)$ can be decomposed into an 'abstract' system of equations:

$$\begin{cases} X_i = F^\sharp{}_i(X_1, X_2, \ldots, X_n) \\ i = 1, \ldots, n \end{cases} \tag{13}$$

where each $X_i$ belongs to a cpo or complete lattice $P^\sharp{}_i(\sqsubseteq^\sharp{}_i)$ and $F^\sharp{}_i(X_1, X_2, \ldots, X_n)$ is equal to the $i$-th component $F^\sharp(X)[i]$ of $F^\sharp(X)$. If $F^\sharp$ is upper-continuous then the least fixpoint $\mathit{lfp}\, F^\sharp = \bigsqcup^\sharp{}_{k\geq 0} F^{\sharp k}$ where $F^{\sharp 0} = \perp^\sharp$ and $F^{\sharp k+1} = F^\sharp(F^{\sharp k})$ can be computed by Jacobi's method of successive approximations, which can be detailed as:

$$\begin{cases} X_i^{k+1} = F^\sharp{}_i(X_1^k, X_2^k, \ldots, X_n^k) \\ i = 1, \ldots, n \end{cases} \tag{14}$$

In practice the Gauss-Seidel's iterative method:

$$\begin{cases} X_1^{k+1} & = & F^\sharp{}_1(X_1^k, & X_2^k, & \ldots, & X_{i-1}^k, & X_i^k, & \ldots, & X_{n-1}^k, & X_n^k) \\ \ldots & & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ X_i^{k+1} & = & F^\sharp{}_i(X_1^{k+1}, & X_2^{k+1}, & \ldots, & X_{i-1}^{k+1}, & X_i^k, & \ldots, & X_{n-1}^k, & X_n^k) \\ \ldots & & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ X_n^{k+1} & = & F^\sharp{}_n(X_1^{k+1}, & X_2^{k+1}, & \ldots, & X_{i-1}^{k+1}, & X_i^{k+1}, & \ldots, & X_{n-1}^{k+1}, & X_n^k) \end{cases} \tag{15}$$

which consists in continually reinjecting in the computations the last results of the computations themselves would reduce the memory congestion and accelerate the convergence.

In general, Gauss-Seidel's method is not algorithmically more reliable than Jacobi's successive approximations method. This means that without sufficient hypotheses on $F^\sharp$, Jacobi's method may converge although the Gauss-Seidel one diverges. The contrary is also possible, that is Gauss-Seidel's method may converge although Jacobi's iterations endless cycle. Fortunately, this phenomenon is impossible when $F^\sharp$ is upper-continuous (or monotone using transfinite iteration sequences). One can arbitrarily determine at each step which are the components of the system of equations which will evolve and in what order, as long as no component is forgotten indefinitely. Otherwise stated any *chaotic iteration method* converges to the least fixpoint of $F^\sharp$. We now define the notion of chaotic iterations more formally and prove convergence.

Let $J$ be a subset of $\{1, \ldots, n\}$. We denote by $F^\sharp{}_J$ the map defined by $F^\sharp{}_J(X_1, \ldots, X_n) = \langle Y_1, \ldots, Y_n\rangle$ where, for all $i = 1, \ldots, n$, we have:

$$\begin{cases} Y_i = F^\sharp{}_i(X_1, \ldots, X_n) & \text{if } i \in J \\ Y_i = X_i & \text{if } i \notin J \end{cases}$$

In particular, $F^\sharp{}_{\{1,\ldots,n\}} = F^\sharp$.

An *ascending sequence of chaotic iterations* for $F^\sharp$ is a sequence $X^k, k \geq 0$ of vectors of $\prod_{i=1}^n P^\sharp{}_i$ starting from the infimum $X^0 = \prod_{i=1}^n \perp^\sharp{}_i$ and recursively defined for $k > 0$ by: $X^k = F^\sharp{}_{J_{k-1}}(X^{k-1})$ where $J_k, k \geq 0$ is a *weakly fair* sequence of subsets of $\{1, \ldots, n\}$, that is: $\forall k \geq 0 : \forall i \in \{1, \ldots, n\} : \exists \ell \geq 0 : i \in J_{k+\ell}$ (so that no component is forgotten indefinitely). For example, Jacobi's successive approximations are obtained by choosing $\forall k \geq 0 : J_k = \{1, \ldots, n\}$, whereas the choice $\forall k \geq 0 : J_k = \{(k \mod n) + 1\}$ corresponds to Gauss-Seidel's iterative method. In [11], we proved:

*Proposition 28 (Convergence of an ascending sequence of chaotic iterations). The limit $\bigsqcup^\sharp_{k\geq 0} X^k$ of any ascending sequence of chaotic iterations $X^k, k \geq 0$ for an upper-continuous map $F^\sharp \in \prod_{i=1}^n P^\sharp{}_i \mapsto \prod_{i=1}^n P^\sharp{}_i$ is the least fixpoint of $F^\sharp$ greater than or equal to $X^0$.*

PROOF. 1) Let us first remark that whenever $X \sqsubseteq^\sharp F^\sharp(X) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$, we have $\forall J \subseteq \{1, \ldots, n\}$, $X \sqsubseteq^\sharp F^\sharp{}_J(X) \sqsubseteq^\sharp F^\sharp(X) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$. Indeed for all $i = 1, \ldots, n$, $X_i \sqsubseteq^\sharp{}_i F^\sharp{}_i(X)$, therefore, if $i \in J$, then $X_i \sqsubseteq^\sharp{}_i F^\sharp{}_i(X) = F^\sharp(X)[i] = F^\sharp{}_J(X)[i]$, else $X_i \sqsubseteq^\sharp{}_i F^\sharp{}_J(X)[i] \sqsubseteq^\sharp{}_i F^\sharp{}_i(X)$.

2) Let us now prove that $\forall k \geq 0 : X^k \sqsubseteq^\sharp X^{k+1} \sqsubseteq^\sharp F^\sharp(X^k) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$. For the infimum $X^0 \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$ we have $X^0 \sqsubseteq^\sharp F^\sharp(X^0) \sqsubseteq^\sharp F^\sharp(\mathit{lfp}\, F^\sharp) = \mathit{lfp}\, F^\sharp$ by monotony and fixpoint property hence $X^0 \sqsubseteq^\sharp X^1 = F^\sharp{}_{J_0}(X^0) \sqsubseteq^\sharp F^\sharp(X^0) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$ by 1). For the induction step, let us assume that $X^{k-1} \sqsubseteq^\sharp X^k \sqsubseteq^\sharp F^\sharp(X^{k-1}) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$ for some $k > 0$. If $i \in J_{k-1}$, then $X_i^k = F^\sharp{}_i(X^{k-1}) \sqsubseteq^\sharp{}_i F^\sharp{}_i(X^k) \sqsubseteq^\sharp{}_i \mathit{lfp}\, F^\sharp[i]$ since $X^{k-1} \sqsubseteq^\sharp X^k \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$ and $F^\sharp{}_i$ is monotone. Otherwise, $i \notin J_{k-1}$, and $X_i^k = X_i^{k-1} \sqsubseteq^\sharp{}_i F^\sharp{}_i(X^{k-1}) \sqsubseteq^\sharp{}_i \mathit{lfp}\, F^\sharp[i]$ by induction hypothesis so that $X_i^k \sqsubseteq^\sharp{}_i F^\sharp{}_i(X^{k-1}) \sqsubseteq^\sharp{}_i F^\sharp{}_i(X^k) \sqsubseteq^\sharp{}_i \mathit{lfp}\, F^\sharp[i]$ by monotony. In both cases, we have $X_i^k \sqsubseteq^\sharp{}_i F^\sharp{}_i(X^k) \sqsubseteq^\sharp{}_i \mathit{lfp}\, F^\sharp[i]$ for all $i = 1, \ldots, n$, therefore $X^k \sqsubseteq^\sharp F^\sharp(X^k) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$, proving that $X^k \sqsubseteq^\sharp X^{k+1} = F^\sharp{}_{J_k}(X^k) \sqsubseteq^\sharp F^\sharp(X^k) \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$.

3) Let us now prove that $\forall k \geq 0 : \exists m \geq k : F^\sharp(X^k) \sqsubseteq^\sharp X^m$. If $i \in \{1, \ldots, n\}$ and $k \geq 0$ then, by weak fairness, there exists $\ell(i)$ such that $i \in J_{k+\ell(i)}$. It follows that $X^{k+\ell(i)+1}[i] = F^\sharp{}_i(X^{k+\ell(i)})$. By induction using 2), we have $X^k \sqsubseteq^\sharp X^{k+\ell(i)}$, so that, by monotony, $F^\sharp(X^k)[i] = F^\sharp{}_i(X^k) \sqsubseteq^\sharp{}_i X^{k+\ell(i)+1}[i] \sqsubseteq^\sharp{}_i X^m[i]$, where $m$ is the maximum of the $\ell(i)$ for $i = 1, \ldots, n$. Whence $F^\sharp(X^k) \sqsubseteq^\sharp X^m$.

4) Let $X^\omega$ be $\bigsqcup^\sharp_{k\geq 0} X^k$. By 2), definition of least upper bounds, and upper-continuity, $X^\omega \sqsubseteq^\sharp \bigsqcup^\sharp_{k\geq 0} F^\sharp(X^k) = F^\sharp(X^\omega)$. By 3), $\forall k \geq 0 : F^\sharp(X^k) \sqsubseteq^\sharp \bigsqcup^\sharp_{m\geq 0} X^m$, whence $F^\sharp(X^\omega) = \bigsqcup^\sharp_{k\geq 0} F^\sharp(X^k) \sqsubseteq^\sharp X^\omega$. By antisymmetry, $X^\omega$ is a fixpoint of $F^\sharp$. By 2), $X^\omega \sqsubseteq^\sharp \mathit{lfp}\, F^\sharp$, whence equality holds by unicity of the least fixpoint. □

Proposition 28 justifies the use of abstract interpreters in which the chaotic iteration strategy is chosen so as to mimic actual program executions. Examples of practical implementation of a particular strategy of chaotic iteration are given by [48], [39], [25], [47], [27], [60]. An example of an abstract interpreter written in a version of Prolog is given in [56].

This result has been generalized to *asynchronous iterations* [6] corresponding to a parallel implementation where $X$ is a shared array and each process $i$ reads the value $x_j$ of element $X[j]$ in any order for $j = 1, \ldots, n$, then computes $x'_i = F^\sharp{}_i(x_1, \ldots, x_n)$ and finally asynchronously writes this value $x'_i$ in shared memory $X[i]$. The relative speed of the processes is irrelevant provided execution is weakly fair. Another generalization in [13] concerns systems of functional fixpoint equations $f_i(\vec{X}_i) = F_i[f_1, \ldots, f_n](\vec{X}_i)$, $i = 1, \ldots, n$. When each $f_i(\vec{X}_i)$ needs to be known only for a subset $\phi_i$ of the domain $P_i$ of $\vec{X}_i$, it is necessary and sufficient to compute the value of $f_i(\vec{X}_i)$ for $\vec{X}_i$ belonging to a subset the domain of $X$ called the $\phi$-$F$-closure and such that $\phi_i \subseteq \phi$-$F$-closure$_i \subseteq P_i$. This technique, which was later popularized by Jones and Mycroft [34] under the name *minimal function graphs*, may be used as a basis for the tabulation method of [2], [35], [23], [57].

4.2.4.4. CONVERGENCE AND TERMINATION. Convergence to the least fixpoint $\mathit{lfp}\, F^\sharp$ is obtained in proposition 28 by taking the join $\bigsqcup^\sharp_{k\geq 0} X^k$ of infinitely many terms in the chaotic iteration sequence. In practice this can be avoided when using finite lattices, posets satisfying the *ascending chain condition*, and more generally in any case when the chaotic iteration sequence is increasing but not strictly (because of properties of $P^\sharp$ and/or $F^\sharp$) so that the fixpoint must be reached after finitely many steps. For example, we have:

*Proposition 29 (Termination of an ascending sequence of chaotic iterations). If the length of strictly increasing chains in $\prod_{i=1}^n P^\sharp{}_i$ is bounded by $\ell$ and the number of steps which are necessary for any component to evolve in chaotic iterations $X^k$, $k \geq 0$ for $F^\sharp$ is bounded by $m$, then $\mathit{lfp}\, F^\sharp = X^{\ell m}$.*

PROOF. Observe that by cases 1°) and 3°) in the proof of proposition 28, we have $X^k \sqsubseteq^\sharp F^\sharp(X^k) \sqsubseteq^\sharp X^{k+m}$ for all $k \geq 0$. Therefore if there exists $k \geq 0$, hence a least one, such that $X^k = X^{k+m}$ then $X^k$ is a fixpoint of $F^\sharp$, whence the least one and $X^{im}$, $0 \leq i \leq k$ is a strictly increasing chain so that $k \leq \ell$. When a fixpoint is reached, the chaotic iterations are stabilized so that $\mathit{lfp}\, F^\sharp = X^{\ell m}$. The remaining case $\forall k \geq 0 : X^k \sqsubset^\sharp X^{k+m}$ is impossible since it is in contradiction with the ascending chain condition. □

Other theoretical upper bounds on fixed point iterations have been given by [46], but these worst case analyses do not take into account the fact that $F^\sharp$ is not indifferent. In particular, proofs that these bounds are tight may lead to consider peculiar $F^\sharp$ not corresponding to any program at all! Pending average-case analyses, practical experiences such as [25], [39], [55], [53], [57] are very useful. Moreover, in practice, it is always possible to use extrapolation techniques such as widenings and narrowings considered below to speed-up convergence at the price of overshooting the least fixpoint.

4.2.4.5. On the Use of Galois Connections. Galois connections correspond to an ideal situation where the set $P^\sharp$ of abstract properties has been defined so that any concrete property has a best abstract upper approximation. Numerous practical abstract interpretations, such as [21] or the k-depth success pattern analysis of Sato and Tamaki [49], [41], which do not satisfy this condition can be easily handled by relaxing some of the hypotheses involved in the Galois connection approach [19].

### *4.3. Approximation of Fixpoint Semantics by Convergence Acceleration Using Widenings and Narrowings*

In [9], we introduced the idea of using widening and narrowing operators to accelerate convergence for fixpoint approximation from above (the dual case considered in [6] is also useful for some applications such as type inference [42] where a sound approximation is from below). This idea offered the possibility of considering infinite lattices not satisfying the ascending chain condition or of speeding up convergence in case of combinatorial explosion [31]. The larger the abstract domain $P^\sharp$ is, the more precise the analyses tend to be because less information is lost. For termination, widening and narrowing operators ensure that only a finite $P^\sharp[\![p]\!]$ subspace of $P^\sharp$ will be considered during analysis of any program $p$. A Galois connection upon that subspace $P^\sharp[\![p]\!]$ would not do when $P^\sharp[\![p]\!]$ is different for each program $p$ and the union of these subspaces $P^\sharp[\![p]\!]$ for all programs $p$ is infinite.

#### *4.3.1. Downward Abstract Iteration Sequence with Narrowing*

A first idea to effectively approximate $\mathit{lfp}\, F^\sharp$ from above is to use a downward iteration $\check{X}^k$, $k \geq 0$, all elements of which are upper approximations of the least fixpoint $\mathit{lfp}\, F^\sharp$ and which is stationary after finitely many steps. In order to ensure that all $\check{X}^k$, $k \geq 0$ are upper approximations of the least fixpoint $\mathit{lfp}\, F^\sharp$, one can look for an inductive argument using the basis $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^0$ and the inductive step $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^k \Rightarrow \mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^{k+1} \preceq^\sharp \check{X}^k$. The basis is easily handled with by starting from the supremum $\check{X}^0 = t^\sharp$. Finding general purpose sufficient conditions ensuring the validity of the inductive step $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^k$ implies $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^{k+1}$ together with $\check{X}^{k+1} \preceq^\sharp \check{X}^k$ is a bit more difficult since the only available information is $\check{X}^k$ and $F^\sharp(\check{X}^k)$ and the least fixpoint $\mathit{lfp}\, F^\sharp$ and more generally the fixpoints of $F^\sharp$ are unknown. Hence we define $\check{X}^{k+1}$ to be $\check{X}^k \bigtriangleup F^\sharp(\check{X}^k)$ that is the composition of the available information using a so called *narrowing operator* $\bigtriangleup$. To ensure $\check{X}^k \bigtriangleup F^\sharp(\check{X}^k) = \check{X}^{k+1} \preceq^\sharp \check{X}^k$ with the additional constraint that it must be valid for all program, that is all $F^\sharp$, we can require more specifically that $\forall x, y \in P^\sharp : x \bigtriangleup y \preceq^\sharp x$. Ensuring $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^k \bigtriangleup F^\sharp(\check{X}^k)$ for all $F^\sharp$, hence without knowing its fixpoints, is a bit more difficult. In practice however, $F^\sharp$ is often monotone for $\preceq^\sharp$. In this case if $p^\sharp$ is a fixpoint of $F^\sharp$ then $p^\sharp \preceq^\sharp x$ implies $p^\sharp \preceq^\sharp F^\sharp(x)$ by monotony and fixpoint property. Therefore if $p^\sharp \preceq^\sharp x$ and $p^\sharp \preceq^\sharp y$ imply $p^\sharp \preceq^\sharp x \mathbin{\triangle} y$ then obviously $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^k$ implies $\mathit{lfp}\, F^\sharp \preceq^\sharp \check{X}^{k+1}$. Since the fixpoints $p^\sharp$ of $F^\sharp$ are unknown, we require the narrowing operator to satisfy $\forall x, y, z \in P^\sharp$ : $z \preceq^\sharp x \wedge z \preceq^\sharp y \Rightarrow z \preceq^\sharp x \mathbin{\triangle} y$. Finally the downward iteration sequence $\check{X}^0, \ldots, \check{X}^{k+1} = \check{X}^k \mathbin{\triangle} F^\sharp(\check{X}^k), \ldots$ must be finite. Again since this must be true for all possible $F^\sharp$, we require the non-existence of strictly decreasing chains of the form $x^0, \ldots, x^{k+1} = x^k \mathbin{\triangle} y^k$ where $y^k$, $k \geq 0$ is a decreasing chain (due to monotony of $F^\sharp$).

The above discussion is a motivation for the definition of a *narrowing operator*, such that:

$$\triangle \in P^\sharp \times P^\sharp \mapsto P^\sharp \tag{16a}$$

$$\forall {p^\sharp}_1, {p^\sharp}_2 \in P^\sharp : {p^\sharp}_1 \mathbin{\triangle} {p^\sharp}_2 \preceq^\sharp {p^\sharp}_1 \tag{16b}$$

$$\forall {p^\sharp}_1, {p^\sharp}_2, {p^\sharp}_3 \in P^\sharp : {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2 \wedge {p^\sharp}_1 \preceq^\sharp {p^\sharp}_3 \Rightarrow {p^\sharp}_1 \preceq^\sharp {p^\sharp}_2 \mathbin{\triangle} {p^\sharp}_3 \tag{16c}$$

$$\begin{array}{l}\text{for all decreasing chains } {p^\sharp}^k, k \geq 0 \text{ and } {p^\sharp}' \in P^\sharp \text{ the chain}\\ \check{X}^0 = {p^\sharp}', \ldots, \check{X}^{k+1} = \check{X}^k \triangle {p^\sharp}^k, \ldots \text{ is not strictly decreasing}\\ \text{for } \preceq^\sharp\end{array} \tag{16d}$$

with the following convergence property showing how to improve upper-approximations of fixpoints:

*Proposition 30 (Downward abstract iteration sequence with narrowing). If $F^\sharp \in P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)$, $\triangle \in P^\sharp \times P^\sharp \mapsto P^\sharp$ is a narrowing operator and $F^\sharp(p^\sharp) = p^\sharp \preceq^\sharp {p^\sharp}'$, then the decreasing chain $\check{X}^0 = {p^\sharp}', \ldots, \check{X}^{k+1} = \check{X}^k \mathbin{\triangle} F^\sharp(\check{X}^k)$ is stationary with limit $\check{X}^\ell$, $\ell \in \mathbb{N}$ such that $p^\sharp \preceq^\sharp \check{X}^\ell \preceq^\sharp {p^\sharp}'$.*

PROOF. We prove $p^\sharp \preceq^\sharp \check{X}^k$ for all $k \in \mathbb{N}$. This holds for $k = 0$ by hypothesis. If $p^\sharp \preceq^\sharp \check{X}^k$ by induction hypothesis, then $p^\sharp = F^\sharp(p^\sharp) \preceq^\sharp F^\sharp(\check{X}^k)$ by monotony whence $p^\sharp \preceq^\sharp \check{X}^{k+1} = \check{X}^k \mathbin{\triangle} F^\sharp(\check{X}^k) \preceq^\sharp \check{X}^k$ by (16b) and (16c). Since the chain $\check{X}^k$, $k \geq 0$ is decreasing for $\preceq^\sharp$ then so is $F^\sharp(\check{X}^k)$, $k \geq 0$ by monotony. Therefore $\check{X}^k$, $k \geq 0$, which is not strictly decreasing by (16d), has a limit $\check{X}^\ell$ such that $p^\sharp \preceq^\sharp \check{X}^\ell \preceq^\sharp \check{X}^0 = {p^\sharp}'$. □

Observe that in a complete lattice satisfying the *descending chain condition* (that is all strictly decreasing chains for $\preceq^\sharp$ are finite) the narrowing operator $x \mathbin{\triangle} y$ can be defined as the greatest lower bound of $x$ and $y$ for $\preceq^\sharp$. Hypotheses (16) have numerous variants. For example if the starting point ${p^\sharp}'$ is a postfixpoint of $F^\sharp$ we can assume that ${p^\sharp}_2 \preceq^\sharp {p^\sharp}_1$ in (16b). Moreover, the narrowing operator can be chosen to depend upon the iteration step. In particular since any term of the chain $\check{X}^k$, $k \geq 0$ is sound we can stop iterations after an arbitrary number $n$ of steps so as to cut analysis costs down. In this case the narrowing $x \mathbin{\triangle^i} y$ would be $y$ if $i \leq n$ else $x$. Finally, more sophisticated convergence enforcement strategies could be designed by using not a single but all previous iterates.

### *4.3.2. Upward Abstract Iteration Sequence with Widening*

In general no approximation of the least fixpoint better than the supremum $t^\sharp$ is known to start with. Since the downward abstract iteration sequence with narrowing cannot undershoot fixpoints no approximation of the least fixpoint better than the greatest fixpoint can be computed by the method of proposition 30. Therefore, in order to get a better initial upper-approximation of the least fixpoint, one can start from below, for example from the infimum $f^\sharp$, using an increasing chain so as to overshoot this unknown least fixpoint. As shown by the practical experience, the benefit of this method is that very often the limit will be below the greatest fixpoint and in all cases below the supremum $t^\sharp$. Three problems have to be solved. When using an increasing chain $\hat{X}^k$, $k \in \mathbb{N}$ starting from below the least fixpoint $\mathit{lfp}\, F^\sharp$, we must first have a computable criterion to check whether a point $\hat{X}^\ell$ above the least fixpoint has been reached. Depending on the problem to be solved, several criteria are available such as $\hat{X}^\ell$ is a fixpoint of

$F^\sharp$ or, by Tarski's fixpoint theorem, $\hat{X}^\ell$ is a postfixpoint of $F^\sharp$. Second, we must ensure that the sequence $\hat{X}^k$, $k \in \mathbb{N}$ eventually reaches a point above the least fixpoint. A simple way to do so is to iterate above the chain ${F^\sharp}^0 = \bot^\sharp, \ldots, {F^\sharp}^{k+1} = F^\sharp({F^\sharp}^k), \ldots$, converging to the least fixpoint $\mathit{lfp}\, F^\sharp = \sqcup^\sharp_{k \geq 0} {F^\sharp}^k$. To do so we can use a *widening operator* $\triangledown \in P^\sharp \times P^\sharp \mapsto P^\sharp$ in order to extrapolate to $\hat{X}^{k+1} = \hat{X}^k \triangledown F^\sharp(\hat{X}^k)$ from two consecutive terms $\hat{X}^k$ and $F^\sharp(\hat{X}^k)$ so that $\hat{X}^k \preceq^\sharp \hat{X}^{k+1}$ and $F^\sharp(\hat{X}^k) \preceq^\sharp \hat{X}^{k+1}$. Third, we must ensure that the iteration sequence $\hat{X}^k$, $k \in \mathbb{N}$ stabilizes after finitely many steps. This leads to the definition of a widening operator, such that:

$$\triangledown \in P^\sharp \times P^\sharp \mapsto P^\sharp \tag{17a}$$

$$\begin{aligned}&\forall {p^\sharp}_1, {p^\sharp}_2, {p^\sharp}'_1, {p^\sharp}'_2 \in P^\sharp : ({p^\sharp}_1 \sqsubseteq^\sharp {p^\sharp}_2) \wedge ({p^\sharp}_1 \preceq^\sharp {p^\sharp}'_1) \wedge ({p^\sharp}_2 \preceq^\sharp {p^\sharp}'_2) \\ &\quad \Rightarrow ({p^\sharp}'_1 \preceq^\sharp {p^\sharp}'_1 \triangledown {p^\sharp}'_2) \wedge ({p^\sharp}_2 \preceq^\sharp {p^\sharp}'_1 \triangledown {p^\sharp}'_2)\end{aligned} \tag{17b}$$

$$\begin{aligned}&\text{for all increasing chains } {p^\sharp}^k, k \geq 0, \text{ the chain } \hat{X}^0 = {p^\sharp}^0, \ldots, \\ &\hat{X}^{k+1} = \hat{X}^k \triangledown {p^\sharp}^k, \ldots \text{ is not strictly increasing for } \preceq^\sharp\end{aligned} \tag{17c}$$

with the following convergence property showing how to compute upper-approximations of the least fixpoint starting from below:

*Proposition 31 (Upward abstract iteration sequence with widening). If $F^\sharp \in P^\sharp(\sqsubseteq^\sharp) \xrightarrow{c} P^\sharp(\sqsubseteq^\sharp)$, $F^\sharp \in P^\sharp(\preceq^\sharp) \xrightarrow{m} P^\sharp(\preceq^\sharp)$, $\triangledown$ is a widening operator and $\forall k \in \mathbb{N} : {p^\sharp}^k \preceq^\sharp {p^\sharp}' \Rightarrow \sqcup^\sharp_{k \in \mathbb{N}} {p^\sharp}^k \preceq^\sharp {p^\sharp}'$, then the increasing chain $\hat{X}^0 = \bot^\sharp$, $\hat{X}^{k+1} = \hat{X}^k \triangledown F^\sharp(\hat{X}^k)$ for $k \in \mathbb{N}$ is stationary with limit $\hat{X}^\ell$ such that $\mathit{lfp}\, F^\sharp \preceq^\sharp \hat{X}^\ell$.*

PROOF. Since $\preceq^\sharp$ is reflexive, we have ${F^\sharp}^0 = \bot^\sharp \preceq^\sharp \bot^\sharp = \hat{X}^0$. Assume ${F^\sharp}^k \preceq^\sharp \hat{X}^k$ then ${F^\sharp}^{k+1} = F^\sharp({F^\sharp}^k) \preceq^\sharp F^\sharp(\hat{X}^k)$ by monotony whence $\hat{X}^k \preceq^\sharp \hat{X}^{k+1}$ and ${F^\sharp}^{k+1} \preceq^\sharp \hat{X}^{k+1}$ by (17b) and $\hat{X}^{k+1} = \hat{X}^k \triangledown F^\sharp(\hat{X}^k)$. It follows that the chain $\hat{X}^k$, $k \geq 0$ hence by monotony $F^\sharp(\hat{X}^k)$, $k \geq 0$ is increasing but not strictly by (17c). For the limit $\hat{X}^\ell$ where $\ell \in \mathbb{N}$, we have ${F^\sharp}^k \preceq^\sharp \hat{X}^k \preceq^\sharp \hat{X}^\ell$ for all $k \leq \ell$. Moreover, if $m \geq \ell$ and ${F^\sharp}^m \preceq^\sharp \hat{X}^m = \hat{X}^\ell$, then $\hat{X}^{m+1} = \hat{X}^m \triangledown F^\sharp(\hat{X}^m) = \hat{X}^\ell \triangledown F^\sharp(\hat{X}^\ell) = \hat{X}^\ell$. It follows that $\forall k \in \mathbb{N} : {F^\sharp}^k \preceq^\sharp \hat{X}^\ell$, whence $\mathit{lfp}\, F^\sharp = \sqcup^\sharp_{k \in \mathbb{N}} {F^\sharp}^k \preceq^\sharp \hat{X}^\ell$. □

Once again one can imagine a number of weaker hypotheses on the widening operator, such as expressing correctness criteria (17) with respect to concrete properties, using widenings based upon all previous iterates and depending upon the rank of the iterates (so as for example to be able to speed up convergence by loosing more information as iteration time passes), using chaotic iterations with one widening operator only along each cycle in the dependence graph of the system of equations, etc. Proposition 31 only shows the way. Moreover, it is not always necessary to wait for the iterates to stabilize since for example, by Tarski's theorem [52], if the computational ordering $\sqsubseteq^\sharp$ coincides with the approximation $\preceq^\sharp$ ordering then $F^\sharp(\hat{X}^\ell) \preceq^\sharp \hat{X}^\ell$ implies $\mathit{lfp}\, F^\sharp \preceq^\sharp \hat{X}^\ell$.

Observe that [4], [3] and [32] use an infinite domain and a nonmonotone widening operator that enlarges, in a nonunique way, the denoted set of terms. This so-called *restriction* operation on normal types/abstract substitutions consists in removing from a type graph the paths of forward arcs where the number of occurrences of the same functor is greater than some given fixed constant. To do so, a cyclic graph is created describing infinitely many trees by paths of all possible lengths. It is also observed that to get more precise analyses, application of the restriction algorithm could be delayed until a diverging computation is observed for recursive calls. Finally, since the widening is not necessarily monotone, proposition 28 on chaotic iterations no longer applies so that the precision on the result may depend upon the chaotic iteration strategy which is chosen (but not its soundness).

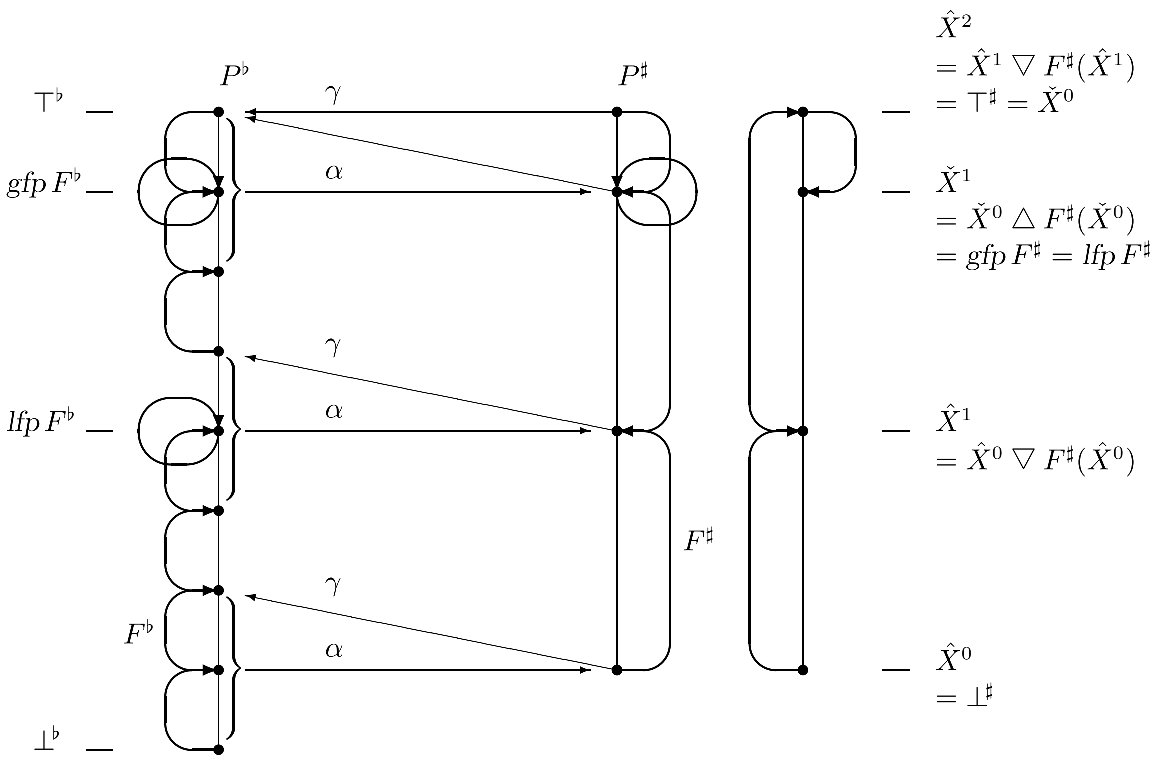



**FIGURE 9.** Fixpoint approximation using a Galois connection, a widening and a narrowing.

*4.3.3. Upward and Downward Abstract Iteration Sequences*

In practice, one first uses an upward abstract iteration sequence with widening to obtain an upper-approximation of the least fixpoint starting from below and then a downward abstract iteration sequence with narrowing so as to improve this upper bound while remaining above any fixpoint. This is illustrated in Figure 9.

*Example 32 (Interval analysis).* In order to analyse the possible values of integer variables, [9] and [10] introduced the abstraction $\alpha \in P^\flat(\subseteq) \mapsto P^\sharp(\preceq^\sharp)$ where $P^\flat = \wp(\mathbb{Z})$, $P^\sharp = \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \ \wedge\ u \in \mathbb{Z} \cup \{+\infty\} \ \wedge\ \ell \leq u\} \cup \{\emptyset\}$, $\mathit{min}\,\mathbb{Z} = -\infty$, $\mathit{max}\,\mathbb{Z} = +\infty$ such that $\alpha(\emptyset) = \emptyset$ and $\alpha(X) = [\mathit{min}\, X, \mathit{max}\, X]$. The computational ordering $\sqsubseteq^\sharp$ and approximation ordering $\preceq^\sharp$ are identical and defined by $\emptyset \preceq^\sharp I$ for all $I \in P^\sharp$ and $[a, b] \preceq^\sharp [c, d]$ if and only if $c \leq a \wedge b \leq d$. Since $P^\sharp$ has infinite strictly increasing chains, it is necessary to introduce a widening operator such that for all intervals $I \in P^\sharp$, $\emptyset \triangledown I = I \triangledown \emptyset = I$ and $[a, b] \triangledown [c, d] = [\text{if } c < a \text{ then } -\infty \text{ else } a, \text{if } d > b \text{ then } +\infty \text{ else } b]$. The strictly decreasing chains of the abstract lattice $P^\sharp$ are all finite but can be very long so that it is useful to define a narrowing such that for all intervals $I \in P^\sharp$, $\emptyset \vartriangle I = I \vartriangle \emptyset = \emptyset$ and $[a, b] \vartriangle [c, d] = [\text{if } a = -\infty \text{ then } c \text{ else } a, \text{if } b = +\infty \text{ then } d \text{ else } b]$.

The analysis of the output of the following PROLOG II program:

```
program   -> init(x,1) while(x);
init(x,x) -> ;
while(x)  -> val(inf(x,100),1) out(x) line val(add(x,2),y)
             while(y);
```

consists in solving the equation:

$$X = \Big([1, 1] \sqcup (X \oplus [2, 2])\Big) \sqcap [-\infty, 99]$$

where $\emptyset \oplus I = I \oplus \emptyset = \emptyset$ and $[a,b] \oplus [c,d] = [a \dot{+} c, b \dot{+} d]$ with $-\infty \dot{+} x = x \dot{+} -\infty = -\infty$ and $+\infty \dot{+} x = x \dot{+} +\infty = +\infty$. The ascending abstract iteration sequence with widening is the following:

$$\begin{aligned}
\hat{X}^0 &= \emptyset \\
\hat{X}^1 &= \hat{X}^0 \mathbin{\triangledown} \Big(([1,1] \sqcup (\hat{X}^0 \oplus [2,2])) \sqcap [-\infty, 99]\Big) = \emptyset \mathbin{\triangledown} [1,1] = [1,1] \\
\hat{X}^2 &= \hat{X}^1 \mathbin{\triangledown} \Big(([1,1] \sqcup (\hat{X}^1 \oplus [2,2])) \sqcap [-\infty, 99]\Big) = [1,1] \mathbin{\triangledown} [1,3] = [1,+\infty] \\
\hat{X}^3 &= \hat{X}^2 \mathbin{\triangledown} \Big(([1,1] \sqcup (\hat{X}^2 \oplus [2,2])) \sqcap [-\infty, 99]\Big) = [1,1] \mathbin{\triangledown} [1,99] = [1,+\infty]
\end{aligned}$$

The descending abstract iteration sequence with narrowing is now:

$$\begin{aligned}
\check{X}^0 &= \hat{X}^3 = [1,+\infty] \\
\check{X}^1 &= \check{X}^0 \mathbin{\vartriangle} \Big(([1,1] \sqcup (\check{X}^0 \oplus [2,2])) \sqcap [-\infty, 99]\Big) = [1,+\infty] \mathbin{\vartriangle} [1,99] = [1,99] \\
\check{X}^2 &= \check{X}^1 \mathbin{\vartriangle} \Big(([1,1] \sqcup (\check{X}^1 \oplus [2,2])) \sqcap [-\infty, 99]\Big) = [1,99] \mathbin{\vartriangle} [1,99] = [1,99]
\end{aligned}$$

Observe that the analysis time does not depend upon the number of iterations in the while-loop which would be the case without using widening and narrowing operators. □

*4.3.4. A Compromise Between Relational and Attribute Independent Analyses Using Widenings*

Since relational analyses are powerful but expansive whereas attribute independent analyses are cheaper but less precise, the use of widening operators may offer an interesting compromise. For example, if $P^{\sharp^i}$ is the lattice $\{\perp, \text{G}, \text{NG}, \top\}$ for groundness analysis of term $t_i$ then the down-set completion $P^\sharp$ of the reduced product $\prod_{i=1}^n P^{\sharp^i}$ can express dependencies between groundness properties of arguments of atoms $p(t_1, \ldots, t_n)$. By proposition 18, elements of $P^\sharp$ can be represented by subsets of $\prod_{i=1}^n \{\text{G}, \text{NG}\}$ in which case strictly increasing chains in $P^\sharp$ have a maximal size of $O(2^n)$. Expressing dependencies between the different arguments of all atoms in the program would be even more expansive [24]. This cost can be cut down using a widening operator. A brute force one would be $X \mathbin{\triangledown} Y \stackrel{\text{def}}{=}$ if Cardinality$(Y) \leq \ell(n)$ then $X \cup Y$ else $\prod_{i=1}^n \{\text{G}, \text{NG}\}$ where $\ell(n)$ is a parameter which can be adjusted to tune the cost/performance ratio.

## 5. OPERATIONAL AND COLLECTING SEMANTICS

Abstract interpretations of programs must be proved correct with respect to a standard semantics of these programs. Following [7], the standard semantics that we will choose is operational. A popular alternative is to choose a denotational semantics. But this choice would be less fundamental since denotational semantics can be derived from the operational semantics by abstract interpretation [20].

It is possible to group program properties into classes, such as invariance and liveness properties, for which all correctness proofs of abstract interpretations of one class will essentially be the same, but for the particular abstract properties which are chosen in the class. By giving a correctness proof of the abstract interpretation for the strongest property in the class, we can factor all these proofs out into two independent steps. First, a fixpoint collecting semantics is given which characterizes the strongest property in the class of interest. It is proved correct with respect to the standard semantics. Second, abstract interpretations in the given class are proved correct with respect to the corresponding collecting semantics.

There are many other interests in this separation process. The collecting semantics is sound but usually also complete with respect to the considered class of program properties. Hence it can serve as a basis for developing program correctness proof methods [8]. Knowledge about the considered class of properties can be usefully incorporated once for all into the collecting semantics. For example [44] and [26] observed that invariance properties can always be proved using sets of states attached to program points and this was incorporated into the *static semantics* of [10]. Another example, recalled in paragraph 4.2.3.6., is given in [15] (example 6.2.0.2) where it is shown how relational invariants can be decomposed into attribute independent ones (where relationships between variables are lost). Taken together these two examples show that an abstract interpretation can be decomposed into what concerns control and what concerns data, the two aspects being treated separately. Using combination methods as proposed in [15] and recalled in section 4.2.3., this leads to a modular design of abstract interpreters. In doing so, useful abstract interpretations can be easily transferred from one language to another.

Another important step was taken in [7] where it was understood that collecting semantics can be studied in abstracto, independently of a particular language. For example, the static semantics of [10] was expressed using transition systems (due to [37]) hence in a language independent manner. The difficulty of generalizing program points for expression languages was solved by understanding them as the more general technique of covering the concrete domain by partitions, partial equivalence relations or other covers.

Choosing once for all a particular collecting semantics and claiming that it is the only sensible alternative would lead to rigid approximation decisions which could later turn out to be impractical (for example by approximating functions by functions whereas tuples (as in type checking) or relations (between argument values and results) can do better) and to rule out analysing program properties which are forgotten by the collecting semantics, or very difficult to express in the chosen framework (such as execution order). Therefore, we proceed by working out meta-collecting semantics, where 'meta' means language independent and easily instantiable for particular programming languages, and by relating them by abstract interpretations, so as to understand this family of collecting semantics as a set of possible intermediate steps in the approximation of program executions.

To illustrate this approach for logic programming languages, we will chose here to start from an operational point of view formalized by transition systems. We will consider invariance properties which are characterized as fixpoints of predicate transformers. This will be first done in a language independent way. Later, these results will be instantiated for logic programs.

Abstract interpretation is mostly used to derive an abstract semantics from a concrete semantics. But the contrary is also possible. For example in [40] the standard domains of goals is the abstract domain, while the concrete domain is a new one containing timing information.

### 5.1. *Operational Semantics as Transition Systems*

The small-steps operational semantics of a programming language $\mathfrak{L}$ associates a transition system $\langle \mathfrak{S}, \mathfrak{I}, \mathfrak{F}, \vdash \mathtt{P} \rightarrow \rangle$ to each program $\mathtt{P}$ of the language. $\mathfrak{S}$ is a set of states, $\mathfrak{I} \subseteq \mathfrak{S}$ is the set of initial states, $\mathfrak{F} \subseteq \mathfrak{S}$ is the set of final states while $\vdash \mathtt{P} \rightarrow \in \wp(\mathfrak{S} \times \mathfrak{S})$ is a transition relation between a state and its possible successors. The idea is that program execution starts with some initial state $s_0 \in \mathfrak{I}$. After i execution steps leading to state $s_i \in \mathfrak{S}$ a further execution step can lead to any successor state $s_{i+1} \in \mathfrak{S}$ as given by the transition relation so that $s_i \vdash \mathtt{P} \rightarrow s_{i+1}$. This execution can either run for ever or else terminate either with a final state $s_n \in \mathfrak{F}$ or with a blocking state without successor for the transition relation. A familiar example which will be developed later is SLD-resolution for logic programs. An initial state consists of an initial goal and the empty substitution. A final state has an empty goal and an answer substitution. A state is simply a current goal and a substitution. A transition consists in unifying a selected atom in the goal with the head of a program clause and in replacing it by a unified instance of the body of the clause in the new goal together with a new substitution obtained by composition of the old one with the most general unifier.

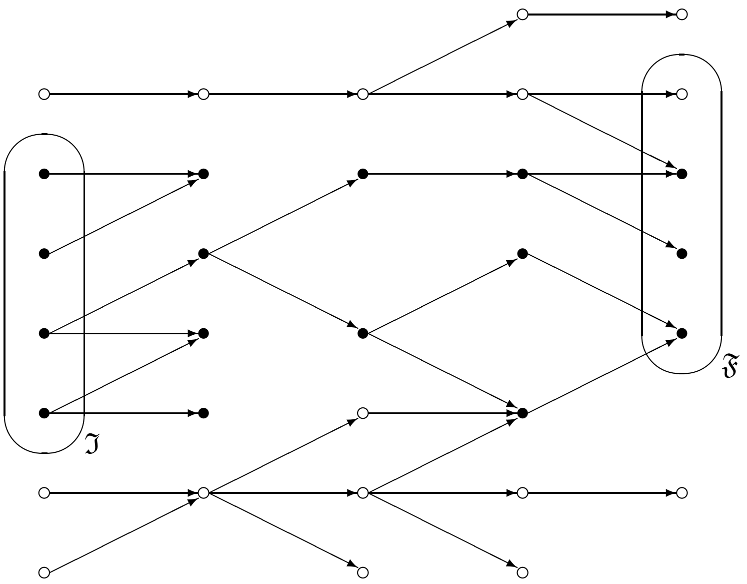



**FIGURE 10.** Descendant states ($\bullet$) of the initial states ($\mathfrak{I}$).

### *5.2. Top/Down — Forward Collecting Semantics*

The top/down (also called forward) collecting semantics characterizes the descendant states of the initial states as illustrated in Figure 10. For logic programs, the set of descendant states of the initial states provides information about all calls for a given initial question regardless of whether they succeed, finitely fail or do not terminate.

Given a relation $t \in \wp(S \times S)$, its *transitive closure* is $t^\star = \bigcup_{n\in\mathbb{N}} t^n$ where $t^0 = 1 = \{\langle s, s'\rangle \mid s = s'\}$, $t^{n+1} = t \circ t^n = t^n \circ t$ and $t \circ t' = \{\langle s, s''\rangle \mid \exists s' : \langle s, s'\rangle \in t \wedge \langle s', s''\rangle \in t'\}$. The fundamental fixpoint characterization of transitive closures is that $t^\star = \mathit{lfp}\, T$ where $T \in \wp(S \times S)(\cup) \xmapsto{a} \wp(S \times S)(\cup)$ is defined by $T(X) = 1 \cup X \circ t$.

The top/down collecting semantics for program $\mathtt{P}$ is the set $\mathfrak{D}$ of descendant states of the initial states, that is $\mathfrak{D} = \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{I} : \mathfrak{s}' \vdash\!\mathtt{P}\!\rightarrow^\star \mathfrak{s}\}$ which can be written $\mathit{post}[\vdash\!\mathtt{P}\!\rightarrow^\star]\mathfrak{I}$ by defining:

$$\mathit{post} \in \wp(\mathfrak{S} \times \mathfrak{S}) \mapsto \big(\wp(\mathfrak{S}) \mapsto \wp(\mathfrak{S})\big) \qquad \mathit{post}[\mathfrak{t}]\mathfrak{X} \stackrel{\text{def}}{=} \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{X} : \langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{t}\} \tag{18}$$

Using the fixpoint inducing proposition 23, we can use the above fixpoint characterization of transitive closures to provide a fixpoint definition of this top/down collecting semantics:

*Proposition 33 (Fixpoint characterization of the top/down collecting semantics).* $\mathfrak{D} = \mathit{lfp}\, \mathfrak{F}[\![\mathtt{P}]\!]$ *where* $F[\![\mathtt{P}]\!] \in \wp(S)(\cup) \xmapsto{a} \wp(S)(\cup)$ *is defined by* $F[\![\mathtt{P}]\!]X = \mathfrak{I} \cup \mathit{post}[\vdash\!\mathtt{P}\!\rightarrow]\mathfrak{X}$.

PROOF. Observe that $\wp(\mathfrak{S} \times \mathfrak{S})(\subseteq, \emptyset, \mathfrak{S} \times \mathfrak{S}, \cup, \cap)$ and $\wp(\mathfrak{S})(\subseteq, \emptyset, \mathfrak{S}, \cup, \cap)$ are complete lattices. Define $\alpha \in \wp(\mathfrak{S} \times \mathfrak{S}) \mapsto \wp(\mathfrak{S})$ by $\alpha(X) = \mathit{post}[X]\mathfrak{I}$. It is a complete $\cup$-morphism so that by proposition 7 there exists $\gamma$ such that $\wp(\mathfrak{S} \times \mathfrak{S})(\subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathfrak{S})(\subseteq)$. We have $\vdash\!\mathtt{P}\!\rightarrow^\star = \mathit{lfp}\, T = \bigcup_{n\in\mathbb{N}} T^n(\emptyset)$ where $T(X) = 1 \cup X\circ \vdash\!\mathtt{P}\!\rightarrow$, $\emptyset = \alpha(\emptyset)$ and $F[\![\mathtt{P}]\!] \in \wp(S)(\cup) \xmapsto{a} \wp(S)(\cup)$ is such that for all $X \in \wp(\mathfrak{S} \times \mathfrak{S})$, we have $\alpha \circ T(X) = \mathit{post}[T(X)]\mathfrak{I} = \{s \mid \exists s' \in \mathfrak{I} : \langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{T}(\mathfrak{X})\}$ $= \{s \mid \exists s' \in \mathfrak{I} : \langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{1} \cup \mathfrak{X}\circ \vdash\!\mathtt{P}\!\rightarrow\} = \{s \mid \exists s' \in \mathfrak{I} : (\langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{1}) \vee (\langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{X}\circ \vdash\!\mathtt{P}\!\rightarrow)\}$ $= \{s \mid \exists s' \in \mathfrak{I} : (\mathfrak{s}' = \mathfrak{s}) \vee (\langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{X}\circ \vdash\!\mathtt{P}\!\rightarrow)\} = \mathfrak{I} \cup \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{I} : \langle \mathfrak{s}', \mathfrak{s}\rangle \in \mathfrak{X}\circ \vdash\!\mathtt{P}\!\rightarrow\}$ $= \mathfrak{I} \cup \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{I} : \exists \mathfrak{s}'' \in \mathfrak{S} : \langle \mathfrak{s}', \mathfrak{s}''\rangle \in \mathfrak{X} \wedge \mathfrak{s}'' \vdash\!\mathtt{P}\!\rightarrow \mathfrak{s}\} = \mathfrak{I} \cup \{\mathfrak{s} \mid \exists \mathfrak{s}'' \in \{\mathfrak{s}'' \mid \exists \mathfrak{s}' \in \mathfrak{I} : \langle \mathfrak{s}', \mathfrak{s}''\rangle \in \mathfrak{X}\} : \mathfrak{s}'' \vdash\!\mathtt{P}\!\rightarrow \mathfrak{s}\} = \mathfrak{I} \cup \{\mathfrak{s} \mid \exists \mathfrak{s}'' \in \mathit{post}[\mathfrak{X}]\mathfrak{I} : \mathfrak{s}'' \vdash\!\mathtt{P}\!\rightarrow \mathfrak{s}\} = \mathfrak{I} \cup \mathit{post}[\vdash\!\mathtt{P}\!\rightarrow](\mathit{post}[\mathfrak{X}]\mathfrak{I})$ $= F[\![\mathtt{P}]\!](\mathit{post}[X]\mathfrak{I}) = F[\![\mathtt{P}]\!] \circ \alpha(X)$. By proposition 23, we have $\alpha(\mathit{lfp}\, T) = \mathit{lfp}\, F[\![\mathtt{P}]\!]$ so that $\mathfrak{D} = \mathit{post}[\vdash\!\mathtt{P}\!\rightarrow^\star]\mathfrak{I} = \alpha(\vdash\!\mathtt{P}\!\rightarrow^\star) = \alpha(\mathit{lfp}\, T) = \mathit{lfp}\, F[\![\mathtt{P}]\!]$. $\square$

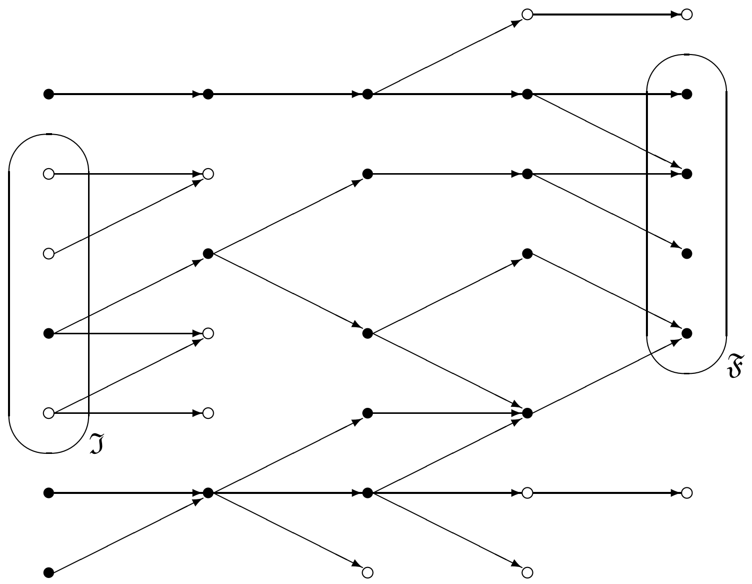



**FIGURE 11.** Ascendant states (•) of the final states ($\mathfrak{F}$).

### *5.3. Bottom/Up — Backward Collecting Semantics*

The bottom/up (also called backward) collecting semantics characterizes the ascendant states of the final states as illustrated in Figure 11. For logic programs, the set of ascendant states of the final states provides information about atoms that can succeed.

The bottom/up collecting semantics for program P is the set $\mathfrak{A}$ of ascendant states of the final states, that is $\mathfrak{A} = \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{F} : \mathfrak{s} \vdash\mathtt{P}\rightarrow^\star \mathfrak{s}'\}$ which can be written $pre[\vdash\mathtt{P}\rightarrow^\star]\mathfrak{F}$ by defining:

$$pre \in \wp(\mathfrak{S} \times \mathfrak{S}) \mapsto (\wp(\mathfrak{S}) \mapsto \wp(\mathfrak{S})) \qquad pre[\mathfrak{t}]\mathfrak{X} \stackrel{\text{def}}{=} \{\mathfrak{s} \mid \exists \mathfrak{s}' \in \mathfrak{X} : \langle \mathfrak{s}, \mathfrak{s}'\rangle \in \mathfrak{t}\} \tag{19}$$

Observe that the ascendant states $\mathfrak{A}$ of the final states $\mathfrak{F}$ of the transition system $\langle\mathfrak{S}, \mathfrak{I}, \mathfrak{F}, \vdash\mathtt{P}\rightarrow\rangle$ is precisely the set of descendant states of the initial states of the inverse transition system $\langle\mathfrak{S}, \mathfrak{F}, \mathfrak{I}, \vdash\mathtt{P}\rightarrow^{-1}\rangle$ where the inverse $t^{-1}$ of a relation $t \in \wp(S \times S)$ is $\{\langle s', s\rangle \mid \langle s, s'\rangle \in t\}$. For that reason it is traditional not to explicitly study the backward abstract interpretations since, from a theoretical point of view, they are essentially the same as the forward ones (provided adequate, that is invertible, collecting semantics are considered, an handicap for denotational semantics). For example, we have:

*Proposition 34 (Fixpoint characterization of the bottom/up collecting semantics).* $\mathfrak{A} = \mathit{lfp}\,\mathfrak{B}[\![\mathtt{P}]\!]$ *where* $B[\![\mathtt{P}]\!] \in \wp(S)(\cup) \xrightarrow{a} \wp(S)(\cup)$ *is defined by* $B[\![\mathtt{P}]\!]X = pre[\vdash\mathtt{P}\rightarrow]X \cup \mathfrak{F}$.

PROOF. Using the fact that $(t^\star)^{-1} = (t^{-1})^\star$ and $pre[t]X = post[t^{-1}]X$, we have $\mathfrak{A} = pre[\vdash\mathtt{P}\rightarrow^\star]\mathfrak{F}$ $= post[(\vdash\mathtt{P}\rightarrow^\star)^{-1}]\mathfrak{F} = post[(\vdash\mathtt{P}\rightarrow^{-1})^\star]\mathfrak{F} = \mathit{lfp}\,\lambda X\boldsymbol{\cdot}\mathfrak{F} \cup post[(\vdash\mathtt{P}\rightarrow)^{-1}]\mathfrak{X}$ by proposition 33, which is equal to $\mathit{lfp}\,\lambda X\boldsymbol{\cdot}pre[\vdash\mathtt{P}\rightarrow]X \cup \mathfrak{F} = \mathit{lfp}\,B[\![\mathtt{P}]\!]$. □

### *5.4. Combining the Top/Down — Forward and Bottom/Up — Backward Collecting Semantics*

In practice, we are interested by programs that succeed, so that the program interpreter should not enter dead-ends, that is states for which execution can only fail or not terminate properly. Therefore, we are interested in characterizing the descendant states of the initial states which are also ascendant states of the final states, as shown in Figure 12. The set of descendant states of the initial states which are the ascendant states of the final states of a transition system $\langle\mathfrak{S}, \mathfrak{I}, \mathfrak{F}, \vdash\mathtt{P}\rightarrow\rangle$ corresponding to a program P is characterized by $\mathfrak{D} \cap \mathfrak{A} = \mathit{lfp}\,F[\![\mathtt{P}]\!] \cap \mathit{lfp}\,B[\![\mathtt{P}]\!]$.

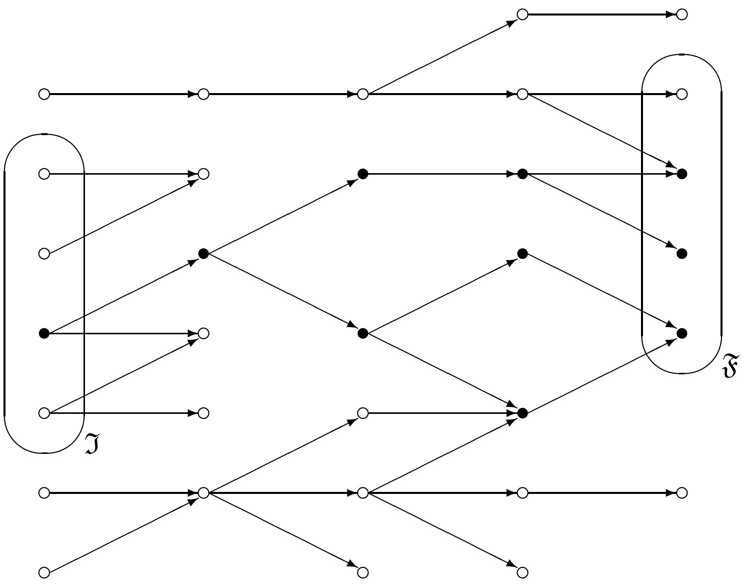



**FIGURE 12.** Descendant states (•) of the initial states ($\mathfrak{I}$) which are ascendant states of the final states ($\mathfrak{F}$).

In order to justify the technique later used to approximate this meet of fixpoints, we will use the following properties ([6]):

*Proposition 35 (Fixpoint properties of collecting semantics). For all transition systems* $\langle \mathfrak{S}, \mathfrak{I}, \mathfrak{F}, \vdash\!\mathtt{P}\!\to\rangle$ *and* $X \subseteq \mathfrak{S}$, *we have:*

1) $(pre[\vdash\!\mathtt{P}\!\to]X) \cap lfp\, F[\![\mathtt{P}]\!] \;\subseteq\; pre[\vdash\!\mathtt{P}\!\to](X \cap lfp\, F[\![\mathtt{P}]\!])$

2) $(post[\vdash\!\mathtt{P}\!\to]X) \cap lfp\, B[\![\mathtt{P}]\!] \;\subseteq\; post[\vdash\!\mathtt{P}\!\to](X \cap lfp\, B[\![\mathtt{P}]\!])$

$lfp\, F[\![\mathtt{P}]\!] \cap lfp\, B[\![\mathtt{P}]\!]$

3) $= lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]X)$

4) $= lfp\, \lambda X \boldsymbol{\cdot} (lfp\, B[\![\mathtt{P}]\!] \cap F[\![\mathtt{P}]\!]X)$

5) $= lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap lfp\, B[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]X)$

6) $= lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap lfp\, B[\![\mathtt{P}]\!] \cap F[\![\mathtt{P}]\!]X)$

PROOF. — To prove 1), observe that (19) and proposition 33 imply that $(pre[\vdash\!\mathtt{P}\!\to]X) \cap lfp\, F[\![\mathtt{P}]\!] = \{s \mid \exists s' \in X : s \vdash\!\mathtt{P}\!\to s'\} \cap \{s \mid \exists s'' \in \mathfrak{I} : \mathfrak{s}'' \vdash\!\mathtt{P}\!\to^\star \mathfrak{s}\} \subseteq \{s \mid \exists s' \in X : \exists s'' \in \mathfrak{I} : \mathfrak{s}'' \vdash\!\mathtt{P}\!\to^\star \mathfrak{s}' \wedge \mathfrak{s} \vdash\!\mathtt{P}\!\to \mathfrak{s}'\}$ since $s'' \vdash\!\mathtt{P}\!\to^\star s$ and $s \vdash\!\mathtt{P}\!\to s'$ imply $s'' \vdash\!\mathtt{P}\!\to^\star s'$. This is precisely $pre[\vdash\!\mathtt{P}\!\to](X \cap post[\vdash\!\mathtt{P}\!\to^\star]\mathfrak{I}) = pre[\vdash\!\mathtt{P}\!\to](X \cap lfp\, F[\![\mathtt{P}]\!])$. The proof of 2) is similar to that of 1).

— To prove 3), let $X^n, n \in \mathbb{N}$ and $Y^n, n \in \mathbb{N}$ be the iteration sequences starting from the infimum $\emptyset$ for $B[\![\mathtt{P}]\!]$ and $\lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]X)$ respectively. We have $lfp\, F[\![\mathtt{P}]\!] \cap X^0 = \emptyset = Y^0$. Assume that $lfp\, F[\![\mathtt{P}]\!] \cap X^n \subseteq Y^n$ by induction hypothesis. Then $lfp\, F[\![\mathtt{P}]\!] \cap X^{n+1} = lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!](X^n) = lfp\, F[\![\mathtt{P}]\!] \cap (pre[\vdash\!\mathtt{P}\!\to]X^n \cup \mathfrak{F}) = lfp\, F[\![\mathtt{P}]\!] \cap \Big((pre[\vdash\!\mathtt{P}\!\to]X^n \cap lfp\, F[\![\mathtt{P}]\!]) \cup \mathfrak{F}\Big)$, which, by 1), is included in $lfp\, F[\![\mathtt{P}]\!] \cap \Big(pre[\vdash\!\mathtt{P}\!\to](X^n \cap lfp\, F[\![\mathtt{P}]\!]) \cup \mathfrak{F}\Big)$ which, by induction hypothesis and monotony, is included in $lfp\, F[\![\mathtt{P}]\!] \cap (pre[\vdash\!\mathtt{P}\!\to](Y^n) \cup \mathfrak{F}) = lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]Y^n = Y^{n+1}$. It follows that $lfp\, F[\![\mathtt{P}]\!] \cap lfp\, B[\![\mathtt{P}]\!] = (\cup_{n\in\mathbb{N}} X^n) \cap lfp\, F[\![\mathtt{P}]\!] = \cup_{n\in\mathbb{N}} (X^n \cap lfp\, F[\![\mathtt{P}]\!]) \subseteq \cup_{n\in\mathbb{N}} Y^n = lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]X)$. But $lfp$ is monotone so that $lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!] \cap B[\![\mathtt{P}]\!]X) \subseteq lfp\, \lambda X \boldsymbol{\cdot} (lfp\, F[\![\mathtt{P}]\!]) \cap lfp\, \lambda X \boldsymbol{\cdot} (B[\![\mathtt{P}]\!]X) = lfp\, F[\![\mathtt{P}]\!] \cap lfp\, B[\![\mathtt{P}]\!]$. Equality follows by antisymmetry. The proofs of 4) to 6) are similar. □

## 6. COMBINING TOP/DOWN-FORWARD AND BOTTOM/UP-BACKWARD ABSTRACT INTERPRETATION

In order to approximate $\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat$ from above using abstract interpretations $F^\sharp$ of $F^\flat$ and $B^\sharp$ of $B^\flat$, we can use the abstract upper approximation $\mathit{lfp}\, F^\sharp \wedge^\sharp \mathit{lfp}\, B^\sharp$. However, a better approximation suggested in [6] can be obtained as the limit of the decreasing chain $\dot{X}^0 = \mathit{lfp}\, F^\sharp$ and $\dot{X}^{2n+1} = \mathit{lfp}\, \lambda\, X \cdot \dot{X}^{2n} \wedge^\sharp F^\sharp(X)$, $\dot{X}^{2n+2} = \mathit{lfp}\, \lambda\, X \cdot \dot{X}^{2n+1} \wedge^\sharp B^\sharp(X)$ for all $n \in \mathbb{N}$. Observe that by proposition **35** there is no improvement when considering the exact collecting semantics. However, when considering approximations of the collecting semantics, not all information can be collected in one pass. So the idea is to propagate the initial conditions top/down so as to get conditions on applicability of the unit clauses. These conditions are then propagated bottom/up to get stronger necessary conditions to be satisfied by the initial goal for possible success. This restricts the possible subgoals as indicated by the next top/down pass. Going on this way, the available information on the descendant states of the initial states which are ascendant states of the final states can be improved on each successive pass, until convergence. A similar scheme was used independently by [36] to infer types in flowchart programs. If the abstract lattice does not satisfy the descending chain condition then [6] also suggests to use a narrowing operator $\triangle$ to enforce convergence of the downward iteration $\dot{X}^k, k \in \mathbb{N}$. The same way a widening/narrowing approach can be used to enforce convergence of the iterates for $\lambda\, X \cdot \dot{X}^{2n} \wedge^\sharp F^\sharp(X)$ and $\lambda\, X \cdot \dot{X}^{2n+1} \wedge^\sharp B^\sharp(X)$. The correctness of this approach follows from:

*Proposition 36 (Fixpoint meet approximation). If $P^\flat(\preceq^\flat, f^\flat, t^\flat, \wedge^\flat, \vee^\flat)$ and $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ are complete lattices, $P^\flat(\preceq^\flat) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, $F^\flat \in P^\flat(\preceq^\flat) \overset{m}{\longmapsto} P^\flat(\preceq^\flat)$ and $B^\flat \in P^\flat(\preceq^\flat) \overset{m}{\longmapsto} P^\flat(\preceq^\flat)$ satisfy hypotheses 5) and 6) of proposition 35, $F^\sharp \in P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)$, $B^\sharp \in P^\sharp(\preceq^\sharp) \overset{m}{\longmapsto} P^\sharp(\preceq^\sharp)$, $\alpha \circ F^\flat \circ \gamma \preceq^\sharp F^\sharp$, $\alpha \circ B^\flat \circ \gamma \preceq^\sharp B^\sharp$, $\dot{X}^0$ is $\mathit{lfp}\, F^\sharp$ or $\mathit{lfp}\, B^\sharp$ and for all $n \in \mathbb{N}$, $\dot{X}^{2n+1} = \mathit{lfp}\, \lambda\, X \cdot (\dot{X}^{2n} \wedge^\sharp B^\sharp(X))$ and $\dot{X}^{2n+2} = \mathit{lfp}\, \lambda\, X \cdot (\dot{X}^{2n+1} \wedge^\sharp F^\sharp(X))$ then $\forall k \in \mathbb{N} : \alpha(\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat) \preceq^\sharp \dot{X}^{k+1} \preceq^\sharp \dot{X}^k$.*

PROOF. Observe that by the fixpoint property, $\dot{X}^{2n+1} = \dot{X}^{2n} \wedge^\sharp B^\sharp(\dot{X}^{2n+1})$ and $\dot{X}^{2n+2} = \dot{X}^{2n+1} \wedge^\sharp F^\sharp(\dot{X}^{2n+2})$, hence $\dot{X}^{2n} \preceq^\sharp \dot{X}^{2n+1} \preceq^\sharp \dot{X}^{2n+2}$ since $\wedge^\sharp$ is the greatest lower bound for $\preceq^\sharp$ so that $\dot{X}^k$, $k \in \mathbb{N}$ is a decreasing chain.

We have $\alpha(\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat) \preceq^\sharp \alpha(\mathit{lfp}\, F^\flat)$ since $\alpha$ is monotone and $\alpha(\mathit{lfp}\, F^\flat) \preceq^\sharp \mathit{lfp}\, F^\sharp$ by propositions 24 and 25, thus proving the proposition for $k = 0$. Let us observe that $\alpha \circ F^\flat \circ \gamma \preceq^\sharp F^\sharp$ implies $F^\flat \circ \gamma \preceq^\flat \gamma \circ F^\sharp$ by (1) so that in particular for an argument of the form $\alpha(X)$, $F^\flat \circ \gamma \circ \alpha \preceq^\flat \gamma \circ F^\sharp \circ \alpha$. By (8), $\gamma \circ \alpha$ is extensive so that by monotony and transitivity $F^\flat \preceq^\flat \gamma \circ F^\sharp \circ \alpha$. Assume now by induction hypothesis that $\alpha(\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat) \preceq^\sharp \dot{X}^{2n}$, whence $\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat \preceq^\sharp \gamma(\dot{X}^{2n})$ by (1). Since $F^\flat \preceq^\flat \gamma \circ F^\sharp \circ \alpha$, it follows that $\lambda\, X \cdot \mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat \wedge^\flat F^\flat(X) \preceq^\sharp \lambda\, X \cdot \gamma(\dot{X}^{2n}) \wedge^\flat \gamma \circ F^\sharp \circ \alpha(X) = \lambda\, X \cdot \gamma(\dot{X}^{2n} \wedge^\flat F^\sharp \circ \alpha(X))$ since $\gamma$ is a complete meet morphism (6). Now by hypothesis 5) of proposition 35, we have $\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat = \mathit{lfp}\, \lambda\, X \cdot (\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat \wedge^\flat F^\flat(X)) \preceq^\sharp \mathit{lfp}\, \lambda\, X \cdot \gamma(\dot{X}^{2n} \wedge^\flat F^\sharp \circ \alpha(X))$ by proposition 25. Let $G$ be $\lambda\, X \cdot \dot{X}^{2n} \wedge^\flat F^\sharp(X)$. By (3), $\alpha \circ \gamma$ is reductive so that by monotony $G \circ \alpha \circ \gamma \preceq^\sharp G$ and $\alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp G \circ \alpha \circ \gamma$, whence, by transitivity, $\alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp G$. By proposition 24, we have $\alpha(\mathit{lfp}\, \gamma \circ G \circ \alpha) \preceq^\sharp \mathit{lfp}\, \alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp \mathit{lfp}\, G$ by proposition 25. Hence, $\mathit{lfp}\, \lambda\, X \cdot \gamma(\dot{X}^{2n} \wedge^\flat F^\sharp \circ \alpha(X)) \preceq^\flat \gamma(\mathit{lfp}\, \lambda\, X \cdot \dot{X}^{2n} \wedge^\flat F^\sharp(X))$ so that by transitivity we conclude that $\alpha(\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat) \preceq^\sharp \dot{X}^{2n+1}$. The proof that $\alpha(\mathit{lfp}\, F^\flat \wedge^\flat \mathit{lfp}\, B^\flat) \preceq^\sharp \dot{X}^{2n+2}$ is similar using hypothesis 6) of proposition 35. □

## REFERENCES

[1] F. Bourdoncle. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In P. Deransart and Małuszyński, editors, *Proceedings of the International Workshop PLILP'90, Programming Language Implementation and Logic Programming,*

Linköping Sweden, 20–22 August 1990, Lecture Notes in Computer Science 456, pages 307–323. Springer-Verlag, Berlin, Germany, 1990.

[2] M. Bruynooghe. A practical framework for abstract interpretation of logic programs. Revised version of report CW 62, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, October 1987. (to appear in Journal of Logic Programming).

[3] M. Bruynooghe and G. Janssens. An instance of abstract interpretation integrating type and mode inferencing (extended abstract). In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming, Volume 1*, Seattle, Washington, pages 669–683. MIT Press, Cambridge, Massachusetts, U.S.A., August 15–19, 1988.

[4] M. Bruynooghe, G. Janssens, A. Callebaut, and B. Demoen. Abstract interpretation: towards the global optimization of Prolog programs. In *Proceedings of the 1987 International Symposium on Logic Programming*, San Francisco, California, pages 192–204. IEEE Computer Society Press, Los Alamitos, California, U.S.A., August 31–September 4, 1987.

[5] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Conference Record of the 19th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 343–354, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[6] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.

[7] P. Cousot. Semantic foundations of program analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

[8] P. Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 15, pages 843–993. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1990.

[9] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, U.S.A.

[11] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. In *ACM Symposium on Artificial Intelligence & Programming Languages*, Rochester, New York, SIGPLAN Notices 12(8):1–12, 1977.

[12] P. Cousot and R. Cousot. Static determination of dynamic properties of generalized type unions. In *ACM Symposium on Language Design for Reliable Software*, Raleigh, North Carolina, SIGPLAN Notices 12(3):77–94, 1977.

[13] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E.J. Neuhold, editor, *IFIP Conference on Formal Description of Programming Concepts*, St-Andrews, N.B., Canada, pages 237–277. North-Holland Pub. Co., Amsterdam, The Netherlands, 1977.

[14] P. Cousot and R. Cousot. A constructive characterization of the lattices of all retractions, pre-closure, quasi-closure and closure operators on a complete lattice. *Portugaliæ Mathematica*, 38(2):185–198, 1979.

[15] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, U.S.A.

[16] P. Cousot and R. Cousot. Semantic analysis of communicating sequential processes. In J. W. de Bakker and J. van Leeuwen, editors, *Seventh International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 85, pages 119–133. Springer-Verlag, Berlin, Germany, July 1980.

[17] P. Cousot and R. Cousot. Invariance proof methods and analysis techniques for parallel programs. In A. W. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic Program Construction Techniques*, chapter 12, pages 243–271. Macmillan, New York, U.S.A., 1984.

[18] P. Cousot and R. Cousot. 'à la Floyd' induction principles for proving inevitability properties of programs. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 8, pages 277–312. Cambridge University Press, Cambridge, U.K., 1985.

[19] P. Cousot and R. Cousot. Abstract interpretation frameworks. Research report LIX/RR/92/08, Laboratoire d'Informatique, École Polytechnique, Palaiseau France, 1991.

[20] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[21] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the 5th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, U.S.A.

[22] R. Cousot. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles*. Thèse d'État ès sciences mathématiques, Institut National Polytechnique de Lorraine, Nancy, France, 15 November 1985.

[23] S. K. Debray and D. S. Warren. Automatic mode inference of logic programs. *Journal of Logic Programming*, 5(3):207–229, 1988.

[24] S.K. Debray. The mythical free lunch: Notes on the complexity/precision tradeoff in dataflow analysis of logic programs. In R. Giacobazzi, editor, *Proceedings of the ICLP'91 Pre-Conference Workshop on Semantics-Based Analysis of Logic Programs*, INRIA, Rocquencourt, France. Dipartimento di Informatica, Università di Pisa, Italy, June 24, 1991.

[25] V. Englebert, B. Le Charlier, D. Roland, and P. Van Hentenryck. Generic abstract interpretation algorithms for Prolog: Two optimizations techniques and their experimental evaluation. Technical report CS-91-67, Department of Computer Science, Brown University, Providence, Rhode Island, October 1991.

[26] R.W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Proceedings of the Symposium in Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, Providence,Rhode Island, U.S.A., 1967.

[27] A. Gallagher and M. Bruynooghe. The derivation of an algorithm for program specialization. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, Jerusalem, Israel, pages 732–746. MIT Press, Cambridge, Massachusetts, U.S.A., June 1990.

[28] P. Granger. Static analysis of arithmetical congruences. *International Journal of Computer Mathematics*, 30:165–190, 1989.

[29] P. Granger. Static analysis of linear congruence equalities among variables of a program. In S. Abramsky and T.S.E. Maibaum, editors, *TAPSOFT'91, Proceedings of the International Joint Conference on Theory and Practice of Software Development*, Brighton, U.K., Volume 1 (CAAP'91), Lecture Notes in Computer Science 493, pages 169–192. Springer-Verlag, Berlin, Germany, 1991.

[30] R. Gupta. A fresh look at optimizing array bound checking. In *Proceedings of the ACM SIGPLAN Conference on Programmimg Language Design and Implementation*, SIGPLAN Notices 25(6):272–282, White Plains, New York, 1990.

[31] P. Hudak and J. Young. Collecting interpretations of expressions. *ACM Transactions on Programming Languages and Systems*, 13(2):269–290, April 1991.

[32] G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. Report CW 107, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 1990. (to appear in the special issue of the Journal of Logic Programming on abstract interpretation).

[33] N. D. Jones and S. S. Muchnick. Complexity of flow analysis, inductive assertion synthesis and a language due to Dijkstra. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 12, pages 380–393. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

[34] N. D. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs: abridged version. In *Conference Record of the 13th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 296–306, St. Petersburg Beach, Florida, 1986. ACM Press, New York, U.S.A.

[35] T. Kanamori and T. Kawamura. Analyzing success patterns of logic programs by abstract hybrid interpretation. Technical report 279, ICOT, Tokyo, Japan, 1987.

[36] M. Kaplan and J.D. Ullman. A general scheme for the automatic inference of variable types. *Journal of the Association for Computing Machinary*, 27(1):128–145, 1980.

[37] R. M. Keller. Formal verification of parallel programs. *Communications of the Association for*

*Computing Machinary*, 19(7):371–384, July 1976.

[38] G. Kildall. A unified approach to global program optimization. In *Conference Record of the 1st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 194–206, Boston, Massachusetts, October 1973. ACM Press, New York, U.S.A.

[39] B. Le Charlier and P. Van Hentenryck. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. In *Proceedings of the 1992 International Conference on Computer Languages*, Oakland, California, pages 137–146. IEEE Computer Society Press, Los Alamitos, California, U.S.A., April 20–23, 1992.

[40] Y. Lichtenstein, M. Codish, and E. Shapiro. Representation and enumeration of Flat Concurrent Prolog computations. In E. Shapiro, editor, *Concurrent Prolog, Collected Papers*, chapter 28, pages 197–210. MIT Press, Cambridge, Massachusetts, U.S.A., 1987.

[41] K. Marriott and H. Søndergaard. On describing success patterns of logic programs. Technical report 88/12, Department of Computer Science, University of Melbourne, Melbourne, Australia, 1988.

[42] B. Monsuez. An attempt to find polymorphic types by abstract interpretation. *BIGRE, Actes JTASPEFL'91, Bordeaux,* IRISA, Rennes, France, 74:18–26, October 1991.

[43] P. Naur. The design of the GIER ALGOL compiler. *BIT*, 3:124–140 and 145–166, 1963.

[44] P. Naur. Checking of operand types in ALGOL compilers. *BIT*, 5:151–163, 1965.

[45] F. Nielson. Tensor products generalize the relational data flow analysis method. In *Proceedings of the Fourth Hungarian Computer Science Conference*, pages 211–225, 1985.

[46] H. R. Nielson and F. Nielson. Bounded fixed-point iteration. *Journal of Logic and Computation*, 2(4):437–464, August 1992.

[47] U. Nilsson. Systematic semantic approximations of logic programs. In P. Deransart and J. Małuszyński, editors, *Proceedings of the International Workshop PLILP'90, Programming Language Implementation and Logic Programming*, Linköping, Sweden, Lecture Notes in Computer Science 456, pages 293–306. Springer-Verlag, Berlin, Germany, August 20–22, 1990.

[48] R.A. O'Keefe. Finite fixed-point problems. In J. L. Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming, Volume 2*, Melbourne, Australia, pages 749–764. MIT Press, Cambridge, Massachusetts, U.S.A., May 1987.

[49] T. Sato and H. Tamaki. Enumeration of success patterns in logic programs. *Theoretical Computer Science*, 34:227–240, 1984.

[50] M. Sintzoff. Calculating properties of programs by valuations on specific models. In *Proceedings of an ACM Conference on Proving Assertions about Programs*, Las Cruses, New Mexico, SIGPLAN Notices 7(1):203–207, January 6–7, 1972.

[51] H. Søndergaard. An application of abstract interpretation of logic programs: occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proceedings ESOP 86*, Lecture Notes in Computer Science 213, pages 327–338. Springer-Verlag, Berlin, Germany, 1986.

[52] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[53] A. Taylor. Removal of dereferencing and trailing in Prolog compilation. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, Lisbon, Portugal, pages 48–60. MIT Press, Cambridge, Massachusetts, U.S.A., June 1989.

[54] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinary*, 23(4):733–742, October 1976.

[55] P. Van Roy and A. M. Despain. The benefits of global dataflow analysis for an optimizing Prolog compiler. In S. K. Debray and M. Hermenegildo, editors, *Proceedings of the 1990 North American Conference on Logic Programming*, Austin, Texas, pages 501–515. MIT Press, Cambridge, Massachusetts, U.S.A., October 1990.

[56] A. Wærn. An implementation technique for the abstract interpretation of Prolog. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming, Volume 1*, Seattle, Washington, pages 700–710. MIT Press, Cambridge, Massachusetts, U.S.A., August 15–19, 1988.

[57] R. Warren, M. Hermenegildo, and S. K. Debray. On the practicality of global flow analysis of logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming, Volume 1*, Seattle, Washington, pages 684–699. MIT Press, Cambridge, Massachusetts, U.S.A., August 15–19, 1988.

[58] B. Wegbreit. Property extraction in well-founded property sets. *IEEE Transactions on Software*

*Engineering*, SE-1(3):270–285, September 1975.

[59] W. Winsborough. Path-dependent reachability analysis for multiple specialization. In E .L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming, Volume 1*, Cleaveland, Ohio, pages 133–153. MIT Press, Cambridge, Massachusetts, U.S.A., October 1989.

[60] W. Winsborough. Multiple specialization using minimal-function graph semantics. *Journal of Logic Programming*, 1992. (to appear in the special issue of the Journal of Logic Programming on abstract interpretation).

[61] E. Yardeni and E. Shapiro. A type system for logic programs. In E. Shapiro, editor, *Concurrent Prolog, Collected Papers*, chapter 28, pages 211–244. MIT Press, Cambridge, Massachusetts, U.S.A., 1987.