Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation

Patrick COUSOT

LIENS, École Normale Supérieure
75230 Paris cedex 05, France
cousot@dmi.ens.fr

Radhia COUSOT

LIX, CNRS & École Polytechnique
91140 Palaiseau cedex, France
rcousot@lix.polytechnique.fr

Abstract

Grammar-based program analysis à la Jones and Muchnick and set-constraint-based program analysis à la Aiken and Heintze are static analysis techniques that have traditionally been seen as quite different from abstract-interpretation-based analyses, in particular because of their apparent non-iterative nature. For example, on page 18 of [17], it is alleged that "The finitary nature of abstract interpretation implies that there is a fundamental limitation on the accuracy of this approach to program analysis. There are decidable kinds of analysis that *cannot* be computed using abstract interpretation (even with widening and narrowing). The set-based analysis considered in this thesis is one example".

On the contrary, we show that grammar and set-constraint-based program analyses are similar abstract interpretations with iterative fixpoint computation using either a widening or a finitary grammar/set-constraints transformer or even a finite domain for each particular program.

The understanding of grammar-based and set-constraint-based program analysis as a particular instance of abstract interpretation of a semantics has several advantages. First, the approximation process is formalized and not only explained using examples. Second, a domain of abstract properties is exhibited which is of general scope. Third, these analyses can be easily combined with other abstract-interpretation-based analyses, in particular for the analysis of numerical values. Fourth, they can be generalized to very powerful attribute-dependent and context-dependent analyses. Finally, a few misunderstandings may be removed.

Permission to make digital/hard copies of all or parts of this material without fee is granted provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.
FPCA '95, La Jolla, CA USA © ACM 0-89791-7/95/0006...$3.50

Conference Record of FPCA '95 SIGPLAN-SIGARCH-WG2.8 Conference on Functional Programming Languages and Computer Architecture, La Jolla, California, 25-28 June 1995, ACM Press, New York, U.S.A., pp. 170–181.

1. Introduction

We construct program analyses by abstract interpretation of a collecting semantics where abstract properties are coded using tuples of formal languages. A further approximation is to approximate such formal languages by different kinds of formal systems such as grammars or systems of set-constraints. The idea of using *regular tree grammars* for program analysis, is due to Jones and Muchnick [22, 23, 24] following Reynolds [29]. It was further developed by Aiken, Mishra, Murphy, Reddy and Sørensen [2, 27, 30], implemented by Aiken and Murphy [1] and reformulated by Heintze and Jaffar [16, 17, 19, 20] as *set-constraint-based program analysis*. We show that in such formal-language-based program analyses, the abstract semantics specifying the strongest abstract program property is defined as the least fixpoint of an abstract property transformer. This abstract semantics can be also presented as a system of constraints. For program analysis by resolution of such systems of set-constraints, no iteration is apparent. Therefore we are interested in understanding whether or not this least fixpoint is computed iteratively or approximated by non-iterative methods (e.g. by elimination). We show that actually this least fixpoint is computed by a chaotic iteration. Convergence can be enforced using either a widening, or a finitary abstract property transformer or else by choosing for each particular program an abstract domain which satisfies the ascending chain condition (and even is finite). This new point of view on grammar and set-constraint-based analysis allows us to easily combine these dependence-free analyses with existing non-grammar or non-set-constraint-based abstract interpretations and to generalize them to obtain very powerful infinitary context-dependent program analyses.

2. Standard Semantics

A *standard semantics* associates a behavior $S_{\mathrm{sd}}[\![P]\!] \in B_{\mathrm{sd}}$ to each program P of the language. This behavior is a specification of the possible program execu-

tions. Elements of the *behavior domain* B_{sd} can be higher-dimensional automata, execution traces, judgments, functions, sets of states, etc.

Example The standard semantics $\mathtt{f}_{sd}$ of the following functional program `f`:

```
f(N) =
  if (N <= 0) then
    cons(0, cons(0, cons(0, nil)))
  else let X = f(N-1) in
    cons(a(hd(X)), cons(b(hd(tl(X))),
      cons(c(hd(tl(tl(X)))), nil)));
```
(1)

(where $\mathtt{hd}(\mathtt{cons}(H,\ T)) \stackrel{\text{def}}{=} H$ and $\mathtt{tl}(\mathtt{cons}(H,\ T)) \stackrel{\text{def}}{=} T$) can be chosen as the function computed by this program, that is simply:

$$\begin{array}{rl}\mathtt{f}_{sd} \ = \ \lambda n \cdot & \text{if } n \leq 0 \text{ then} \\ & \quad \mathtt{cons}(0, \mathtt{cons}(0, \mathtt{cons}(0, \mathtt{nil}))) \\ & \text{else} \\ & \quad \mathtt{cons}(\mathtt{a}^n(0), \mathtt{cons}(\mathtt{b}^n(0), \mathtt{cons}(\mathtt{c}^n(0), \mathtt{nil})))\end{array}$$

where we use the abbreviation:

$$\mathtt{a}^i(0) \quad = \quad \underbrace{\mathtt{a}(\cdots \mathtt{a}(}_{i \text{ times}} 0) \cdots)$$

In this example, we have $B_{sd} = \mathbb{Z} \mapsto (\mathbb{L} \cup \{\bot\})$ where $\mathbb{L}$ is the set of lists and $\bot$ denotes non-termination. □

We assume that B_{sd} involves symbolic semantic values (in L) associated to indexes (in I). These symbolic semantic values are assumed to be described by sentences of a *formal language* L such that:

— the ranked *alphabet* A contains function symbols $f^n \in \Sigma$ of fixed arity $n \geq 0$;

— the language L is the set of ground terms t on A (written in prefix parenthesized notation):

$$t \quad ::= \quad f^0 \mid f^n(t_1, \ldots, t_n);$$

(L can be understood isomorphically as the algebra T_Σ of trees on the signature Σ).

Example For the program (1), the ranked alphabet is $A = \{\mathtt{0}^0, \mathtt{nil}^0, \mathtt{a}^1, \mathtt{b}^1, \mathtt{c}^1, \mathtt{cons}^2\}$. The language L is the set of ground terms t defined by the grammar:

$$t \quad ::= \quad \mathtt{0} \mid \mathtt{a}(t) \mid \mathtt{b}(t) \mid \mathtt{c}(t) \mid \mathtt{nil} \mid \mathtt{cons}(t_1, t_2)$$

Integers are simply (arbitrarily) ignored. □

The set I of *indexes* is assumed to be finite. The precision of the analysis depends upon the choice of this set I of indexes which can be program points, variables, expression labels, store locations, auxiliary variables (e.g. to denote intersections [27]), etc.

Example For the program (1), the set of indexes can be chosen to be $I = \{\mathcal{X}\}$. This (arbitrary) choice is made to compute a single global program invariant (associated to the index $\mathcal{X}$[1]). A more refined choice would consist of program points so as to compute local invariants (associated to these program points). □

[1] The name $\mathcal{X}$ is certainly not meaningful because it is associated with `f`. $\mathcal{X}$ has been preferred to other choices (such as $\mathcal{F}$) because it will later correspond to an unknown.

3. Collecting Semantics

A *collecting semantics* associates a strongest property $S_{co}\{\!|P|\!\} \in D_{co}$ to each program P of the language: $S_{co}\{\!|P|\!\} \stackrel{\text{def}}{=} \{S_{sd}[\![P]\!]\}$; Here a property is understood as the set of behaviors satisfying this property. The concrete *semantic domain* $D_{co} \stackrel{\text{def}}{=} \wp(B_{sd})$ is a complete boolean lattice $(D_{co}, \subseteq, \emptyset, B_{sd}, \cup, \cap, \neg)$. We are interested in answering questions of the form $S_{co} \subseteq P_{co}$ where $P_{co} \in D_{co}$ is a semantic property.

We assume that the collecting semantics S_{co}[2] can be expressed as the least fixpoint $S_{co} \stackrel{\text{def}}{=} \mathrm{lfp}\, F_{co}$ of a $\emptyset$-strict complete $\cup$-morphism *concrete property transformer* $F_{co} \in D_{co} \mapsto D_{co}$[3]. Usually F_{co} is defined by induction on the syntax of the program, an aspect that we neglect here for simplicity.

D_{co} usually formalizes program properties as the set of elements having this property. Many choices are possible. A few simple examples of collecting semantics for formal-language program analysis of functional, imperative, parallel and logic programs are considered below.

Example The collecting semantics $\mathtt{f}_{co}$ for the functional program (1) can be chosen as the set of possible functions computed by this program ignoring possible non-termination, that is simply:

$$\begin{array}{rl}\mathtt{f}_{co} \ = \ \{\lambda n \cdot & \text{if } n \leq 0 \text{ then} \\ & \quad \mathtt{cons}(0, \mathtt{cons}(0, \mathtt{cons}(0, \mathtt{nil}))) \\ & \text{else} \\ & \quad \mathtt{cons}(\mathtt{a}^n(0), \mathtt{cons}(\mathtt{b}^n(0), \mathtt{cons}(\mathtt{c}^n(0), \mathtt{nil})))\}\end{array}$$

In this example, we have $D_{co} = \wp(\mathbb{Z} \mapsto \mathbb{L})$. A program property is therefore chosen to be an invariant $I \in D_{co}$. The collecting semantics of a program P is the strongest property of P, that is the singleton set $\mathtt{f}_{co} = \{\mathtt{f}_{sd}\} \in D_{co}$[4]. This allows us to formalize "P has property I" as $\mathtt{f}_{co} \subseteq I$ that is equivalently $\mathtt{f}_{sd} \in I$. □

Example The collecting semantics of the following imperative program:

```
{1}  X := cons(0, cons(0, cons(0, nil)));
{2}  while N > 0 do begin
{3}    X := cons(a(hd(X)), cons(b(hd(tl(X))),
                 cons(c(hd(tl(tl(X)))), nil)));
{4}    N := N - 1;
{5}  end;
{6}
```
(2)

associates to each program point p the set $\rho_{co}[p]$ of pairs $\langle n,\ x\rangle$ of possible values of variables `N` and `X`:

$$\rho_{co}[1] \quad = \quad \{\langle n,\ x\rangle : n \in \mathbb{Z} \wedge x \in \mathbb{L}\}$$

[2] In what follows we write S_{co} for $S_{co}\{\!|P|\!\}$ thus leaving the involved program P implicit.

[3] The general situation in abstract interpretation is more complicated since the *approximation ordering* (for comparing program properties by implication) and the *computational ordering* (for computing fixpoints) do not coincide (see [9]). This situation does not appear for the *invariance properties* considered here.

[4] For more details on collecting semantics, see e.g. [9].