Set Constraints: a Pearl in Research on Constraints

Leszek Pacholski$^1$ and Andreas Podelski$^2$

$^1$ Institute of Computer Science, University of Wrocław
Przesmyckiego 20, PL-51-151 Wrocław, Poland
www.tcs.uni.wroc.pl/~pacholsk
pacholsk@tcs.uni.wroc.pl
$^2$ Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
www.mpi-sb.mpg.de/~podelski
podelski@mpi-sb.mpg.de

Abstract. The topic of set constraints is a pearl among the research topics on constraints. It combines theoretical investigations (ranging from logical expressiveness, decidability, algorithms and complexity analysis to program semantics and domain theory) with practical experiments in building systems for program analysis, addressing questions like implementation issues and scalability. The research has its direct applications in type inference, optimization and verification of imperative, functional, logic and reactive programs.

1 Introduction

Set constraints are first-order logic formulas interpreted over the domain of sets of trees. These sets of trees are possibly recursively defined. The first-order theory that they form is interesting on its own right. Essentially, we study it because the problem of computationally reasoning about sets (of trees) is fundamentally important. Thus, research on set constraints can be fundamental research.

Research on set constraints can also be applied research. This is because set constraints form the algorithmic basis for a certain kind of program analysis that is called set-based. Here, the problem of reasoning about runtime properties of a program is transferred to the problem of solving set constraints. Several systems have been built, each addressing a particular program analysis problem (obtained, for example, by the restriction to a particular class of programs). The latter means to single out a subclass of set constraints that fits with the analysis problem and to build a system solving set constraints in this subclass (efficiently).

In the next two sections, we will survey results that cover both these aspects of research on set constraints.
2 Constraint solving

The history of set constraints and set-based program analysis goes back to Reynolds [83] in 1969. He was the first to derive recursively defined sets as approximations of runtime values from programs, here first-order functional programs. Jones and Muchnick [61] had a similar idea in 1979 and applied it to imperative programs with data constructors like cons and nil (and data destructors like car and cdr). The set constraints used in these approaches were rather inexpressive. It was only in the nineties when people crystallized the problem of solving set constraints and studied it systematically.

Heintze and Jaffar [53] coined the term of set constraints in 1990 and formulated the general problem (schema) which has occupied a number of people since then: is the satisfiability of inclusions between set expressions decidable, when these set expressions are built up by

- variables (interpreted over sets of trees),
- tree constructors (interpreted as functions over sets of trees),
- a specific choice of set operators, Boolean and possibly others.

Assume given a signature $\Sigma$ fixing the arity of the function symbols $f$, $g$, $a$, $b$... and defining the set $T_\Sigma$ of trees. The symbol $f$ denotes a function over trees, $f : (T_\Sigma)^n \to T_\Sigma$, $\bar{t} \mapsto f(\bar{t})$ (where $\bar{t} = (t_1, \ldots, t_n)$ is a tuple of length $n \geq 0$ according to the arity of $f$). By the canonical extension of this function to sets $M_1, \ldots, M_n \in 2^{T_\Sigma}$,

$$f(M_1, \ldots, M_n) = \{f(t_1, \ldots, t_n) \mid t_1 \in M_1, \ldots, t_n \in M_n\},$$

the symbol $f$ denotes also an operator over sets of trees, $f : (2^{T_\Sigma})^n \to 2^{T_\Sigma}$, $M \mapsto f(M)$. The “inverse” of this operator is the projection of $f$ to the $k$-th argument,

$$f^{-1}_k(M) = \{t \in T_\Sigma \mid \exists t_1, \ldots, t_k \text{ s.t. } f(t_1, \ldots, t_n) \in M\}.$$

A general set expression $e$ is built up by: variables (that range over $2^{T_\Sigma}$), function symbols, the Boolean set operators and the projection operator [53]. If $e$ does not contain the complement operator, then $e$ is called a positive set expression. A general set constraint $\varphi$ is a conjunction of inclusions of the form $e \subseteq e'$.

The full class of general set constraints is not motivated by a program analysis problem. Note that, generally, $\varphi$ does not have a least or greatest solution.

Heintze and Jaffar [53] also gave the first decidability result for a class of set constraints that they called definite, for the reason that all satisfiable constraints in the class have a least solution (the class seems to be the largest one having this property). A definite set constraint is a conjunction of inclusions $e_l \subseteq e_r$ between positive set expressions, where the set expressions $e_r$ on the right-hand side of $\subseteq$ are furthermore restricted to contain only variables, constants and function symbols and the intersection operator (that is, no projection or union). This class is used for the set-based analysis of logic programs [54].
Two years later, in 1992, Aiken and Wimmers [8] proved the decidability for the class of positive set constraints (in NEXPTIME). Their definition is so natural (the choice of set operators are exactly the Boolean ones) that the term set constraints is often used to refer to this class. Ganzinger assisted Aiken’s presentation at LICS’92 and, during the talk, he recognized that this class is equivalent to a certain first-order theory called the monadic class. One can test the satisfiability of a set constraint \( \varphi \) by transforming \( \varphi \) into a so-called flat clause, which is a skolemized form of a formula of the monadic class and for which a decision procedure based on ordered resolution exists [9]. Thus, the history of set constraints goes in fact back to 1915 when Löwenheim [69] gave the first decision procedure for the monadic class. The proof by Ackermann [1] of the same result gives an algorithm that appears to be usable in practice. The equivalence between set constraints and the monadic class lead Bachmair, Ganzinger and Waldmann [11] to give a lower bound and thus characterize the complexity of the satisfiability problem, namely by NEXPTIME. Aiken, Kozen, Vardi and Wimmers [3] gave the detailed analysis of the complexity of the set-constraint solving problem depending on the given signature of function symbols. Later, the decidability, and with it that same complexity result, was extended to richer classes of set constraints with negation [45,4,88,19] and then with projection by Charatonik and Pacholski [20] (which settled the open problem for the general class formulated by Heintze and Jaffar [53]). Set constraints were also studied from the logical and topological point of view [63,25,62,22] and also in domains different from the Herbrand universe [50,17,71]. Kozen [64] explores the use of set constraints in constraint logic programming. Uribe [92] uses set constraints in order-sorted languages. Seyneye, Tommasi and Treinen [85] showed that the \( \exists^* \forall^* \)-fragment of the theory of set constraints is undecidable.

Charatonik [17,18] studied set constraints in the presence of additional equational axioms like associativity or commutativity. It turns out that in the most interesting cases (associativity, associativity together with commutativity) the satisfiability problem becomes undecidable. McAllester, Givan, Witty and Kozen [71] liberalized the notions of set constraints to so-called Tarskian set constraints over arbitrary first-order domain, with a link to modal-logics. Recently, Charatonik and Poledski [24] singled out set constraints with intersection (the choice of set operators includes only the intersection), shown that they are equivalent to definite set constraints, and gave the first DEXPTIME characterization for set constraints. They have also defined co-definite set constraints (which have a greatest solution, if satisfiable) and shown the same complexity for this class [23,81]. A co-definite set constraint is a conjunction of inclusions \( e_I \sqsubseteq e_r \) between positive set expressions, where the set expressions \( e_r \) on the left-hand side of \( \sqsubseteq \) are furthermore restricted to contain only variables, constants, unary function symbols and the union operator (that is, no projection, intersection or function symbol of arity greater than one). Recently, Devienne, Talbot and Tison [30] have improved the algorithms for the two classes of definite and co-definite set constraints (essentially, by adding strategies); although the theo-
retical complexity does not change, an exponential gain can be obtained in some cases.

The DEXPTIME lower bound can be expected for any class of of set constraints that can express regular sets of trees, since conjunction corresponds to intersection (and since the emptiness of intersection of $n$ tree automata is DEXPTIME-hard [36,84]). Note that there is a close relation between (certain classes of) set constraints and two-way alternating tree automata [36,24,81,16,86,94,93,40,78] (cf., however, also the formalization of a connection with 2NPDA's by Heintze and McAllester [59]).

To give some intuition, we will translate the tree automaton with the transitions below (over the alphabet with the constant symbol $\emptyset$ and the unary symbol $s$; note that a string automaton is the special case of a tree automaton over unary function symbols and a constant symbol)

$$
\begin{align*}
\text{init} & \rightarrow \emptyset \\
 x & \rightarrow y \\
 y & \rightarrow x
\end{align*}
$$

first into the regular tree grammar [40]

$$
\begin{align*}
x & \rightarrow \emptyset \\
x & \rightarrow s(y) \\
y & \rightarrow s(x)
\end{align*}
$$

and from there into the regular systems of equations [10,40]

$$
\begin{align*}
x & = s(y) \cup \emptyset \\
y & = s(x).
\end{align*}
$$

We observe that regular systems of equations have:

- variables interpreted as sets of trees,
- tree constructors applied on sets of trees, and
- the Boolean set operator "union".

We generalize regular systems of equations to set constraints by

- replacing equality "$=$" with inclusion "$\subseteq$",
- allowing composed terms on both sides of "$\subseteq$" (which introduces the "two-way" direction of the automata),
- adding more set operators, Boolean and others (roughly, alternation accounts for intersection on the right hand side of "$\subseteq$").

Many set constraints algorithm have to deal with the special role played by the empty set. Namely, when testing the satisfiability of, for example, the set constraint

$$
\varphi \land f(a,y) \subseteq f(b,y'),
$$
we can derive $a \subseteq b$ (and, thus, \textit{false}, showing that the set constraint is not satisfiable) only after we have derived \textit{"y is nonempty"} from the rest constraint $\varphi$. Otherwise, if the value of \textit{y} in a solution $a$ of $\varphi$ can be $\emptyset$, then $f(\{a\}, a(y)) = \{ f(a, t_2) \mid t_2 \in \emptyset \} = \emptyset$ and the inclusion $f(a, y) \subseteq f(b, y')$ is satisfied. It thus seems natural to investigate the satisfiability problem when the empty set is excluded from the domain [76,22,24]. It turns out that \textit{nonempty-set constraints} have interesting algorithmic properties [75] and logic properties, such as the fundamental property of independence for set constraints with intersection [22,24].

3  Set-based analysis

Before we survey results, we will give some intuition. We obtain the \textit{set-based abstract semantics} of a program by executing the program with \textit{set} environments. A set environment at program point \textit{[1]} assigns each variable \textit{x} a set $x_{[1]}$ of values. That is, for the abstract semantics, we replace the pointwise environments $\langle$program point$\rangle \mapsto $\textit{<runtime value>} of the concrete semantics by set environments: $\langle$program point$\rangle \mapsto $\textit{<set of runtime values>}. A set constraint expresses the relation between these sets $x_{[1]}$.

We take, for example, the set-based analysis of an imperative programming language with data constructors (e.g., \textsf{cons} for lists) [49,64]. If \textit{[1]} and \textit{[2]} are the two program points before and after the instruction

$$\begin{align*}
{\textit{[1]} } & \quad x := \textsf{cons}(y,x) \\
{\textit{[2]} } &
\end{align*}$$

then the derived set constraint is

$$x_{[2]} = \textsf{cons}(y_{[1]}; x_{[1]}) \land y_{[2]} = y_{[1]};$$

which expresses naturally the relation between the sets of possible values of \textit{x} and \textit{y} at the two program points.

The following example program illustrates that the set-based analysis ignores dependencies between variables in the pointwise environments. If the set $x_{[1]}$ contains two elements, then the set $x_{[2]}$ will contain four.

$$\begin{align*}
{\textit{[1]} } & \quad y := \textsf{car}(x) \\
 & \quad z := \textsf{cdr}(x) \\
 & \quad x := \textsf{cons}(y,z) \\
{\textit{[2]} } &
\end{align*}$$

In summary, in set-based analysis, one expresses the abstract semantics of a program $\mathit{P}$ by a set constraint $\varphi_{\mathit{P}}$ (which is obtained by a syntactical transformation
of the program text) and then computes the abstract semantics of $P$ by solving
$\varphi_P$; the latter means to compute effective representations of the values for $x_{[a]}$
under a distinguished (the least, or the greatest) solution of $\varphi_P$. The values are
approximations of the sets of runtime values at $x$. The two steps correspond to
the specification of the analysis and to its implementation, respectively.

We next analyse a small example program.

```plaintext
while x /= nil do
  i := i+1
  x := cdr(x)
```

The derived set constraint is

$$\neg\text{il} \subseteq x \land x \subseteq \text{cons}^{-1}(x)$$

whose solved form is

$$\neg\text{il} \cup [T|x] \subseteq x$$

with the set of all lists as the value for $x$ under the least solution.

To give an example of a set-based analysis of a reactive program, we take the
following definition of a procedure in Oz [87],

```plaintext
proc {P X I}
  local Y in
  X=I|Y
  {P Y (I+1)}
end
 end
```

The derived set constraint is

$$x \subseteq \text{cons}(T,y) \land y \subseteq x$$

with the solved form

$$x \subseteq [T|x]$$

whose greatest solution assigns $x$ the set of all infinite lists.

The analysis of logic programs is one area of application of set constraints where
systems were and are being built. The area was developed mainly due to the
work of Heintze and Jaffar [34,56,49,57]. Other research groups, for example in
Bristol [38,37] and also in Saarbrücken, are now building systems too. Heintze
and Jaffar started by observing the lack of a formal definition of set-based ap-
proximation in the earlier work of Mishra [72] and Yardeni and Shapiro [97,98].
They gave such a definition for logic programs in terms of the $T_P$ operator, which
is obtained from the $T_F$ operator by replacing substitutions with set-valued sub-
stitutions (later, McAllester and Heintze [70] gave such a definition for functional
languages). The $T_P$ operator formalizes the intuition of set environments given
above. They gave an equivalent characterization of the set-based approximation of the logic program $P$ via a transformation of $P$ to another logic program $P'$. They were probably the first to look at decidability issues (most of the previous works just had various ad hoc algorithms). Namely, is the least fixpoint of the $T_{F}$ operator, or, equivalently, the least model of $P'$, effectively representable (such that, for example, emptiness is decidable)? The effective representation is, as mentioned above, by tree automata, whose emptiness test is linear. Frühwirth, Shapiro, Vardi and Yardeni [36] present a set-based analysis with (a restricted class of) logic programs and showed the DEXPTIME-completeness of the problem of membership (i.e., of a ground atom in the set-based approximation).

Logic programs are also set constraints in the sense that they express a relation between sets of trees (namely, the denotations of the predicates in models of the program). There is also recent work on the set-based analysis of reactive infinite-state systems that are specified by logic programs [81]; here, definite and co-definite set constraints are derived from logic programs with oracles in order to approximate temporal logic properties of possibly infinite program executions. That work also yields that the analysis of Mishra [72] is so weak that it approximates even the greatest model of a logic program.

The standard Hindley-Milner type system is extended by type inference systems based on set constraints (soft typing) and it can be weakened even further to provide a family of set based safety analysis. Early work in this domain was done by Mishra and Reddy [73] and Thatte [90] and was extended by many researchers including Aiken, Wimmers and Lakshman [5,7,2,9], Palsberg and O’Keefe [79], Palsberg and Schwartzbach [80]. McAllester and Heintze [70] systematized the notion of set-based analyses of functional languages and gave a thorough study of its complexity. Heintze and McAllester [58,59] address the complexity of control-flow analysis, which is at the heart of set-based analyses for ML. Cousot and Cousot [27] showed that set-based analysis can be seen as an instance of an abstract interpretation in the sense that the process of solving a set constraint is isomorphic to the iteration of an appropriate fixpoint operator (defining an abstract program semantics).

The work on Tarskian set constraints [71] employs different constraint solving techniques and has applications different from program analysis; this area has much in common with the areas of artificial intelligence, model checking and the $\mu$-calculus.

Several set-based analysis systems have been built. The origins of inefficiency and other insufficiencies in early systems have by now been recognized. The language and CLP(SC) of Kozen [64] and Foster [33] allows one to easily prototype a set-based analysis system. The systems built by Aiken and Fähndrich [33] at Berkeley and by Heintze [52] at Bell Labs perform the analysis of functional programs of several thousand lines of code in acceptable time. The concentrated effort on the set constraint solving problem was the precondition for the existence of such systems.

Acknowledgements. We thank Witold Charatonik for helpful comments.
References


