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 programs 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 resesearch 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 Σ fixing the arity of the function symbols f, g, a, $b \ldots$ and defining the set T_Σ 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 $M_1, \ldots, M_n \in 2^{T_\Sigma}$,

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

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

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

A *general* set expression e is built up by: variables (that range over 2^{T_Σ}), 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 φ 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, φ 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 satsifiability of a set constraint φ by transforming φ 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 [62]. 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 [63,25,65,22] 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 constaints in order-sorted languages. Seynhave, 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 Podelski [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 $e_l \subseteq e_r$ between positive set expressions, where the set expressions e_l on the left-hand side of $\subseteq$ 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 theoretical 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 0 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)

$$\begin{aligned} \mathsf{init} &\xrightarrow{0} x \\ x &\xrightarrow{s} y \\ y &\xrightarrow{s} x \end{aligned}$$

first into the regular tree grammar [40]

$$\begin{aligned} &x \to 0 \\ &x \to s(y) \\ &y \to s(x) \end{aligned}$$

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

$$\begin{aligned} &x = s(y) \cup 0 \\ &y = s(x). \end{aligned}$$

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

$$\varphi \wedge f(a, y) \subseteq f(b, y'),$$

we can derive $a \subseteq b$ (and, thus, *false*, showing that the set constraint is not satisfiable) only after we have derived "y is nonempty" from the rest constraint φ. Otherwise, if the value of y in a solution α of φ can be $\emptyset$, then $f(\{a\}, \alpha(y)) = \{f(a, t_2) \mid t_2 \in \emptyset\} = \emptyset$ 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 *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 *set-based abstract semantics* of a program by executing the program with *set* environments. A set environment at program point $_{[1]}$ assigns each variable x a set $x_{[1]}$ of values. That is, for the abstract semantics, we replace the pointwise environments `<program point>` $\mapsto$ `<runtime value>` of the concrete semantics by set environments: `<program point>` $\mapsto$ `<set of runtime values>`. A set constraint expresses the relation between these sets $x_{[k]}$.

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

```
[1]
   x := cons(y,x)
[2]
```

then the derived set constraint is

$$\mathtt{x}_{[2]} = \mathtt{cons}(\mathtt{y}_{[1]},\ \mathtt{x}_{[1]}) \wedge \mathtt{y}_{[2]} = \mathtt{y}_{[1]},$$

which expresses naturally the relation between the sets of possible values of `x` and `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 $\mathtt{x}_{[1]}$ contains two elements, then the set $\mathtt{x}_{[2]}$ will contain four.

```
[1]
   y := car(x)
   z := cdr(x)
   x := cons(y,z)
[2]
```

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

We next analyse a small example program.

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

The derived set constraint is

$$nil \subseteq x \wedge x \subseteq cons^{-1}_{(2)}(x)$$

whose solved form is

$$nil \cup [\top|x] \subseteq x$$

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].

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

The derived set constraint is

$$x \subseteq cons(\top, y) \wedge y \subseteq x$$

with the solved form

$$x \subseteq [\top|x]$$

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 [54,56,49,57]. 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 approximation* 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 $\mathcal{T}_P$ operator, which is obtained from the T_P operator by replacing substitutions with set-valued substitutions (later, McAllester and Heintze [70] gave such a definition for functional languages). The $\mathcal{T}_P$ 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 $\mathcal{T}_\mathcal{P}$ 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 artifical intelligence, model checking and the μ-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 system 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

1. W. Ackermann. *Solvable Cases of the Decision Problem*. North-Holland, Amsterdam, 1954.
2. A. Aiken. Set constraints: Results, applications and future directions. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, LNCS 874, pages 326–335. Springer-Verlag, 1994.
3. A. Aiken, D. Kozen, M. Y. Vardi, and E. L. Wimmers. The complexity of set constraints. In *1993 Conference on Computer Science Logic*, LNCS 832, pages 1–17. Springer-Verlag, Sept. 1993.
4. A. Aiken, D. Kozen, and E. L. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30–44, Oct. 1995.
5. A. Aiken and B. Murphy. Implementing regular tree expressions. In *ACM Conference on Functional Programming Languages and Computer Architecture*, pages 427–447, August 1991.
6. A. Aiken and B. Murphy. Static type inference in a dynamically typed language. In *Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 279–290, January 1991.
7. A. Aiken and E. Wimmers. Type inclusion constraints and type inference. In *Proceedings of the 1993 Conference on Functional Programming Languages and Computer Architecture*, pages 31–41, Copenhagen, Denmark, June 1993.
8. A. Aiken and E. L. Wimmers. Solving systems of set constraints (extended abstract). In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 329–340, 1992.
9. A. Aiken, E. L. Wimmers, and T. Lakshman. Soft typing with conditional types. In *Twenty-First Annual ACM Symposium on Principles of Programming Languages*, Portland, Oregon, Jan. 1994.
10. D. Arden. Delayed logic and finite state machines. In *Proceedings of the 2nd Annual Symposium on Switching Circuit Theory and Logical Design*, pages 133–151, 1961.
11. L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83, 1993.
12. C. Beeri, S. Nagvi, O. Schmueli, and S. Tsur. Set constructors in a logic database language. *The Journal of Logic Programming*, pages 181–232, 1991.
13. S. K. Biswas. A demand-driven set-based analysis. In *The 24th ACM Symposium on Principles of Programming Languages POPL '97*, pages 372–385, Paris, France, January 1997.
14. B. Bogaert and S. Tison. Automata with equality tests. Technical Report IT 207, Laboratoire d'Informatique Fondamentale de Lille, 1991.
15. P. Bruscoli, A. Dovier, E. Pontelli, and G. Rossi. Compiling intensional sets in CLP. In *Proceedings of the International Conference on Logic Programming*, pages 647–661. The MIT Press, 1994.
16. J. Brzozowski and E. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theorical Computer Science*, 10:19–35, 1980.
17. W. Charatonik. Set constraints in some equational theories. In *1st International Conference Constraints in Computational Logics*, LNCS 845, pages 304–319. Springer-Verlag, 1994. Also to appear in *Information and Computation*.
18. W. Charatonik. Set constraints in some equational theories. PhD thesis, Polish Academy of Sciences, 1995.

19. W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 128–136, 1994.
20. W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symposium on Foundations of Computer Science*, pages 642–653, 1994.
21. W. Charatonik and L. Pacholski. Negative set constraints: an easy proof of decidability. Technical Report MPI-I-93-265, Max-Planck Institute für Informatik, December 1993.
22. W. Charatonik and A. Podelski. The independence property of a class of set constraints. In *Conference on Principles and Practice of Constraint Programming*, LNCS 1118, pages 76–90. Springer-Verlag, 1996.
23. W. Charatonik and A. Podelski. Set constraints for greatest models. Technical Report MPI-I-97-2-004, Max-Planck-Institut für Informatik, April 1997. http://www.mpi-sb.mpg.de/~podelski/papers/greatest.html.
24. W. Charatonik and A. Podelski. Set constraints with intersection. In G. Winskel, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 362–372. IEEE, June 1997.
25. A. Cheng and D. Kozen. A complete Gentzen-style axiomatization for set constraints. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, LNCS 1099, 1996.
26. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Proc. POPL '92*, pages 83–94. ACM Press, 1992.
27. P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Record of FPCA '95 - Conference on Functional Programming and Computer Architecture*, pages 170–181, La Jolla, California, USA, 25-28 June 1995. SIGPLAN/SIGARCH/WG2.8, ACM Press, New York, USA.
28. F. M. Damm. Subtyping with union types, intersection types and recursive types. In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software (TACS'94). Proceedings of the International Symposium, Sendai, Japan, April 19-22, 1994.*, volume 789 of *LNCS*, pages 687–706, Berlin, April 1994. Springer-Verlag.
29. P. W. Dart and J. Zobel. *A regular type language for logic programs*, chapter 5, pages 157–188. MIT Press, 1992.
30. P. Devienne, J.-M. Talbot, and S. Tison. Solving classes of set constraints with tree automata. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming - CP97*, LNCS, Berlin, Germany, October 1997. Springer-Verlag. To appear.
31. A. Dovier. *Computable Set Theory and Logic Programming.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1996.
32. A. Dovier and G. Rossi. Embedding Extensional Finite Sets in CLP. In *International Logic Programming Symposium*, 1993.
33. M. Fähndrich and A. Aiken. Making set-constraint based program analyses scale. Computer Science Division Tech Report 96-917, Computer Science Division, University of California at Berkeley, September 1996. Also presented at the Workshop on Set Constraints, Cambridge MA, August 1996.
34. J. S. Foster. CLP(SC): Implementation and efficiency considerations. workshop on set constraints. Available at `http://http.cs.berkeley.edu/~jfoster/`. Presented at the Workshop on Set Constraints, Cambridge MA, August 1996, August 1996.

35. J. S. Foster, M. Fähndrich, and A. Aiken. Flow-insensitive points-to analysis with term and set constraints. Computer Science Division Tech Report 97-964, , University of California at Berkeley, September 1997. Also presented at the Workshop on Set Constraints, Cambridge MA, August 1996.
36. T. Frühwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300–309, July 1991.
37. J. P. Gallagher, D. Boulanger, and H. Saglam. Practical model-based static analysis for definite logic programs. In *Proceedings of the 1995 International Symposium on Logic Programming*, pages 351–368.
38. J. P. Gallagher and L. Lafave. Regular approximation of computation paths in logic and functional languages. In *Proceedings of the Dagstuhl Workshop on Partial Evaluation*, pages 1–16. Springer-Verlag, February 1996.
39. K. L. S. Gasser, F. Nielson, and H. R. Nielson. Systematic realisation of control flow analyses for CML. In *Proceedings of ICFP'97*, pages 38–51. ACM Press, 1997.
40. F. Gécseg and M. Steinby. *Tree Automata*. Akademiai Kiado, Budapest, 1984.
41. C. Gervet. Conjunto: Constraint logic programming with finite set domains. In M. Bruynooghe, editor, *Proceedings of the International Logic Programming Symposium*, pages 339–358, 1994.
42. C. Gervet. *Set Intervals in Constraint-Logic Programming: Definition and Implementation of a Language*. PhD thesis, Université de Franche-Compté, 1995. European Thesis.
43. C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints*, 1(2), 1997.
44. R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In *10th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 665, pages 505–514. Springer-Verlag, 1993.
45. R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the* 34^{th} *Symp. on Foundations of Computer Science*, pages 372–380, 1993. Full version in [46].
46. R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. Technical Report IT 247, Laboratoire d'Informatique Fondamentale de Lille, 1993.
47. R. Gilleron, S. Tison, and M. Tommasi. Set constraints and automata. Technical Report IT 292, Laboratoire d'Informatique Fondamentale de Lille, 1996.
48. R. Gilleron, S. Tison, and M. Tommassi. Some new decidability results on positive and negative set constraints. In *1st International Conference Constraints in Computational Logics*, LNCS 845, pages 336–351. Springer-Verlag, 1994.
49. N. Heintze. Set based program analysis. PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.
50. N. Heintze. Set based analysis of arithmetic. Draft manuscript, July 1993.
51. N. Heintze. Set based analysis of ML programs. Technical Report CMU–CS–93–193, School of Computer Science, Carnegie Mellon University, July 1993.
52. N. Heintze. Set based analysis of ML programs. In *Conference on Lisp and Functional Programming*, pages 306–317. ACM, 1994. Preliminary version in [51].
53. N. Heintze and J. Jaffar. A decision procedure for a class of set constraints (extended abstract). In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, 1990.

54. N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 197–209, January 1990.
55. N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. Technical report, School of Computer Science, Carnegie Mellon University, Aug. 1990. 66 pages.
56. N. Heintze and J. Jaffar. An engine for logic program analysis. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 318–328, 1992.
57. N. Heintze and J. Jaffar. *Semantic Types for Logic Programs*, chapter 4, pages 141–156. MIT Press, 1992.
58. N. Heintze and D. McAllester. Linear-time subtransitive control-flow analysis. In *ACM Conference on Programming Language Design and Implementation*, 1997. To appear.
59. N. Heintze and D. McAllester. On the cubic bottleneck in subtyping and flow analysis. In G. Winskel, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 342–351. IEEE, June 1997.
60. ILOG. ILOG SOLVER 3.2 user manual. www.ilog.com.
61. N. D. Jones and S. S. Muchnick. Flow analysis and optimization of lisp-like structures. In *Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 244–256, January 1979.
62. W. Joyner. Resolution strategies as decision procedures. *Journal of the Association for Computing Machinery*, 23(3):398–417, 1979.
63. D. Kozen. Logical aspects of set constraints. In *1993 Conference on Computer Science Logic*, LNCS 832, pages 175–188. Springer-Verlag, Sept. 1993.
64. D. Kozen. Set constraints and logic programming (abstract). In *1st International Conference Constraints in Computational Logics*, LNCS 845. Springer-Verlag, 1994. Also to appear in *Information and Computation*.
65. D. Kozen. Rational spaces and set constraints. In *TAPSOFT: 6th International Joint Conference on Theory and Practice of Software Development*, LNCS 915, pages 42–61. Springer-Verlag, 1995.
66. D. Kozen. Rational spaces and set constraints. *Theoretical Computer Science*, 167(1–2):73–94, October 1996.
67. G. Kuper. Logic programming with sets. New York, NY, 1990. Academic Press.
68. B. Legeard and E. Legros. Short Overview of the CLPS System. 1991.
69. L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:228–251, 1915.
70. D. McAllester and N. Heintze. On the complexity of set-based analysis. www.ai.mit.edu/people/dam/setbased.ps.
71. D. A. McAllester, R. Givan, C. Witty, and D. Kozen. Tarskian set constraints. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 138–147, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.
72. P. Mishra. Towards a theory of types in Prolog. In *IEEE International Symposium on Logic Programming*, pages 289–298, 1984.
73. P. Mishra and U. Reddy. Declaration-free type checking. In *Twelfth Annual ACM Symposium on the Principles of Programming Languages*, pages 7–21, 1985.
74. M. Müller. *Type Analysis for a Higher-Order Concurrent Constraint Language*. PhD thesis, Universität des Saarlandes, Technische Fakultät, 66041 Saarbrücken, Germany, expected 1997.

75. M. Müller, J. Niehren, and A. Podelski. Inclusion constraints over non-empty sets of trees. In M. Bidoit and M. Dauchet, editors, *Proceedings of the 9th International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 1214 of *LNCS*, pages 345–356, Berlin, April 1997. Springer-Verlag.
76. M. Müller, J. Niehren, and A. Podelski. Ordering constraints over feature trees. In G. Smolka, editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming - CP97*, LNCS, Berlin, Germany, October 1997. Springer-Verlag. To appear.
77. T. Müller and M. Müller. Finite set constraints in Oz. In *13. Workshop Logische Programmierung*, Technische Universität München, September 1997. to appear.
78. M. Nivat and A. Podelski. *Tree Automata and Languages*. North-Holland, Amsterdam, 1992.
79. J. Palsberg and O'Keefe. A type system equivalent to flow analysis. In *Symposium of Principles of Programming Languages*, pages 367–378. ACM, 1995.
80. J. Palsberg and M. Schwartzbach. Safety analysis versus type inference. *Information and Computation*, 118(1):128–141, April 1995.
81. A. Podelski, W. Charatonik, and M. Müller. Set-based analysis of reactive infinite-state systems. Submitted for publication, 1997.
82. J. F. Puget. Finite Set Intervals. In *Proceedings of the Second International Workshop on Set Constraints*, Cambridge, Massachusetts, 1996.
83. J. C. Reynolds. Automatic computation of data set definitions. *Information Processing*, 68:456–461, 1969.
84. H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52:57–60, 1994.
85. F. Seynhave, M. Tommasi, and R. Treinen. Grid structures and undecidable constraint theories. In M. Bidoit and M. Dauchet, editors, *Proceedings of the 9th International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 1214 of *LNCS*, pages 357–368, Berlin, April 1997. Springer-Verlag.
86. G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.
87. G. Smolka. The Oz programming model. In *Volume 1000 of Lecture Notes in Computer Science*, 1995.
88. K. Stefansson. Systems of set constraints with negative constraints are *NEXPTIME*-complete. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 137–141, 1994.
89. F. Stolzenburg. Membership-constraints and complexity in logic programming with sets. In F. Baader and K. U. Schulz, editors, *Frontiers in Combining Systems*, pages 285–302. Kluwer Academic, Dordrecht, The Netherlands, 1996.
90. S. Thatte. Type inference with partial types. In *15th International Colloquium on Automata, Languages and Programming*, volume 317 of *LNCS*, pages 615–629. Springer-Verlag, 1988.
91. W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. Elsevier, 1990.
92. T. E. Uribe. Sorted unification using set constraints. In *11th International Conference on Automated Deduction*, LNAI 607, pages 163–177. Springer-Verlag, 1992.
93. M. Y. Vardi. An automata-theoretic approach to linear-temporal logic. *In Logics for Concurrency: Structure versus Automata. LNCS*, 1043:238–266, 1996.
94. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32, 1986.

95. C. Walinsky. CLP(Σ^*): Constraint Logic Programming with Regular Sets. In *Proceedings of the International Conference on Logic Programming*, pages 181–190, 1989.
96. E. Yardeni, , T. Frühwirth, and E. Shapiro. *Polymorphically typed logic programs*, chapter 2, pages 157–188. MIT Press, 1992.
97. E. Yardeni. A type system for logic programs. Master's thesis, Weizmann Institute of Science, 1987.
98. E. Yardeni and E. Shapiro. *A type system for logic programs*, volume 2, chapter 28, pages 211–244. The MIT Press, 1987.
99. E. Yardeni and E. Shapiro. A type system for logic programs. *Journal of Logic Programming*, 10:125–153, 1991. Preliminary version in [98].
100. J. Young and P. O'Keefe. Experience with a type evaluator. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation*, pages 573–581. North-Holland, 1988.