Abstract Interpretation Based Formal Methods and Future Challenges (Extended Electronic Version)

Patrick Cousot

École normale supérieure, Département d'informatique,
45 rue d'Ulm, 75230 Paris cedex 05, France
Patrick.Cousot@ens.fr http://www.di.ens.fr/~cousot/

Abstract. In order to contribute to the solution of the software reliability problem, tools have been designed to analyze statically the run-time behavior of programs. Because the correctness problem is undecidable, some form of approximation is needed. The purpose of *abstract interpretation* is to formalize this idea of *approximation*. We illustrate informally the application of abstraction to the *semantics* of programming languages as well as to *static program analysis*. The main point is that in order to reason or compute about a complex system, some information must be lost, that is the observation of executions must be either partial or at a high level of abstraction.

In the second part of the paper, we compare static program analysis with deductive methods, model-checking and type inference. Their foundational ideas are briefly reviewed, and the shortcomings of these four methods are discussed, including when they should be combined. Alternatively, since program debugging is still the main program verification method used in the software industry, we suggest to combine formal with informal methods.

Finally, the grand challenge for all formal methods and tools is to solve the software reliability, trustworthiness or robustness problems. A few challenges more specific to static program analysis by abstract interpretation are briefly discussed.

1 Introductory Motivations

The evolution of hardware by a factor of 10^6 over the past 25 years has lead to the explosion of the size of programs in similar proportions. The scope of application of very large programs (from 1 to 40 millions of lines) is likely to widen rapidly in the next decade. Such big programs will have to be designed at a reasonable cost and then modified and maintained during their lifetime (which is often over 20 years). The size and efficiency of the programming and maintenance teams in charge of their design and follow-up cannot grow in similar proportions. At a not so uncommon (and often optimistic) rate of one bug per thousand lines such huge programs might rapidly become hardly manageable in particular for safety critical systems [562]. Therefore in the next 10 years, the *software reliability problem* is likely to become a major concern and challenge to modern highly computer-dependent societies.

In the past decade a lot of progress has been made both on *thinking/methodological tools* (to enhance the human intellectual ability) to cope with complex software systems and *mechanical tools* (using the computer) to help the programmer to reason about programs.

Mechanical tools for computer aided program verification started by executing or simulating the program in as much as possible environments. However debugging of compiled code or simulation of a model of the source program hardly scale up and often offer a low coverage of dynamic program behavior.

Formal program verification methods attempt to mechanically prove that program execution is correct in all specified environments. This includes *deductive methods*, *model checking*, *program typing* and *static program analysis*.

Since program verification is undecidable, computer aided program verification methods are all partial or incomplete. The undecidability or complexity is always solved by using some form of *approximation*. This means that the mechanical tool will sometimes suffer from practical time and space complexity limitations, rely on finiteness hypotheses or provide only semi-algorithms, require user interaction or be able to consider restricted forms of specifications or programs only. The mechanical program verification tools are all quite similar and essentially differ in their choices regarding the approximations which have to be done in order to cope with undecidability or complexity. The purpose of *abstract interpretation* is to formalize this notion of approximation in a unified framework [217, 233, 217, 239, 220, 221, 224, 228, 229].

2 Abstract Interpretation

Since program verification deals with properties, that is sets (of objects with these properties), abstract interpretation can be formulated in an application independent setting, as a theory for approximating sets and set operations as considered in set (or category) theory, including inductive definitions [245]. A more restricted understanding of abstract interpretation is to view it as a theory of approximation of the behavior of dynamic discrete systems (e.g. the formal semantics of programs or a communication protocol specification). Since such behaviors can be characterized by fixpoints [238, 704] (e.g. corresponding to iteration), an essential part of the theory provides constructive and effective methods for fixpoint approximation and checking by abstraction [239, 243].

2.1 Fixpoint Semantics

The *semantics of a programming language* defines the semantics of any program written in this language. The *semantics of a program* provides a formal mathematical model of all possible behaviors of a computer system executing this program in interaction with any possible environment. In the following we will try to explain informally why the semantics of a program can be defined as the solution of a fixpoint equation. Then, in order to compare semantics, we will show that all the semantics of a program can be organized in a hierarchy by abstraction. By observing computations at different levels of abstraction, one can approximate fixpoints hence organize the semantics of a program in a lattice [231, 222].

2.2 Trace Semantics

Our finer grain of observation of program execution, that is the most precise of the semantics that we will consider, is that of a *trace semantics* [231, 239], a model also frequently used in temporal logic [302, 637, 716]. An execution of a program for a given specific interaction with its environment is a sequence of states, observed at discrete intervals of time, starting from an initial state, then moving from one state to the next state by executing an atomic program step or transition and either ending in a final regular or erroneous state or non terminating, in which case the trace is infinite (see Fig. 1).

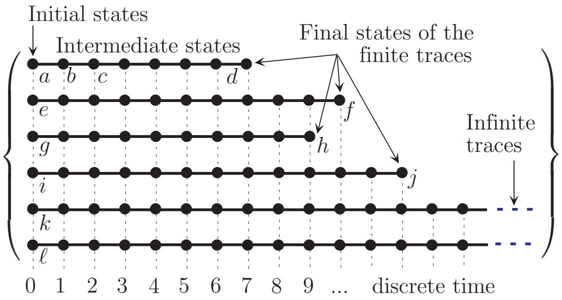


Fig. 1. Examples of Computation Traces

2.3 Least Fixpoint Trace Semantics

Introducing the *computational partial ordering* [231, 245, 223], we define the trace semantics in fixpoint form [231, 245, 223], as the least solution of an equation of the form $X = \mathcal{F}(X)$ where X ranges over sets of finite and infinite traces.

More precisely, let **Behaviors** be the set of execution traces of a program, possibly starting in any state. We denote by $\textbf{Behaviors}^{+}$ the subset of finite traces and by $\textbf{Behaviors}^{\infty}$ the subset of infinite traces.

A finite trace $\overset{a}{\bullet}\!\!-\!\!-\!\ldots\!-\!\!-\!\!\overset{z}{\bullet}$ in $\textbf{Behaviors}^{+}$ is either reduced to a final state (in which case there is no possible transition from state $\overset{a}{\bullet} = \overset{z}{\bullet}$) or the initial state $\overset{a}{\bullet}$ is not final and the trace consists of a first computation step $\overset{a}{\bullet}\!\!-\!\!-\!\!\overset{b}{\bullet}$ after which, from the intermediate state $\overset{b}{\bullet}$, the execution goes on with the shorter finite trace $\overset{b}{\bullet}\!\!-\!\!-\!\ldots\!-\!\!-\!\!\overset{z}{\bullet}$ ending in the final state $\overset{z}{\bullet}$. The finite traces are therefore all well defined by induction on their length.

An infinite trace $\overset{a}{\bullet}\!\!-\!\!-\!\ldots\!-\!\!-\!\ldots$ in $\textbf{Behaviors}^{\infty}$ starts with a first computation step $\overset{a}{\bullet}\!\!-\!\!-\!\!\overset{b}{\bullet}$ after which, from the intermediate state $\overset{b}{\bullet}$, the execution goes on with an infinite trace $\overset{b}{\bullet}\!\!-\!\!-\!\ldots\!-\!\!-\!\ldots$ starting from the intermediate state

$\overset{b}{\bullet}$. These remarks and **Behaviors** $=$ **Behaviors**$^{+}$ $\cup$ **Behaviors**$^{\infty}$ lead to the following fixpoint equation:

$$\begin{aligned}\mathbf{Behaviors} = {} & \{\overset{a}{\bullet} \mid \overset{a}{\bullet} \text{ is a final state}\}\\ & \cup \{\overset{a}{\bullet}\!-\!\!-\!\overset{b}{\bullet}\!-\!\!-\ldots-\!\!-\!\overset{z}{\bullet} \mid \overset{a}{\bullet}\!-\!\!-\!\overset{b}{\bullet} \text{ is an elementary step } \&\\ & \qquad\qquad \overset{b}{\bullet}\!-\!\!-\ldots-\!\!-\!\overset{z}{\bullet} \in \mathbf{Behaviors}^{+}\}\\ & \cup \{\overset{a}{\bullet}\!-\!\!-\!\overset{b}{\bullet}\!-\!\!-\ldots-\!\!-\ldots \mid \overset{a}{\bullet}\!-\!\!-\!\overset{b}{\bullet} \text{ is an elementary step } \&\\ & \qquad\qquad \overset{b}{\bullet}\!-\!\!-\ldots-\!\!-\ldots \in \mathbf{Behaviors}^{\infty}\}\end{aligned}$$

In general, the equation has multiple solutions. For example if there is only one non-final state $\overset{a}{\bullet}$ and only possible elementary step $\overset{a}{\bullet}\!-\!\!-\!\overset{a}{\bullet}$ then the equation is **Behaviors** $= \{\overset{a}{\bullet}\!-\!\!-\!\overset{a}{\bullet}\!-\!\!-\ldots-\!\!-\ldots \mid \overset{a}{\bullet}\!-\!\!-\ldots-\!\!-\ldots \in \mathbf{Behaviors}\}$. One solution is $\{\overset{a}{\bullet}\!-\!\!-\!\overset{a}{\bullet}\!-\!\!-\!\overset{a}{\bullet}\!-\!\!-\!\overset{a}{\bullet}\!-\!\!-\ldots-\!\!-\ldots\}$ but another one is the empty set $\emptyset$. Therefore, we choose the least solution for the *computational partial ordering* [231, 245, 223]:

« More finite traces & less infinite traces » .

2.4 Abstractions & Abstract Domains

A programming language semantics is more or less precise according to the considered observation level of program execution [17, 377, 434]. This intuitive idea can be formalized by *Abstract interpretation* [231] and applied to different languages [64, 63, 338], including for proof methods [219, 525].

The *theory of abstract interpretation* formalizes this notion of approximation and abstraction in a mathematical setting which is independent of particular applications. In particular, abstractions must be provided for all mathematical constructions used in semantic definitions of programming and specification languages [239, 243, 537, 579, 588, 589, 595, 598, 644, 645, 663, 664].

An *abstract domain* is an abstraction of the concrete semantics in the form of *abstract properties* (approximating the concrete properties **Behaviors**) and *abstract operations* (including abstractions of the concrete approximation and computational partial orderings, an approximation of the concrete fixpoint transformer $\mathcal{F}$, etc.). Abstract domains for complex approximations of designed by composing abstract domains for simpler components [239], see Sec. 2.10.

If the approximation is coarse enough, the abstraction of a concrete semantics can lead to an abstract semantics which is less precise, but is *effectively computable* by a computer. By effective computation of the abstract semantics, the computer is able to analyze the behavior of programs and of software before and without executing them [232]. *Abstract interpretation algorithms* provide approximate methods for computing this abstract semantics. The most important algorithms in abstract interpretation are those providing effective methods for the exact or approximate iterative resolution of fixpoint equations [233].

We will first illustrate formal and effective abstractions for sets. Then we will show that such abstractions can be lifted to functions and finally to fixpoints.

The abstraction idea and its formalization are equally applicable in other areas of computer science such as artificial intelligence [361] e.g. for intelligent planning [23, 99], proof checking [358], automated deduction, theorem proving [98, 100, 307, 357, 359, 360, 362, 363, 99, 356], etc.

2.5 Hierarchy of Abstractions

As shown in Fig. 2 (from [231], where **Behaviors**, denoted $\tau^{\vec{\infty}}$ for short, is the lattice infimum), all abstractions of a semantics can be organized in a lattice (which is part of the lattice of abstract interpretations introduced in [239, 237]). The *approximation partial ordering* of this lattice formally corresponds to logical implication, intuitively to the idea that one semantics is more precise than another one.

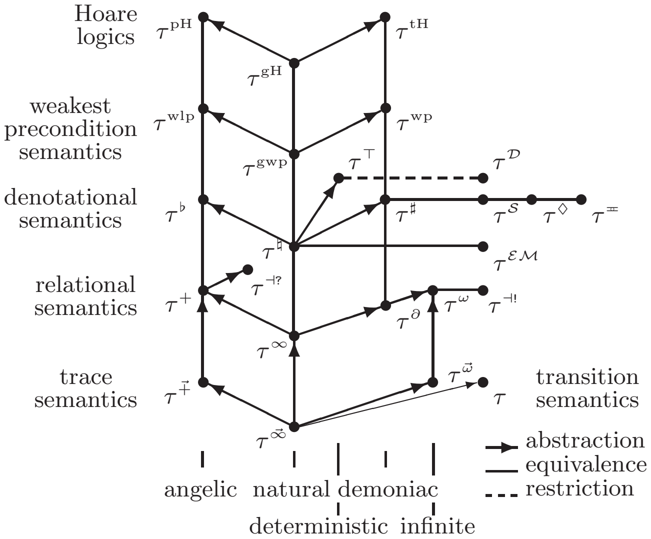


Fig. 2. The Hierarchy of Semantics

Fig. 3 illustrates the derivation of a *relational semantics* [434, 561] (denoted τ^{∞} in Fig. 2) from a trace semantics (denoted $\tau^{\vec{\infty}}$ in Fig. 2). The abstraction α_r from trace to relational semantics consists in replacing the finite traces $\overset{a}{\bullet}$——. . .——$\overset{z}{\bullet}$ by the pair $\langle a, z\rangle$ of the initial and final states. The infinite traces $\overset{a}{\bullet}$——$\overset{b}{\bullet}$——. . .——. . . are replaced by the pair $\langle a, \perp\rangle$ where the symbol $\perp$ denotes non-termination. Therefore the abstraction is:

$$\alpha_r(X) = \{\langle a, z\rangle \mid \overset{a}{\bullet}\text{——}\ldots\text{——}\overset{z}{\bullet} \in X\} \cup \{\langle a, \perp\rangle \mid \overset{a}{\bullet}\text{——}\overset{b}{\bullet}\text{——}\ldots\text{——}\ldots \in X\} .$$

The *denotational semantics* [575, 673, 705] (denoted $\tau^{\natural}$ in Fig. 2) is the isomorphic representation of a relation by its right-image:

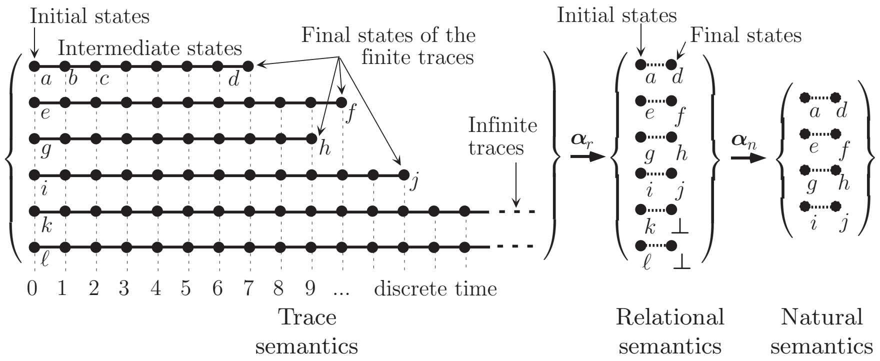


Fig. 3. Abstraction from Trace to Relational and Natural Semantics

$$\alpha_d(R) = \boldsymbol{\lambda}\, a \cdot \{x \mid \langle a, x\rangle \in R\}.$$

The abstraction from relational to *big-step operational or natural semantics* (denoted τ^+ in Fig. 2) [489, 635] simply consists in forgetting everything about non-termination, so $\alpha_n(R) = \{\langle a, x\rangle \in R \mid x \neq \bot\}$, as illustrated in Fig. 3.

A non comparable abstraction consists in collecting the set of initial and final states as well as all transitions $\langle x, y\rangle$ appearing along some finite or infinite trace $\overset{a}{\bullet}$——$\ldots\overset{x}{\bullet}$——$\overset{y}{\bullet}\ldots$ of the trace semantics. One gets the *small-step operational or transition semantics* [495, 635] (denoted τ in Fig. 2 and also called Kripke structure in modal logic [508]) as illustrated in Fig. 4.

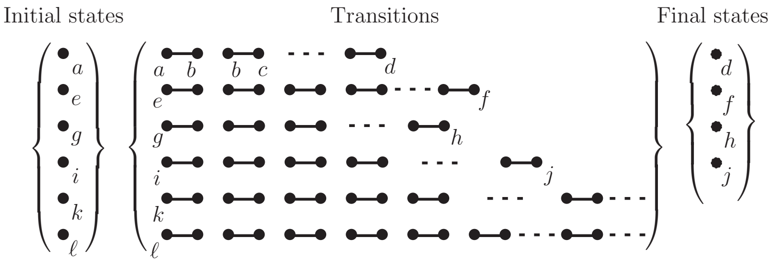


Fig. 4. Transition Semantics

A further abstraction consists in collecting all states appearing along some finite or infinite trace as illustrated in Fig. 5. This is the *partial correctness semantics* [219, 331, 433, 592] or the *static/collecting semantics* [233] for proving invariance properties of programs.

All abstractions considered in this paper are "from above" so that the abstract semantics describes a superset or logical consequence of the concrete

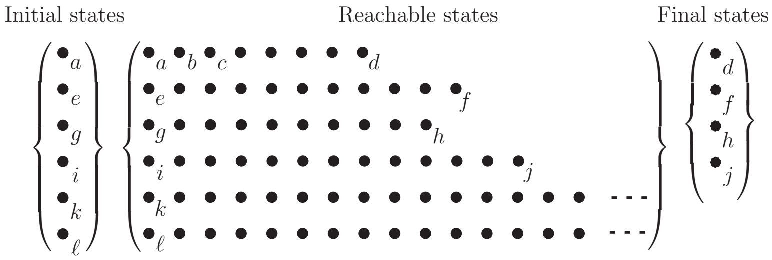
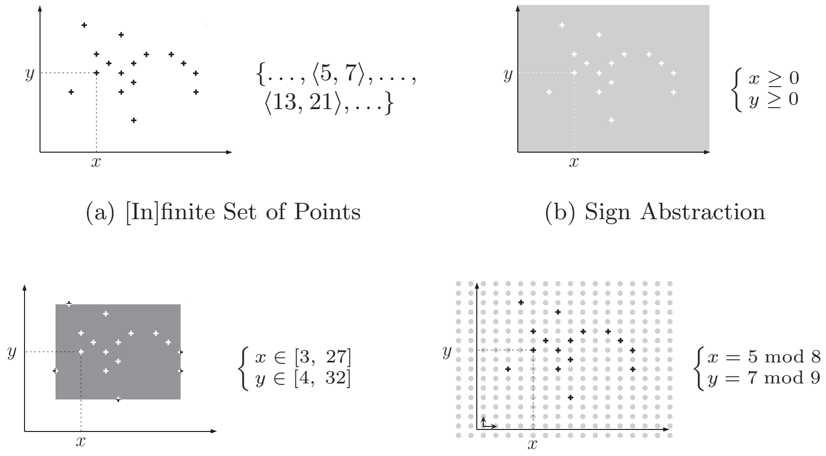


Fig. 5. Static / Collecting / Partial Correctness Semantics



(a) [In]finite Set of Points

(b) Sign Abstraction

(c) Interval Abstraction

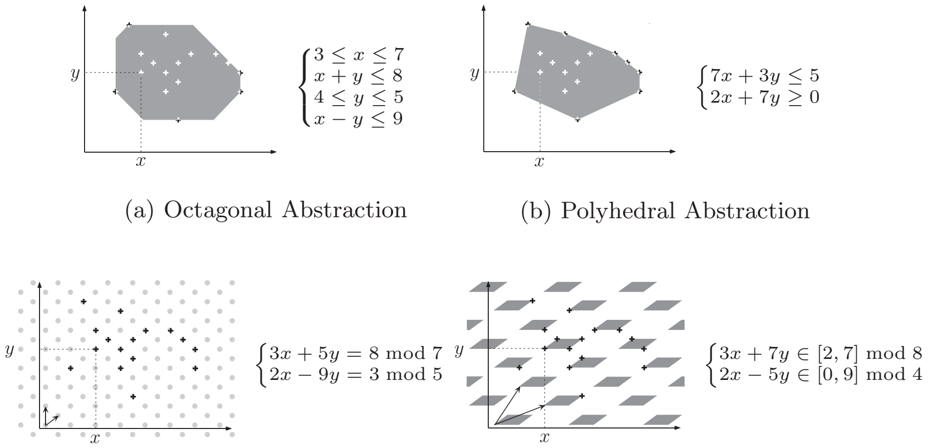
(d) Simple Congruence Abstraction

Fig. 6. Non-relational Abstractions

semantics. Abstractions "from below" are dual and consider a subset of the concrete semantics. An example of approximation "from below" is provided by debugging techniques which consider a subset of the possible program executions or by existential checking where one wants to prove the existence of an execution trace prefix fulfilling some given specification. In order to avoid repeating two times dual concepts and as we do usually, we only consider approximations "from above", knowing that approximations "from below" can be easily derived by applying the duality principle (as found e.g. in lattice theory [56]).

2.6 Effective Abstractions

Numerical Abstractions Assume that a program has two integer variables `X` and `Y`. The trace semantics of the program (Fig. 1) can be abstracted in the static/collecting semantics (Fig. 5). A further abstraction consists in forgetting



(a) Octagonal Abstraction

(b) Polyhedral Abstraction

(c) Relational Congruence Abstraction

(d) Trapezoidal Congruence Abstraction

Fig. 7. Relational Abstractions

in a state all but the values x and y of variables `X` and `Y`. In this way the trace semantics is abstracted to a set of points (pairs of values), as illustrated in the plane by Fig. 6(a).

We now illustrate informally a number of effective abstractions of an [in]finite set of points.

Non-relational Abstractions The non-relational, attribute independent or cartesian abstractions [239, example 6.2.0.2] [480] consists in ignoring the possible relationships between the values of the `X` and `Y` variables. So a set of pairs is approximated through projection by a pair of sets. Each such set may still be infinite and in general not exactly computer representable. Further abstractions are therefore needed.

The *sign abstraction* [239] illustrated in Fig. 6(b) consists in replacing integers by their sign thus ignoring their absolute value. The *interval abstraction* [232, 233, 217] illustrated in Fig. 6(c) is more precise since it approximates a set of integers by it minimal and maximal values (including $-\infty$ and $+\infty$ as well as the empty set if necessary).

The *congruence abstraction* [372, 373, 375] (generalizing the parity abstraction [239]) is not comparable, as illustrated in Fig. 6(d).

Relational Abstractions Relational abstractions are more precise than non relational ones [480] in that some of the relationships between values of the program states are preserved by the abstraction.

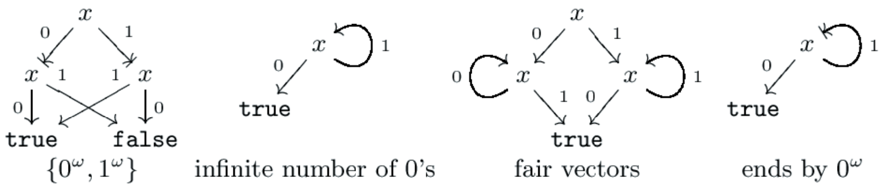


Fig. 8. Binary Decision Graphs

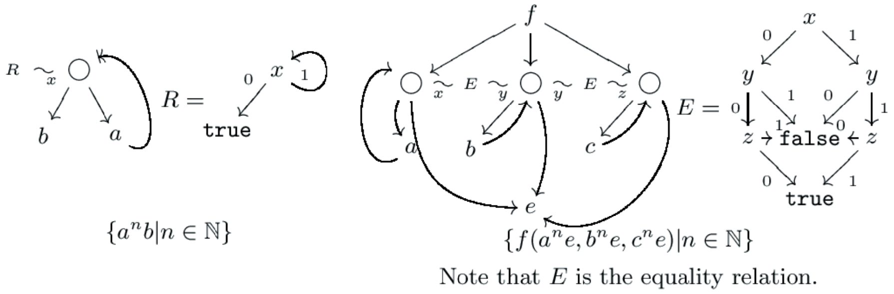


Fig. 9. Tree Schemata

For example the *polyhedral abstraction* [255, 380] illustrated in Fig. 7(b) approximates a set of integers by its convex hull. Only non-linear relationships between the values of the program variables are forgotten.

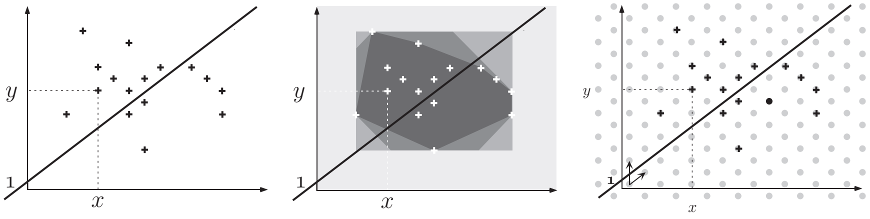
The use of an *octagonal abstraction* [31] illustrated in Fig. 7(a) is less precise since only some shapes of polyhedra are retained or equivalently only linear relations between any two variables are considered with coefficients +1 or -1 (of the form $\pm x \pm y \leq c$ where c is an integer constant).

A non comparable relational abstraction is the *linear congruence abstraction* [374] illustrated in Fig. 7(c).

A combination of non-relational dense approximations (like intervals) and relational sparse approximations (like congruences) is the *trapezoidal linear congruence abstraction* [543, 542] as illustrated in Fig. 7(d).

Symbolic Abstractions Most structures manipulated by programs are *symbolic structures* such as control structures (call graphs), data structures (search trees, pointers [289, 294, 696, 722]), communication structures (distributed & mobile programs [319, 399, 721]), etc. It is very difficult to find compact and expressive abstractions of such sets of objects (sets of languages, sets of automata, sets of trees or graphs, etc.). For example B chi automata or automata on trees are very expressive but algorithmically expensive [706].

A compromise between semantic expressivity and algorithmic efficiency was recently introduced by [548, 546, 547, 549] using *Binary Decision Graphs* and



(a) yes (b) unkown (c) yes

Fig. 10. Is `1/(X+1-Y)` well-defined?

Tree Schemata to abstract infinite sets of infinite trees as illustrated in Fig. 8 & 9.

2.7 Information Loss

Any abstraction introduces some loss of information. For example the abstraction of the trace semantics into relational or denotational semantics loses all information on the computation cost since all intermediate steps in the execution are removed.

All answers given by the abstract semantics are always correct with respect to the concrete semantics. For example, if termination is proved using the relational semantics then there is no execution abstracted to $\langle a, \perp\rangle$, so there is no infinite trace $\overset{a}{\bullet}\!\!-\!\!-\!\!\overset{b}{\bullet}\!\!-\!\!-\ldots-\!\!-\ldots$ in the trace semantics, whence non termination is impossible when starting execution in initial state a.

However, because of the information loss, not all questions can be definitely answered with the abstract semantics. For example, the natural semantics cannot answer questions about termination as can be done with the relational or denotational semantics. These semantics cannot answer questions about concrete computation costs.

The more concrete is the semantics, the more questions it can answer. The more abstract semantics are simpler. Non comparable abstract semantics (such as intervals and congruences) answer non comparable sets of questions.

To illustrate the loss of information, let us consider the problem of deciding whether the operation `1/(X+1-Y)` appearing in a program is always well defined at run-time. The answer can certainly be given by the concrete semantics since it has no point on the line $x + 1 - y = 0$, as shown in Fig. 10(a).

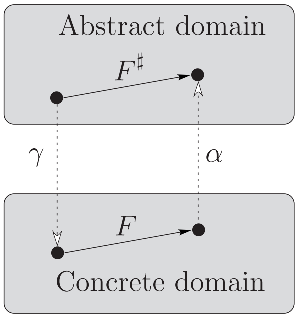
In practice the concrete abstraction is not computable so it is hardly usable in a useful effective tool. The dense abstractions that we have considered are too approximate as is illustrated in Fig. 10(b).

However the answer is positive when using the relational congruence abstraction, as shown in Fig. 10(c).

2.8 Function Abstraction

We now show how the abstraction of complex mathematical objects used in the semantics of programming or specification languages can be defined by composing abstractions of simpler mathematical structures.

For example knowing abstractions of the parameter and result of a monotonic function on sets, a function F can be abstracted into an abstract function $F^\sharp$ as illustrated in Fig. 11 [239]. Mathematically, $F^\sharp$ takes its parameter x in the abstract domain. Let $\gamma(x)$ be the corresponding concrete set (γ is the adjoined, intuitively the inverse of the abstraction function α). The function F can be applied to get the concrete result $\circ\ F \circ \gamma(x)$. The abstraction function α can then be applied to approximate the result $F^\sharp(x) = \alpha \circ F \circ \gamma(x)$.
In general, neither F, α nor γ are computable even though the abstraction α may be effective. So we have got a formal specification of the abstract function $F^\sharp$ and an algorithm has to be found for an effective implementation.



$$F^\sharp = \alpha \circ F \circ \gamma$$

Fig. 11. Function Abstraction

2.9 Fixpoint Abstraction

A fixpoint of a function F can often be obtained as the limit of the iterations of F from a given initial value $\perp$ [238, 704]. In this case the abstraction of the fixpoint can often be obtained as the abstract limit of the iteration of the abstraction $F^\sharp$ of F starting from the abstraction $\alpha(\perp)$ of the initial value $\perp$. The basic result is that the concretization of the abstract fixpoint is related to the concrete fixpoint by the approximation relation expressing the soundness of the abstraction [239]. This is illustrated in Fig. 12.

Often states have some finite component (e.g. a program counter) which can be used to partition into fixpoint system of equations by projection along that component. Then *chaotic* [234] and *asynchronous iteration strategies* [217, 216] can be used to solve the equations iteratively. Various efficient iteration strategies have been studied [472, 129, 392, 481, 482, 473, 484, 590, 651, 670, 515, 725], including ones taking particular properties of abstractions into account [135, 314, 320, 452, 505, 601, 606, 605, 675] and others to speed up the convergence of the iterates [244, 77, 232, 233, 393, 394].

2.10 Composing Abstractions

Abstractions hence abstract interpreters for static program analysis can be designed compositionally by stepwise abstraction, combination or refinement [349, 227, 185, 183, 239, 323, 344, 339].

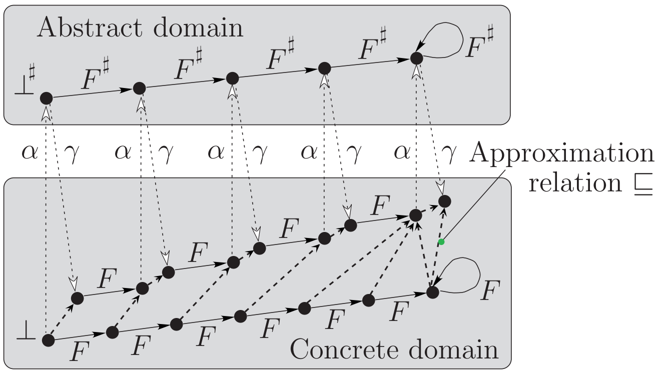


Fig. 12. Fixpoint Abstraction lfp $F \sqsubseteq \gamma(\text{lfp}\, F^\sharp)$

An example of stepwise abstraction is the functional abstraction of Sec. 2.8. The abstraction of a function is parameterized by abstractions for the function parameters and the function result which can be chosen later in the modular design of the abstract interpreter.

An example of abstraction combination is the *reduced product* of two abstractions [239, 186, 348] which is the most abstract abstraction more precise than these two abstractions or the *reduce cardinal power* [239, 391] generalizing case analysis. Such combination of abstract domains can be implemented as parameterized modules in static analyzer generators (e.g. [517, 213]) so as to partially automate the design of expressive analyses from simpler ones.

An example of refinement is the *disjunctive completion* [239, 324, 326, 343–346, 597] which completes an abstract domain by adding concrete disjunctions missing in the abstract domain. Another example of abstract domain refinement is the *complementation* [209, 208, 209, 325] adding concrete negations missing in the abstract domain.

2.11 Sound and Complete Abstractions

Abstract interpretation theory has mainly been concerned with the *soundness* of the abstract semantics/interpreter, relative to which questions can be answered correctly despite the loss of information [233]. Soundness is essential in practice and leads to a formal design method [239].

However *completeness*, relative to the formalization of the loss of information in a controlled way so as to answer a given set of questions, has also been intensively studied [239, 349, 342, 348, 587, 347, 349], including in the context of model checking [230].

In practice complete abstractions, including a most abstract one, always exist to check that a given program semantics satisfies a given specification. Moreover any given abstraction can be refined to a complete one. Nevertheless this approach has severe practical limitations since, in general, the design of such complete abstractions or the refinement of a given one is logically equivalent to the design of an inductive argument for the formal proof that the given program satisfies the given specification, while the soundness proof of this abstraction logically amounts to checking the inductive verification conditions or proof obligations of this formal proof [230]. Such proofs can hardly be fully automated hence human interaction is unavoidable. Moreover the whole process has to be repeated each time the program or specification is modified.

Instead of considering such strong specifications for a given specific program, the objective of static program analysis is to consider (often predefined) specifications and all possible programs. The practical problem in static program analysis is therefore to design useful abstractions which are computable for all programs and expressive enough to yield interesting information for most programs.

3 Static Program Analysis

Static program analysis is the automatic static determination of dynamic runtime properties of programs.

3.1 Foundational Ideas of Static Program Analysis

Given a program and a specification, a program analyzer will check if the program semantics satisfies the specification (Fig. 13). In case of failure, the analyzer will provide hints to understand the origin of errors (e.g. by a backward analysis providing necessary conditions to be satisfied by counter-examples).

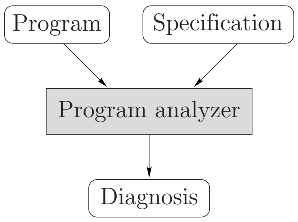


Fig. 13. Program Analysis

The principle of the analysis is to compute an approximate semantics of the program in order to check a given specification. Abstract interpretation is used to derive, from a standard semantics, the approximate and computable abstract semantics. The derivation can often be done by composing standard abstractions to fit a particular kind of information which has to be discovered about program execution. This derivation is itself not (fully) mechanizable but *static analyzer generators* such as PAG [540, 10], GENA [312]and others [32, 172, 226, 227, 301, 300, 304, 305, 715, 463, 516, 517, 620, 523, 610, 124] can provide generic abstractions to be composed with problem specific ones.

In practice, the program analyzer contains a *generator* reading the program text and producing equations or constraints whose solution is a computer representation of the program abstract semantics. A *solver* is then used to solve these abstract equations/constraints. A popular resolution method is to use iteration. Of the numerical abstractions considered in Sec. 2.6, only the sign and simple congruence abstractions ensure the finite convergence of the iterates. If the limit of the iterates is inexistent (which may be the case e.g. for the polyhedral abstraction) or it is reached after infinitely many iteration steps (e.g. interval and octagonal abstractions), the convergence may have to be ensured and/or accelerated using a *widening* to over estimate the solution in finitely many steps followed by a *narrowing* to improve it [217, 233, 244].

In *abstract compilation*, the generator and solver are directly compiled into a program which directly yields the approximate solution [70, 532].

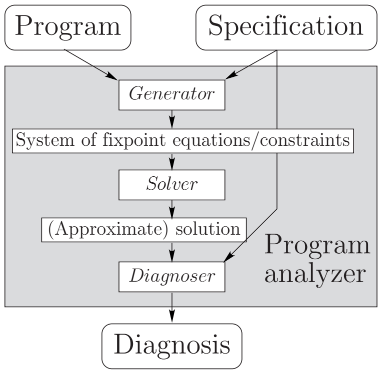


Fig. 14. Principle of Program Analysis

This solution is an approximation of the abstract semantics which is then used by a *diagnoser* to check the specification. Because of the loss of information, the diagnosis is always of the form "yes", "no", "unknown" or "irrelevant" (e.g. a safety specification for unreachable code). The general structure of program analyzers is illustrated in Fig. 14. Besides diagnosis, static program analysis is also used for other applications in which case the diagnoser is replaced by an *optimiser* (for compile-time optimization), a *program transformer* (for partial evaluation [477]), etc.

3.2 Shortcomings of Static Program Analysis

Static program analysis can be used for large programs (e.g. 220,000 lines of C) without user interaction. The abstractions are chosen to be of wide scope without specialization to a particular program. Abstract algebras can be designed and implemented into libraries which are reusable for different programming languages. The objective is to discover invariants that are likely to appear in many programs so that the abstraction must be widely reusable for the program analyzer to be of economic interest.

The drawback of this general scope is that the considered abstract specifications and properties are often simple, mainly concerning elementary safety properties such as absence of run-time errors. For example non-linear abstractions of sets of points are very difficult and very few mathematical results are of practical interest and directly applicable to program analysis [43]. Checking termination and similar liveness properties is trivial with finite state systems, at least from a theoretical if not algorithmic point of view (e.g. finding loops in finite graphs). The same problem is much more difficult for infinite state systems because of fairness [548] or of potentially infinite data structures (as considered e.g. in partial evaluation [478]) which do not amount to finite cycles so that termination or inevitability proofs require the discovery of *variant functions on well-founded sets* which is very difficult in full generality [16, 28, 82, 81, 247, 355, 491, 534, 578, 650, 691, 626, 726].

Even when considering restricted simple abstract properties, the semantics of real-life programming languages is very complex (recursion, concurrency, modularity, etc.) whence so is the corresponding abstract interpreter. The abstraction of this semantics, hence the design of the analyzer is mostly manual (and beyond the ability of casual programmers or theorem provers) whence costly. The considered abstractions must have a large scope of application and must be easily reusable to be of economic interest.

From a user point of view, the results of the analysis have to be presented in a simple way (for example by pointing at errors only or by providing abstract counter-examples, or less frequently concrete ones). Experience shows that the cases of uncertainty represent 5 to 10 % of the possible cases. They must be handled with other empirical or formal methods (including more refined abstract interpretations).

3.3 Applications of Static Program Analysis

Among the numerous applications of static program analysis, let us cite *data flow analysis* [666, 252, 239, 536, 654, 665, 658]; *program optimization* [127, 126] *and transformation* [73, 71, 552, 596] (including partial evaluation and program specialization [477, 322, 334, 413, 475, 476, 604, 639, 521] and data dependence analysis for the parallelisation of sequential languages [257, 388, 486, 490, 559, 649, 699, 711, 712]); *set-based analysis* [248, 6, 308, 418, 417, 419, 420]; *type inference* [225, 235] (including undecidable systems [570, 572] and soft typing [114, 329, 734]); verification of *reactive* [384, 426, 321, 381, 465], *real-time* [387] and *(linear) hybrid systems* [272, 386, 425, 427, 614] including state space reduction [80, 249]; *cryptographic protocol* analysis [567]; *abstract model-checking* of infinite systems [252, 250, 284]; *abstract debugging*, testing and verification [76, 78, 200, 198, 199, 197, 196, 218, 251, 330, 611]; *cache and pipeline behavior prediction* [315, 9, 316, 317, 667]; *probabilistic analysis* [568, 113]; *communication topology analysis* for mobile/distributed code [319, 399, 721, 191, 318, 158, 648, 720, 721]; *automatic differentiation* of numerical programs [698]; *abstract simulation* of temporal specifications [112]; Semantic tattooing/*watermarking* of software [254]; etc.

Static program analysis has been intensively studied for grammars and polynomial systems [222, 566], term graph rewriting [366, 367, 136, 137], sequent calculi [15], typesetting languages [437, 436], procedural languages [74, 236, 75] (for alias analysis [722, 96, 277, 286–288, 389, 430, 438, 459, 550, 642, 719], pointer analysis [289, 294, 117, 337, 669, 656, 657, 722], parameter boxing/unboxing [368], copy elimination [668], dependence analysis [541, 640, 712], exception analysis [662], constant propagation [500], (linear) equality or inequality relationships analysis [255, 493, 299, 380] etc.), parallel procedural languages [258, 365, 133, 134, 138, 139, 192, 261, 694, 504], functional languages (for binding time analysis [718, 20, 95, 97, 270, 335, 421, 423, 453, 513, 488, 603, 621, 655, 688], strictness analysis [106, 246, 584, 30, 40, 105, 102, 107, 103, 202, 271, 286, 306, 422, 440, 445, 450, 451, 466, 468, 471, 474, 483, 509, 510, 544, 545, 573, 583, 594, 600, 599, 608, 274, 676, 677, 676, 678, 687, 689, 730, 735], inverse image analysis [298, 297, 446, 443, 442, 444], projection analysis [104 , 447, 448 , 512 , 514 , 629 , 679 , 731], comportment analysis [247 , 591 , 630, 631], dependency analysis [62], path/trace analysis [193 , 61 , 60], closure analysis [622 , 21, 22 , 256 , 565], control flow analysis [683 , 278 , 311 , 460 , 576 , 612, 613 , 624 , 624 , 652 , 692 , 702 , 714], value flow analysis [65 , 441 , 577 , 684 , 736], compile-time garbage collection [449], stackability and escape analysis [58 , 34 , 397], data structures and abstract data type analysis [526 , 45 , 467 , 469, 470 , 607 , 680], heap shape analysis [479 , 696 , 354 , 336], exception analysis [738 , 737 , 739], polymorphic function analysis [36 , 5 , 35 , 201], kind/sort analysis [378 , 128], typing [225 , 203 , 395 , 407, 408 , 506, 507 , 560 , 563 , 569 , 431 , 571, 572 , 570 , 585, 586], effect systems [487 , 701 , 602 , 454, 455 , 707], termination analysis [626], time complexity analysis [646 , 295 , 661 , 729], parallelization [710], etc.), parallel functional languages [262 , 259, 260 , 439 , 531], data parallel languages [125], logic languages including Prolog [242 , 275 , 39 , 89 , 86 , 179 , 168 , 162 , 167] (for mode [555 , 279, 280 , 530 , 553, 554 , 556 , 700 , 456] and type analysis [435 , 38 , 41 , 170 , 180 , 184 , 462 , 328] and their combination [88], finiteness analysis [55], relational argument size analysis [671 , 686], dependency analysis [582 , 19 , 18 , 118 , 119 , 333], detecting determinate/functional computations [350], mutually exclusive rules detection [638], occur check reduction [690], WAM code optimization [33], copy avoidance [332], groundness analysis [212 , 165, 166 , 163 , 169 , 171 , 210 , 210 , 416 , 122 , 538 , 647 , 674], sharing analysis [207 , 25, 26 , 181, 182 , 187 , 181 , 187 , 207 , 428 , 458 , 742 , 740], freeness analysis [164 , 205 , 296 , 429] and their combinations [188 , 84, 85 , 87 , 376 , 206 , 211 , 313 , 273 , 524 , 581 , 697], termination analysis [491 , 28, 29 , 82, 83 , 81 , 189 , 281, 282 , 285 , 672 , 355 , 522 , 534 , 578 , 636 , 650 , 691 , 726, 727 , 724 , 723], time complexity and cost analysis [276 , 2], parallelisation [91, 92], etc.) including its search rule and the cut [327 , 123 , 564] and database programming languages [12–14 , 46 , 615 , 708], concurrent logic languages [108 , 132 , 173 , 175, 176 , 174 , 685 , 501–503 , 713], functional logic languages [401 , 48 , 398 , 403 , 743 , 404], constraint logic languages [57 , 24 , 72 , 115 , 190 , 296 , 340, 341 , 400 , 390 , 402 , 464 , 485 , 496 , 121 , 518 , 539 , 558], concurrent constraint logic languages [741 , 177, 178 , 309], specification languages [353 , 351, 352], synchronous languages [384 , 49 , 383 , 469] (such as LUSTRE [116 , 385 , 382]), concurrent/parallel languages [241 , 37 , 261 , 1], communicating and distributed languages [240 , 557 , 131 , 310 , 492] and more recently object-oriented languages [59 , 8 , 283 , 379 , 715 , 520 , 609 , 623 , 625 , 160 , 728].

Abstract interpretation based static program analyses have been used for the static analysis of the embedded ADA software of the Ariane 5 launcher[1] and the ARD[2] [511 , 290]. The static program analyser aims at the automatic detection of the *definiteness*, *potentiality*, *impossibility* or *inaccessibility* of run-time errors such as scalar and floating-point overflows, array index errors, divisions by zero and related arithmetic exceptions, uninitialized variables, data races on shared data structures, etc. The analyzer was able to automatically discover the Ariane

[1] Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).
[2] Atmospheric Reentry Demonstrator.

501 flight error. The static analysis of embedded safety critical software (such as avionic software [643]) is very promising [253].

3.4 Industrialization of Static Analysis by Abstract Interpretation

The impressive results obtained by the static analysis of real-life embedded critical software [511 , 643] is quite promising for the industrialization of abstract interpretation.

This is the explicit objective of AbsInt Angewandte Informatik GmbH (`www.absint.com`) created in Germany by R. Wilhelm and C. Ferdinand in 1998 commercializing the program analyzer generator PAG and an application to determine the worst-case execution time for modern computer architectures with memory caches, pipelines, etc [315].

Polyspace Technologies (`www.polyspace.com`) was created in France by A. Deutsch and D. Pilaud in 1999 to develop and commercialize ADA and C program analyzers.

Other companies like Connected Components Corporation (`www.concmp.com`) created in the U.S.A. by W.L. Harrison in 1993 use abstract interpretation internally e.g. for compiler design [410].

4 Abstract Formal Methods

No automatic formal method can ultimately find all errors in a software system nor can their combinations. We will briefly review the automatic formal methods for computer-aided program verification, discussing their principles, advantages and shortcomings. Since static program analysis has already been discussed, we now consider typing, model-checking, deductive methods and their combination.

4.1 Typing

Polymorphic typing and type inference [560 , 265] was a definite step in the design of programming languages and compilers [408, 409]. The question for the next decade seems to be to scale to more expressive properties.

Foundational Ideas of Typing Typing is based on decidable program analyses. This approach is always possible by restricting both on specifications (allowed types) and on programs, as shown when considering types as abstract interpretations [225]. In theory, type systems have a clean presentation of the type analysis (inference algorithm [560]) through an equivalent logical formal system (type verification [265]). Monomorphic typing [431] was extended to polymorphism [560], complex data structures, references [405, 406], exceptions and separate modules [407] in a way that scales up for very large programs. It is nicely integrated in the compiler and the certification can go down to the generated code (proof-carrying code [593], certified compiler [703 , 574]).

Shortcomings of Typing Type systems (e.g. with subtle subtyping) can be very complex to understand for the casual user. One difficulty is that typing is compositional but not fully abstract (e.g. the same polymorphic code can type differently in different utilization contexts). The interaction with the user is often crude (no hint is given to understand why wrong programs do not type well). It is hardly possible for the user to provide hints to help the typing process. The logical specification of the type system is often inexistent in the reference manual, not equivalent to the type inference algorithm or so inextricable that it is useless both to the programmer and the compiler designer. The programs considered in type theory are both complex (higher-order modules) and too restricted (mainly functional languages). The most severe restrictions are on the considered properties (arithmetic, out of range array indexing, null pointer dereferencing, … errors are checked at run-time, all liveness properties are ignored). These restrictions and the difficulty to generalize to more expressive properties mainly follow from the encoding of types as terms/formulæ and from the one iterate fixpoint approximation.

4.2 Deductive Methods

Foundational Ideas of Deductive Methods Deductive methods use a (manually designed abstraction of) the program semantics to obtain minimal verification conditions to prove program correctness. These verification conditions can be derived from the program trace semantics by abstract interpretation [231]. Then a theorem prover [619, 618] or a proof assistant [632, 204] is used to check the verification conditions.

Shortcomings of Deductive Methods Deductive methods use the schema of Fig. 14 but for the fact that the solver is replaced by a verifier or checker thus avoiding fixpoint computations. So the constraints or equations corresponding to the verification conditions are not solved. This means that an inductive argument (e.g. invariant, variant function) has to be provided, generally by the user. Since the implication involved in the verification condition is itself undecidable, the proof verification can only be partially automatized, even though the solution to the equations/constraints is provided. Therefore interaction of the programmer with the prover is ultimately needed. This (wo)man/prover interaction is hard if not despairing, in particular because the size of the proof is often exponential in the program size. Therefore debugging an unsuccessful proof (because of a program error or a prover weakness) can be as complex as (if not much more complex than) debugging the program itself.

An alternative [519] consists in restricting the form of predicates considered by the prover, (which is an abstract interpretation [239, Sec. 5]). This can go up to unsound verification condition simplifications, essentially to make the verifier simpler (e.g. modular arithmetic).

Because theorem provers are driven by unformalized heuristics, and these heuristics and their interactions are changed over time for improving proof strategies, theorem provers are often unstable over time (e.g. proof strategies get changed so that old proofs no longer work). Another weakness which makes interaction with other formal methods somewhat difficult is the uniform encoding of properties as syntactical terms/formulæ (so that e.g. BBDs are hardly efficiently encodable). It follows that the theorem prover has ultimately to be extended with program analyzers, model checkers, typing, among others [682, 616, 617, 681], often without supporting theory, in particular for mechanizing and combining abstractions.

4.3 Model Checking

Model checking [140, 641, 50, 142, 146, 580] has been very successful for the verification of hardware [54, 130, 263, 634], communication protocols [148, 67, 66, 150, 149, 154], cryptographic protocols [51], and real-time [111, 79, 110, 415, 414, 424, 161, 732] or probabilistic [717, 412, 27] processes. As far as software systems are concerned, the question for the next decade is whether model checking can be extended to the verification of very large real-life programs.

Foundational Ideas of Model Checking First a model of the program (i.e. manually designed abstraction of the program semantics) must be designed (in the form of a transition system similar to a small step operational semantics). Then a specification of the program must be provided by the user in a very expressive temporal logic [637, 42]. A model checker can then check the specification by exhaustive search/symbolic exploration of the state space.

The spectacular success of model checking followed from the clever design of data structures (e.g. BDDs [7, 90, 101, 215, 214, 551, 709] or QDDs [68]) and algorithms (e.g. minimal state graph generation [69], fixpoint computation [529, 47, 51, 157, 151, 152, 264, 303, 693, 528] or SAT [52, 4, 53]) for representing very large sets of booleans and their transformations.

The approximation is that the model must be finite-state or some form of abstract interpretation must be used [143, 370, 3, 44, 120, 144, 147, 156, 194, 195, 267, 268, 266, 628, 269, 369, 293, 457, 497, 494, 498, 499, 527, 533, 627] to reduce the verification problem to finite state, including symmetries [141], etc. Also clever semantics of concurrent systems have been considered, e.g. to avoid the combinatorial explosion of interleaving [148, 145, 633, 695].

Another trend in infinite-state model checking is to consider safety properties only and polyhedral abstractions, with variants (e.g. Presburger arithmetic [93, 94, 733]). This is a direct application of polyhedral static program analysis [255, 380], including the use of widenings. This allows e.g. for the analysis of reactive [321, 364], real-time [381, 109, 153, 155, 291, 292, 411, 424, 432] and hybrid systems [384, 11, 94, 386, 387, 614].

Shortcomings of Model Checking Although model checking gained a factor of 100 in 10 years, it is very difficult to scale up because of the state explosion problem. So, the necessary restriction to available computer resources often reduces the model checker from formal verification to debugging on part of the state space. Since the model must ultimately be finite (to allow for exhaustive search/symbolic exploration), abstraction is mandatory, which is a very difficult task to do manually and/or is left informal. Moreover, some forms of abstractions (such as interval [233, 217] or polyhedral [255, 380] abstractions) do not abstract concrete transition systems into abstract transition systems so that the model checker may not be reusable in the abstract [250, 230]. One can use abstraction for model checking which are complete in that there always exists a program specific abstraction into a finite model to prove a given specification correct (see [230] for safety properties) but none will be complete for all programs, even for simple properties as considered in static program analysis [244]. It follows that complete abstractions are difficult and not reusable, hence not cost effective.

5 Combining Program Verification Methods

Since no single formal method can ultimately solve the verification problem, a current trend is to combine formal methods.

For example, one can rely on a user designed abstraction and derive a finite abstract model of the program semantics by abstract interpretation, prove the correctness of the abstraction by deductive methods and later verify the abstract model by model-checking [660, 371, 653, 159, 535, 659].

A fundamental limitation [230] is that the abstraction discovery and the derivation of the abstract semantics are respectively logically equivalent hence practically as difficult as invariant discovery and invariant verification in a formal proof. So we have the feeling that the combination of tools might simplify formal proofs but still will ultimately not solve the program verification problem.

6 Combining Empirical and Formal Methods

Formal methods have made a lot of progress in the last decade. Nevertheless there are few automatic light weight tools to apply them in practice. Integration of such tools is difficult and cannot ultimately solve all verification problems.

It follows that the only mechanical tool for verifying programs is still *debugging*, despite its well-known defaults, incompleteness and cost. There again progress was slow, in particular because theory never took debugging seriously. The main advantage of debugging is that a debugger is a light weight tool which is very easily understood by all programmers. Because of its well-known incredible cost for weak results, debugging may not scale up in the next decade for very large or safety critical software.

An alternative which still remains to be investigated is the combination of informal methods like debugging with verification tools. Let us consider for example *abstract testing* [251, 76, 78, 200, 198, 199, 197, 196, 218, 330, 611].

The classical debugging methodology consists in running the program on test data, checking if the execution satisfies informal specifications. This process is repeated by providing more tests until reaching a satisfactory coverage.

By an easily understandable analogy, the abstract testing methodology [251] consists in computing the abstract semantics for a finitary or infinitary abstraction chosen by the programmer among a predefined palette (not user defined, which would be too difficult). The abstract semantics is then checked against user-provided abstract assertions or the abstraction of a formal specification. This process is repeated with more refined abstractions until enough assertions are proved or no predefined abstraction can do.

Observe that one can prove the absence of (some categories of) bugs, not only their presence. Moreover, abstract evaluation can range from an analogy to program execution to the application of proof methods (using e.g. forward as well as backward reasonings providing abstract counter-examples) without attempting to make a one-shot complete formal proof of the specification.

7 Conclusions on the Past Decade

Full program verification by formal methods (e.g. model checking/deductive methods), which requires user interaction (for discovering an abstraction or inductive argument) is very costly in human resources, hence is not likely to scale up for very large software. Abstraction is mandatory for program verification, but difficult, hardly automatizable and beyond the common capabilities of most programmers.

Partial program verification by static analysis (with typing being considered as a particular and successful case) is *cost-effective*[3] because no user intervention is mandatory for performing the analysis and universal abstractions are reusable, hence commercializable.

For large and complex programs, complete verification by formal methods is not likely to be viable at low cost. Program debugging is still and will probably remain for some time the prominent industrial program "verification" method.

In this context, abstract interpretation based static program analysis can be extended to *abstract program testing*. Abstract interpretation based methods offer powerful techniques which, in the presence of approximation, can be viable alternatives or complements both to the exhaustive search of model-checking and to the partial exploration methods of classical debugging.

8 Grand Challenge for the Next Decade

We believe that in the next decade the software industry will certainly have to face its responsibility imposed by a computer-dependent society, in particular for safety critical systems. Consequently, *Software reliability*[4] will be a grand challenge for computer science and practice.

[3] e.g. less than 0.25$ per program line costing 50 to 80$.

[4] other suggestions were "trustworthiness" (C. Jones) and "robustness" (R. Leino).

The grand challenge for formal methods, in particular abstract interpretation based formal tools, is both the large scale industrialization and the intensification of the fundamental research effort.

General-purpose, expressive and cost-effective abstractions have to be developed e.g. to handle floating point numbers, data dependences (e.g. for parallelization), liveness properties with fairness (to extend finite-state model-checking to software), timing properties for embedded software, probabilistic properties, etc. Present-day tools will have to be enhanced to handle higher-order compositional modular analyses and to cope with new programming paradigms involving complex data and control concepts (such as objects, concurrent threads, distributed/mobile programming, etc.), to automatically combine and locally refine abstractions in particular to cope with "unknow" answers, to interact nicely with users and other formal or informal methods.

The most challenging objective might be to integrate formal analysis by abstract interpretation in the full software development process, from the initial specifications to the ultimate program development.

Acknowledgements I thank Radhia Cousot and Reinhard Wilhelm for their comments on a preliminary version of this paper. This work was supported by the DAEDALUS [253] and TUAMOTU [254] projects.

References

1. N. Kobayashi A. Igarashi. Type-based analysis of communication for concurrent programming languages. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 187–201. Springer-Verlag, 1997.
2. F. Benoy A. King, . Shen. Lower-bound time-complexity analysis of logic programs. In J. Małuszyński, editor, *Proc. Int. Symp. ILPS '1997*, Port Jefferson, Long Island, NY, US, pages 261–275. MIT Press, 13–16 Oct. 1997.
3. P.A. Abdulla, A. Annichini, S. Bensalem, A. Bouajjani, P. Habermehl, and L. Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. In N. Halbwachs and D. Peled, editors, *Proc. 11^{th} Int. Conf. CAV '99*, Trento, IT, LNCS 1633, pages 146–159. Springer-Verlag, 6–10 Jul. 1999.
4. P.A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In S. Graf and M.I. Schwartzbach, editors, *Proc. 6^{th} Int. Conf. TACAS '2000*, Berlin, DE, 25 Mar. – 2 Apr. 2000, LNCS 1785, pages 411–425. Springer-Verlag, 2000.
5. S. Abramsky and T.P. Jensen. A relational approach to strictness analysis for higher-order polymorphic functions. In *18^{th} POPL*, pages 49–54, Orlando, FL, 1991. ACM Press.
6. A. Aiken. Introduction to set constraint-based program analysis. *Sci. Comput. Programming, Special Issue on SAS'96*, 35(1):79–111, September 1999.
7. S.B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, C-27(6), 1978.
8. J. Aldrich, C. Chambers, E.M. Sirer, and S. Eggers. Static analyzes for eliminating unnecessary synchronization from Java programs. In A. Cortesi and G. Filé,

editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 18–38. Springer-Verlag, 1999.

9. M. Alt, Ferdinand C., F. Martin, and R. Wilhelm. Cache behavior prediction by abstract interpretation. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 52–66. Springer-Verlag, 1996.
10. M. Alt and F. Martin. Generation of efficient interprocedural analyzers with PAG. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 33–50. Springer-Verlag, 1995.
11. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoret. Comput. Sci. B*, 138:3–34, Jan. 1995.
12. G. Amato, F. Giannotti, and G. Mainetto. Data sharing analysis for a database programming language via abstract interpretation. In R. Agrawal, S. Baker, and D.A. Bell, editors, *Proc. 19th Int. Conf. VLDB '93*, Dublin, IE, pages 405–415. Morgan Kaufmann Pub., 24–27 Aug. 1993.
13. G. Amato, F. Giannotti, and G. Mainetto. Static analysis of transactions: an experiment of abstract interpretation usage. In A. Heuer and M.H. Scholl, editors, *Proc. 5th Workshop FMLDO '99*, Aigen, AT, 20–24 Sep. 1993, Informatik-Berichte des IfI 93/9, pages 19–29. Technische Universit t Clausthal, DE, 1993.
14. G. Amato, F. Giannotti, and G. Mainetto. Conservative multigranularity locking for an object-oriented persistent language via abstract interpretation. In S. Bergamaschi, C. Sartori, and P. Tiberio, editors, *Atti del Secondo Convegno Nazionale "Sistemi Evoluti per Basi di Dati", SEBD 94*, Rimini, IT, 6–8 June 1994, pages 329–349. Editrice Esculapio Progetto Leonardo, via U. Terracini, 30, 40131 Bologna, IT, 6–8 June 1994.
15. G. Amato and G. Levi. Abstract interpretation based semantics of sequent calculi. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 38–57. Springer-Verlag, 29 June – 1 Jul. 2000.
16. P.H. Andersen and Holst C.K. Termination analysis for offline partial evaluation of a higher order functional language. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 67–82. Springer-Verlag, 1996.
17. K.R. Apt and G.D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, Oct. 1986.
18. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of boolean functions for dependency analysis. *Sci. Comput. Programming*, 31(1):3–45, May 1998.
19. T. Armstrong, K. Marriott, P.r Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 266–280. Springer-Verlag, 1994.
20. K. Asai. Binding-time analysis for both static and dynamic expressions. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 117–133. Springer-Verlag, 1999.
21. J.M. Ashley. A practical and flexible analysis for higher-order languages. In *23rd POPL*, pages 184–194, St. Petersburg Beach, FL, 1996. ACM Press.
22. A.E. Ayers. Efficient closure analysis with reachability. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int.*

Work. WSA '92, Bordeaux, FR. BIGRE, volume 81–82, pages 126–134. IRISA, Rennes, FR, 23–25 Sep. 1992.

23. C. B ckstr m and P. Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. 17^{th} IJCAI '95*, pages 1599–1604, 1995.
24. R. Bagnara, R. Giacobazzi, and G. Levi. Static analysis of CLP programs over numeric domains. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 43–50. IRISA, Rennes, FR, 23–25 Sep. 1992.
25. R. Bagnara, P.H. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 53–67. Springer-Verlag, 1997.
26. R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of Pos. In A.M. Haeberer, editor, *Proc. 7^{th} Int. Conf. AMAST '98*, Amazonia, BR, 4–8 Jan. 1999, LNCS 1548, pages 471–485. Springer-Verlag, 1999.
27. C. Baier, E.M. Clarke, V. Hartonas-Garmhausen, M.Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24^{th} Int. Coll. ICALP '97*, volume 1256 of *Bologna, IT, 7–11 Jul. 1997, LNCS*, pages 430–440. Springer-Verlag, 7–11 Jul. 1997.
28. J. Bailey, L. Crnogorac, K. Ramamohanarao, and H. Søndergaard. Abstract interpretation of active rules and its use in termination analysis. In F.N. Afrati and P. Kolaitis, editors, *Proc. 6^{th} Int. Conf. ICDT '97*, Delphi, GR, LNCS 1186, pages 188–202. Springer-Verlag, 8–10 Jan. 1997.
29. J. Bailey and A. Poulovassilis. Abstract interpretation for termination analysis in functional active databases. *J. Int. Inf. Syst.*, 12(2–3):243–273, 1999.
30. C.A. Baker-Finch. Relevant logic and strictness analysis. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 221–228. IRISA, Rennes, FR, 23–25 Sep. 1992.
31. V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *Proc. ACM SIGPLAN '89 Conf. PLDI. ACM SIGPLAN Not. 24(7)*, pages 41–53, Portland, OR, US, 21–23 June 1989.
32. D. Baldan, N. Civran, G. Filé, and F. Pulvirenti. A simple and general method for integrating abstract interpretation in SICStus. In G. Nadathur, editor, *Proc. Int. Conf. PPDP '99*, Paris, FR, 29 Sep. – 1 Oct. 1999, LNCS 1702, pages 207–223. Springer-Verlag, 1999.
33. D. Baldan and G. Filé. Abstract interpretation for improving WAM code. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, page 364. Springer-Verlag, 1997.
34. A. Banerjee and D.A. Schmidt. Stackability in the simply-typed call-by-value lambda calculus. *Sci. Comput. Programming*, 31(1):47–73, May 1998.
35. G. Baraki. A note on abstract interpretation of polymorphic functions. In J. Hughes, editor, *Proc. 5^{th} FPCA*, LNCS 523, pages 367–378. Springer-Verlag, Aug. 1991.
36. G. Baraki and R.J.M. Hughes. Abstract interpretation of polymorphic functions. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow 1989*, Proc. 1989 Glasgow Workshop, Fraserburgh, UK. Springer-Verlag and BCS, 31–40 Aug. 1989.

37. R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini. Abstract interpretation of trace semantics for concurrent calculi. *Inf. Process. Lett.*, 70(2):69–78, Feb. 1999.
38. R. Barbuti, R. Giacobazzi, and G. Levi. A bottom-up polymorphic type inference in logic programming. *Sci. Comput. Programming*, 19(3):281–313, Dec. 1992.
39. R. Barbuti, R. Giacobazzi, and G. Levi. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *TOPLAS*, 15(1):133–181, Jan. 1993.
40. M. Beemster. Strictness optimization for graph reduction machines (why it might not be strict). *TOPLAS*, 16(5):1449–1466, Sep. 1994.
41. C. Beierle and G. Meyer. Using types as approximations for type checking Prolog programs. In A. Middeldorp and T. Sato, editors, *4th FLOPS '99*, Tsukuba, JP, 11–13 Nov. 1999, LNCS 1722, pages 251–266. Springer-Verlag, 1999.
42. M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informat.*, 20:207–226, 1983.
43. S. Bensalem, M. Bozga, J.-C. Fernandez, L. Ghirvu, and L. Lakhnech. A transformational approach for generating non-linear invariants. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 58–74. Springer-Verlag, 29 June – 1 Jul. 2000.
44. S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems compositionally and automatically. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Int. Conf. CAV '98*, Vancouver, BC, CA,LNCS 1427, pages 319–331. Springer-Verlag, 28 June – 2 Jul. 1998.
45. P.N. Benton. Strictness properties of lazy algebric datatypes. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 194–205. Springer-Verlag, Sep. 22–24, 1993.
46. V. Benzaken and X. Schaefer. Ensuring efficiently the integrity of persistent object systems via abstract interpretation. In R.C.H. Connor and S. Nettles, editors, *Proc. 7th Workshop POS (1996)*, Cape May, NJ, US, pages 72–87. Morgan Kaufmann Pub., 13–16 Oct. 1997.
47. S. Berezin, S.V.A. Campos, and E.M. Clarke. Compositional reasoning in model checking. In W.P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference, Int. Symp. COMPOS '97, Revised Lectures*, Bad Malente, DE, LNCS 1536, pages 81–102. Springer-Verlag, 8–12 Sep. 1997 1998.
48. D. Bert, R. Echahed, and K. Adi. Resolution of goals with the functional and logic programming language LPG: Impact of abstract interpretation. In M. Wirsing and M. Nivat, editors, *Proc. 5th Int. Conf. AMAST '96*, Munchen, DE, LNCS 1101, pages 629–632. Springer-Verlag, 1–5 Jul. 1996.
49. F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral analysis for synchronous languages. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 51–68. Springer-Verlag, 1999.
50. G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for CTL⋆. In *Proc. 10th LICS '95*, San Diego, CA, US, pages 388–397. IEEE Comp. Soc. Press, 26–29 June 1995.
51. A. Biere. μcke - efficient μ-calculus model checking. In O. Grumberg, editor, *Proc. 9th Int. Conf. CAV '97*, Haifa, IL, LNCS 1254, pages 468–471. Springer-Verlag, 22–25 Jul. 1997.
52. A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. 36th Conf. DAC '99*, New Orleans, LA, US, pages 317–320. ACM Press, 21–25 June 1999.

53. A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In W.R. Cleaveland, editor, *Proc. 5th Int. Conf. TACAS '99*, Amsterdam, NL, 22-25 Mar. 1999, LNCS 1579, pages 193–207. Springer-Verlag, 1999.
54. A. Biere, E.M. Clarke, R. Raimi, and Y. Zhu. Properties of a power PC microprocessor using symbolic model checking without BDDs. In N. Halbwachs and D. Peled, editors, *Proc. 11th Int. Conf. CAV '99*, Trento, IT, LNCS 1633, pages 60–71. Springer-Verlag, 6–10 Jul. 1999.
55. P.A. Bigot, S.K. Debray, and K. Marriott. Understanding finiteness analysis using abstract interpretation. In K.R. Apt, editor, *Proc. JICSLP '92*, Washington, DC, US, pages 735–749. MIT Press, Nov. 1992.
56. G. Birkhoff. *Lattice Theory*, volume 25 of *Colloquium publications*. AMS, 3rd edition, 1973.
57. S. Bistarelli, P. Codognet, and F. Rossi. An abstraction framework for soft constraints and its relationship with constraint propagation. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 71–86. Springer-Verlag, 26–29 Jul. 2000.
58. B. Blanchet. Escape analysis: Correctness proof, implementation and experimental results. In *25th POPL*, pages 25–37, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
59. B. Blanchet. Escape analysis for object-oriented languages: Application to Java. In *Proc. ACM SIGPLAN Conf. OOPSLA '99. ACM SIGPLAN Not. 34(10)*, pages 20–34, Denver, CO, US, 1–5 Nov. 1999.
60. A. Bloss. Path analysis and the optimization of nonstrict functional languages. *TOPLAS*, 16(3):328–369, 1994.
61. A. Bloss and P. Hudak. Path semantics. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proc. 3rd workshop on Mathematical Foundations of Programming Languages Semantics*, LNCS 298, pages 476–489. Springer-Verlag, Apr. 1986.
62. M. Blume. Dependency analysis for Standard ML. *TOPLAS*, 21(4):790–812, Jul. 1999.
63. C. Bodei, P. Degano, and C. Priami. Constructing specific SOS semantics for concurrency via abstract interpretation. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 168–183. Springer-Verlag, 1998.
64. C. Bodei and C. Priami. True concurrency via abstract interpretation. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 202–216. Springer-Verlag, 1997.
65. R. Bod k and S. Anik. Path-sensitive value-flow analysis. In *25th POPL*, pages 237–251, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
66. B. Boigelot and P. Godefroid. Model checking in practice: An analysis of the ACCESS.bus protocol using SPIN. In M.C. Gaudel and J. Woodcock, editors, *Industrial Benefit and Advances in Formal Methods, 3rd Int. Symp. of Formal Methods Europe, FME '96: Industrial Benefit of Formal Methods*, Oxford, UK, LNCS 1051, pages 465–478. Springer-Verlag, 18–22 Mar. 1996.
67. B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs (extended abstract). In R. Alur and T.A. Henzinger, editors, *Proc. 8th Int. Conf. CAV '96*, New Brunswick, NJ, US, LNCS 1102, pages 1–12. Springer-Verlag, 31 Jul. –3 Aug. 1996.
68. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs (extended abstract). In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 172–186. Springer-Verlag, 1997.

69. A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Sci. Comput. Programming*, 18:247–269, 1992.
70. D. Boucher and M. Feeley. Abstract compilation: A new implementation paradigm for static analysis. In T. Gyimothy, editor, *Proc. 6th Int. Conf. CC '96*, Link ping, SE, LNCS 1060, pages 192–207. Springer-Verlag, 24–26 Apr. 1996.
71. D. Boulanger and M. Bruynooghe. Deriving fold/unfold transformations of logic programs using extended OLDT-based abstract interpretation. *J. Symbolic Comput.*, 15(5 & 6):495–521, 1993.
72. D. Boulanger, M. Bruynooghe, and D. De Schreye. Compiling control revisited: A new approach based upon abstract interpretation for constraint logic programs. In M. Ducassé, B. Le Charlier, Y.-J. Lin, and L. . Yalçinalp, editors, *Proc. 5th Workshop LPE 1993*, Vancouver, BC, CA, pages 39–51. IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, FR, 29–30 Oct. 1993.
73. D.Y. Boulanger. Deep logic program transformation using abstract interpretation. In A. Voronkov, editor, *Proc. 1st & 2nd Russian Conf. on Logic Programming*, Irkutsk, RU, 14–18 Sep. 1990 & St. Petersburg, RU, 11–16 Sep. 1991 LNCS 592, pages 79–101. Springer-Verlag, 1992.
74. F. Bourdoncle. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In P. Deransart and J. Małuszyński, editors, *Proc. Int. Work. PLILP '90*, Link ping, SE, LNCS 456, pages 307–323. Springer-Verlag, 20–22 Aug. 1990.
75. F. Bourdoncle. Abstract interpretation by dynamic partitioning. *J. Func. Prog.*, 2(4):407–435, 1992.
76. F. Bourdoncle. Abstract debugging of higher-order imperative languages. In *Proc. ACM SIGPLAN '93 Conf. PLDI. ACM SIGPLAN Not. 28(6)*, pages 46–55, Albuquerque, NM, US, 23–25 June 1993. ACM Press.
77. F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. FMPA*, Akademgorodok, Novosibirsk, RU, LNCS 735, pages 128–141. Springer-Verlag, 28 June –2 Jul. 1993.
78. F. Bourdoncle. Assertion-based debugging of imperative programs by abstract interpretation. In I. Sommerville and M. Paul, editors, *Proc. 4th ESEC '93*, Garmisch-Partenkirchen, DE, 13–17 Sep. 1993, LNCS 717, pages 501–516. Springer-Verlag, 1999.
79. M. Bozga, C. Daws, O. Maler, A. Olivero, and S. Tripakis. Kronos: A model-checking tool for real-time systems. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Int. Conf. CAV '98*, Vancouver, BC, CA,LNCS 1427, pages 546–550. Springer-Verlag, 28 June – 2 Jul. 1998.
80. M. Bozga, J.C. Fernandez, and L. Ghirvu. State space reduction based on live variables analysis. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 164–178. Springer-Verlag, 1999.
81. J. Brauburger. Automatic termination analysis for partial functions using polynomial orderings. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 330–344. Springer-Verlag, 1997.
82. J. Brauburger and J. Giesl. Termination analysis for partial functions. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 113–127. Springer-Verlag, 1996.
83. J. Brauburger and J. Giesl. Approximating the domains of functional and imperative programs. *Sci. Comput. Programming, Special Issue on SAS'96*, 35(1):113–136, September 1999.

84. M. Bruynooghe and M. Codish. Freeness, sharing, linearity and correctness — all at once. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3[rd] Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 153–164. Springer-Verlag, 22–24 Sep. 1993.
85. M. Bruynooghe, M. Codish, and A. Mulkers. Abstract unification for a composite domain deriving sharing and freeness properties of program variables. In *ICLP '94 post-Conf. Workshop on the verification and analysis of logic programs*, pages 213–230, Santa Margherita Ligure, IT, June 1994.
86. M. Bruynooghe, M. Codish, and A. Mulkers. Abstracting unification: A key step in the design of logic program analyses. In *Computer Science Today*, volume 1000 of *LNCS*, pages 406–425. Springer-Verlag, 1995.
87. M. Bruynooghe, B. Demoen, D. Boulanger, M. Denecker, and A. Mulkers. A freeness and sharing analysis of logic programs based on a pre-interpretation. In R. Cousot and D.A. Schmidt, editors, *Proc. 3[rd] Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 128–142. Springer-Verlag, 1996.
88. M. Bruynooghe and G. Janssens. An instance of abstract interpretation integrating type and mode inferencing (extended abstract). In R. Kowalski and K. Bowen, editors, *Proc. 5[th] Int. Conf. & Symp. on Logic Programming, Volume 1*, Seattle, WA, US, pages 669–683. MIT Press, 15–19 Aug. 1988.
89. M. Bruynooghe, G. Janssens, A. Callebaut, and B. Demoen. Abstract interpretation: towards the global optimization of Prolog programs. In *Proc. 1987 Int. Symp. on Logic Programming*, San Francisco, CA, pages 192–204. IEEE Comp. Soc. Press, 31 Aug. – 4 Sep. 1987.
90. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, C-35(8), 1986.
91. F. Bueno, M.J. Garc a de la Banda, and M.V. Hermenegildo. Effectiveness of abstract interpretation in automatic parallelization: A case study in logic programming. *TOPLAS*, 21(2):189–239, Mar. 1999.
92. F. Bueno, M.J. Garc a de la Banda, and M.V. Hermenegildo. Effectiveness of global analysis in strict independence-based automatic parallelization. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 320–336. MIT Press, 13–17 Nov. 1994.
93. T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In O. Grumberg, editor, *Proc. 9[th] Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 400–411. Springer-Verlag, 22–25 Jul. 1997.
94. T. Bultan, R. Gerber, and W. Pugh. Model checking concurrent systems with unbounded variables, symbolic representations, approximations and experimental results. *TOPLAS*, 21(4):747–789, Jul. 1999.
95. M.A. Bulyonkov. Extracting polyvariant binding time analysis from polyvariant specializer. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 59–65. ACM Press, 1993.
96. M.A. Bulyonkov and D.V. Kochetov. Grammar approach to alias analysis. *Programming*, 3:36–46, 1996.
97. M.A. Bulyonkov and V.Ja. Kurlyandchik. Polyvariant binding time analysis for high-order programs. In *Tools and methgods for program development*, pages 29–41. Institute of Informatics Systems, Novosibirsk, RU, 1995. In russian.
98. A. Bundy, F. Giunchiglia, R. Sebastiani, and T. Walsh. Calculating criticalities. *Art. Int.*, 88(1–2):39–67, Dec. 1996.
99. A. Bundy, F. Giunchiglia, R. Sebastiani, and T. Walsh. Computing abstraction hierarchies by numerical simulation. In *Proc. 30[th] Nat. Conf. AAAI '96*, pages 523–529, Portland, OR, US, 4–8 Aug. 1996. AAAI Press / MIT Press.

100. A. Bundy, F. Giunchiglia, A. Villafiorita, and T. Walsh. Abstract proof checking: An example motivated by an incompleteness theorem. *J. Autom. Reason.*, 19(3):319–346, Dec. 1997.
101. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inform. and Comput.*, 98(2):142–170, June 1992.
102. G. Burn. The abstract interpretation of functional languages. In G. Burn, S. Gay, and M. Ryan, editors, *Theory and Formal Methods 1993*, Workshops in Comp. 724, pages 3–14. Springer-Verlag, Isle of Thorns Conf. Center, Chelwood Gate, Sussex, UK, 29–31 Mar. 1993.
103. G. Burn and D. Le Métayer. Proving the correctness of compiler optimizations based on strictness analysis. In M. Bruynooghe and J. Penjam, editors, *Proc. 5th Int. Symp. PLILP '93*, Tallinn, EE, 25–27, Aug. 1993, LNCS 714, pages 346–364. Springer-Verlag, 1993.
104. G.L. Burn. A relationship between abstract interpretation and projection analysis (extended abstract). In *17th POPL*, pages 151–156, San Francisco, CA, 1990. ACM Press.
105. G.L. Burn. *Lazy Functional Languages: Abstract Interpretation and Compilation.* Research Monographs in Parallel and Distributed Computing. Pitman and MIT Press, 1991.
106. G.L. Burn, C.L. Hankin, and S. Abramsky. Strictness analysis of higher-order functions. *Sci. Comput. Programming*, 7:249–278, Nov. 1986.
107. G.L. Burn, C.L. Hankin, and S. Abramsky. The theory of strictness analysis for higher-order functions. In H. Ganzinger and N.D. Jones, editors, *Programs as Data Objects, Proceedings of a Workshop*, Copenhagen, DK, 17–19 Oct. 1985, LNCS 217, pages 42–62. Springer-Verlag, 1986.
108. M.-M. Corsini C. Codognet, P. Codognet. Abstract interpretation for concurrent logic languages. In S.K. Debray and M.V. Hermenegildo, editors, *NACLP 1997*, Austin, TX, US, pages 215–232. MIT Press, 29 Oct. – 1 Nov. 1990.
109. S.V.A. Campos and E.M. Clarke. Analysis and verification of real-time systems using quantitative symbolic algorithms. *STTT*, 2(3):260–269, 1999.
110. S.V.A. Campos, E.M. Clarke, W. Marrero, and M. Minea. Verus: A tool for quantitative analysis of finite-state real-time systems. In *Proc. ACM SIGPLAN 1995 Workshop on Languages, Compilers & Tools for Real-Time Systems*, pages 75–83, La Jolla, CA, 21–22 June 1995.
111. S.V.A. Campos, E.M. Clarke, and M. Minea. The Verus tool: A quantitative approach to the formal verification of real-time systems. In O. Grumberg, editor, *Proc. 9th Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 452–455. Springer-Verlag, 22–25 Jul. 1997.
112. D. Cansell and D. Méry. Abstract animator for temporal specifications: Application to TLA. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 284–299. Springer-Verlag, 1999.
113. C. Carreras and M.V. Hermenegildo. Grid-based histogram arithmetic for the probabilistic analysis of functions. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 107–123. Springer-Verlag, 26–29 Jul. 2000.
114. R. Cartwright and M. Felleisen. Program verification through soft typing. *ACM Comput. Surv.*, 28:349–351, June 1996.
115. Y. Caseau. Abstract interpretation of constraints on order-sorted domains. In K. Ueda V.A. Saraswat, editor, *Proc. 1991 Int. Symp. ISLP '91*, San Diego, CA, US, pages 435–452. MIT Press, 28 Oct. – 1 Nov. 1997.

116. P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. LUSTRE: a declarative language for programming synchronous systems. In *14th POPL*, Munchen, DE, 1987. ACM Press.
117. H. Casse, L. Feraud, C. Rochange, and P. Sinrat. Using abstract interpretation techniques for static pointer analysis. In *The Third Workshop on Interaction Between Compilers and Computer Architectures*, San Jose, CA, October 7, 1998.
118. J. Chang and A.M. Despain. Semi-intelligent backtracking of Prolog based on static data dependency analysis. In *Proc. 1985 Int. Symp. on Logic Programming*, Boston, MA, pages 10–21. IEEE Comp. Soc. Press, Jul. 1985.
119. J. Chang, A.M. Despain, and D. DeGroot. AND-parallelism of logic programs based on a static data dependency analysis. In *Digest of Papers, COMPCON 85*, pages 218–225. IEEE Comp. Soc. Press, Feb. 1985.
120. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Proc. 4th Int. Conf. TACAS '98*, Lisbon, PT, LNCS 1384, pages 358–375. Springer-Verlag, 28 Mar. – 4 Apr. 1998.
121. B. Le Charlier. Abstract interpretation and finite domain symbolic constraints. In A. Podelski, editor, *Constraint Programming: Basics and Trends, Selected Papers*, Ch tillon Spring School, Ch tillon-sur-Seine, FR, 16–20 May 1994, LNCS 910, pages 147–170. Springer-Verlag, 1994.
122. B. Le Charlier and P. Van Hentenryck. Groundness analysis for Prolog: Implementation and evaluation of the domain `Prop`. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 99–110. ACM Press, 1993.
123. B. Le Charlier, S. Rossi, and P. van Hentenryck. An abstract interpretation framework which accurately handles Prolog search-rule and the cut. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 157–171. MIT Press, 13–17 Nov. 1994.
124. B. Le Charlier and P. van Hentenryck. On the design of generic abstract interpretation frameworks. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 229–246. IRISA, Rennes, FR, 23–25 Sep. 1992.
125. S. Chatterjee, B.E. Blelloch, and A.L. Fisher. Size and access inference for data-parallel programs. In *Proc. ACM SIGPLAN '91 Conf. PLDI. ACM SIGPLAN Not. 26(6)*, pages 130–144, Toronto, Ontario, CA, 26–28 June 1991.
126. T. Cheatham, H. Gao, and D. Stefanescu. A suite of analysis tools based on a general purpose abstract interpretor. In P.A. Fritzson, editor, *Proc. 5th Int. Conf. CC '94*, Edinburg, UK, LNCS 786, pages 188–202. Springer-Verlag, Apr. 1994.
127. T. Cheatham and D.C. Stefanescu. A suite of optimizers based on abstract interpretation. In *Proc. PEPM '92*, San Francisco, CA, US, pages 75–81. Yale University, Tech. rep. TR YALEU/DCS/RR-90, 19–20 June 1995.
128. J. Chen and J. Staples. Defining soft sortedness by abstract interpretation. In A.M. Borzyszkowski and S. Sokolowski, editors, *Proc. 18th Int. Symp. MFCS '93*, Gdansk, PL, 30 Aug. – 3 Sep. 1993, LNCS 711, pages 362–371. Springer-Verlag, 20–22 Aug. 1990.
129. Li-Ling Chen, W.L. Harrison III, and Kwangkeun Yi. Efficient computation of fixpoints that arise in complex program analysis. *Journal of Programming Languages*, 3(1):31–68, 1995.
130. Y-A. Chen, E.M. Clarke, P.H. Ho, Y. Hoskote, T. Kam, M. Khaira, J. O'Leary, and X. Zhao. Verification of all circuits in a floating-point unit using word-level model checking. In M.S. Srivas and A.J. Camilleri, editors, *Proc. 1st Int. Conf. on Formal Methods in Computer-Aided Design, FMCAD '96*, number 1166 in LNCS, pages 19–33, Palo Alto, CA, US, 6–8 Nov. 1996. Springer-Verlag.

131. S.-C. Cheung and J. Kramer. Tractable flow analysis for anomaly detection in distributed programs. In I. Sommerville and M. Paul, editors, *Proc. 4^{th} ESEC '93*, Garmisch-Partenkirchen, DE, 13–17 Sep. 1993, LNCS 717, pages 283–300. Springer-Verlag, 1999.
132. K. Cho and . Ueda. Diagnosing non-well-moded concurrent logic programs. In M.J. Maher, editor, *Proc. JICSLP '96*, Bonn, DE, pages 215–229. MIT Press, 2–6 Sep. 1996.
133. J.-H. Chow and W.L. III Harrison. Compile-time analysis of parallel programs that share memory. In *19^{th} POPL*, pages 130–141, Albuquerque, NM, 1992. ACM Press.
134. J.-H. Chow and W.L. III Harrison. State space reduction in abstract interpretation of parallel programs. In *Proc. 1994 ICCL*, Toulouse, FR, pages 277–288. IEEE Comp. Soc. Press, 16–19 May 1994.
135. T.-R. Chuang and B. Goldberg. A syntactic approach to fixed point computation on finite domains. *LISP Pointers*, 5(1):109–118, Jan. – Mar. 1992.
136. D. Clark and C. Hankin. A lattice of abstract graphs. In M. Bruynooghe and J. Penjam, editors, *Proc. 5^{th} Int. Symp. PLILP '93*, Tallinn, EE, 25–27, Aug. 1993, LNCS 714, pages 318–331. Springer-Verlag, 1993.
137. D. Clark, C. Hankin, and S. Hunt. Safety of strictness analysis via term graph rewriting. In J. Palsberg, editor, *Proc. 7^{th} Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 95–114. Springer-Verlag, 29 June – 1 Jul. 2000.
138. E.M. Clarke. Synthesis of resource invariants for concurrent programs. In *6^{th} POPL*, pages 211–221. ACM Press, Jan. 1979.
139. E.M. Clarke. Synthesis of resource invariants for concurrent programs. *TOPLAS*, 2(3):338–358, 1980.
140. E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *IBM Workshop on Logics of Programs*, Yorktown Heights, NY, US, LNCS 131. Springer-Verlag, May 1981.
141. E.M. Clarke, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model checking. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10^{th} Int. Conf. CAV '98*, Vancouver, BC, CA,LNCS 1427, pages 147–158. Springer-Verlag, 28 June – 2 Jul. 1998.
142. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *10^{th} POPL*, pages 117–126. ACM Press, Jan. 1983.
143. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *19^{th} POPL*, pages 343–354, Albuquerque, NM, 1992. ACM Press.
144. E.M. Clarke, O. Grumberg, and D.E. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of concurrency–Reflections and Perspectives*, LNCS 803. Springer-Verlag, 1994.
145. E.M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *STTT*, 2(3):279–287, 1999.
146. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 1999.
147. E.M. Clarke, S. Jha, Y. Lu, and D. Wang. Abstract BDDs: A technique for using abstraction in model checking. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods, Proc. 10^{th} IFIP WG 10.5 Adv. Res. Work. Conf. CHARME '99*, Bad Herrenalp, DE, LNCS 1703, pages 172–186. Springer-Verlag, 27–29 Sep. 1999.

148. E.M. Clarke, S. Jha, and W.R. Marrero. Partial order reductions for security protocol verification. In S. Graf and M.I. Schwartzbach, editors, *Proc. 6th Int. Conf. TACAS '2000*, Berlin, DE, 25 Mar. – 2 Apr. 2000, LNCS 1785, pages 503–518. Springer-Verlag, 2000.
149. R. Cleaveland. The Concurrency Factory: A development environment for concurrent systems. In R. Alur and T. Henzinger, editors, *Proc. 8th Int. Conf. CAV '96*, New Brunswick, NJ, LNCS 1102, pages 398–401. Springer-Verlag, Jul. 1996.
150. R. Cleaveland. The Concurrency Factory software development environment. In T. Margaria and B. Steffen, editors, *Proc. 2nd Int. Conf. TACAS '96*, number 1055 in LNCS, pages 391–395. Springer-Verlag, Passau, DE, Mar. 1996.
151. R. Cleaveland. Efficient local model checking for fragments of the modal μ-calculus. In T. Margaria and B. Steffen, editors, *Proc. 2nd Int. Conf. TACAS '96*, number 1055 in LNCS, pages 107–126. Springer-Verlag, Passau, DE, Mar. 1996.
152. R. Cleaveland. Efficient model checking via the equational μ-calculus. In *Proc. 11th LICS '96*, pages 304–312. IEEE Comp. Soc. Press, New Brunswick, NJ, Jul. 1996.
153. R. Cleaveland. Formal timing analysis for fault-tolerant active structural control systems. In *Proc. 1st Workshop on Formal Methods in System Practice*, San Diego, CA, Jan. 1996. A SCP journal version subsumes this paper.
154. R. Cleaveland. The NCSU concurrency workbench. In R. Alur and T. Henzinger, editors, *Proc. 8th Int. Conf. CAV '96*, New Brunswick, NJ, LNCS 1102, pages 394–397. Springer-Verlag, Jul. 1996.
155. R. Cleaveland. Modeling and verifying active structural control systems. *SCICP*, 29(1–2):99–122, Jul. 1997.
156. R. Cleaveland, P. Iyer, and D. Yankelevitch. Optimality in abstractions of model checking. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 51–63. Springer-Verlag, 1995.
157. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal Mu-calculus. In K.G. Larsen and A. Skou, editors, *Proc. 3rd Int. Work. CAV '91*, Aalborg, DK, LNCS 575, pages 48–58. Springer-Verlag, 1–4 Jul. 1991, 1992.
158. W.R. Cleaveland, editor. *Finite State Verification for the Asynchronous π-Calculus*, Amsterdam, NL, LNCS 1579. Springer-Verlag, 22–28 Mar. 1999.
159. W.R. Cleaveland, editor. *On Proving Safety Properties by Integrating Static Analysis, Theorem Proving and Abstraction*, Amsterdam, NL, LNCS 1579. Springer-Verlag, 22–28 Mar. 1999.
160. W.R. Cleaveland, editor. *Proving the Soundness of a Java Bytecode Verifier Specification in Isabelle/HOL*, Amsterdam, NL, LNCS 1579. Springer-Verlag, 22–28 Mar. 1999.
161. W.R. Cleaveland, editor. *Timed Diagnostics for Reachability Properties*, Amsterdam, NL, LNCS 1579. Springer-Verlag, 22–28 Mar. 1999.
162. M. Codish. Efficient goal directed bottom-up evaluation of logic programs. *J. Logic Programming*, 38(3):354–370, 1999.
163. M. Codish. Worst-case groundness analysis using positive boolean functions. *J. Logic Programming*, 41(1):125–128, 1999.
164. M. Codish, D. Dams, G. Fil , and M. Bruynooghe. Freeness analysis for logic programs – and correctness? In D.S. Warren, editor, *Proc. 10th ICLP '93*, Budapest, HU, pages 116–131. MIT Press, 21–25 June 1993.

165. M. Codish, D. Dams, and E. Yardeni. Abstract unification for the analysis of groundness and aliasing in logic programs. Tech. rep. TR-CS90–10, Weizmann Institute of Science, Department of applied mathematics and computer science, Aug. 1990.
166. M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In K. Furukawa, editor, *Proc. 8th ICLP '91*, Paris, FR, pages 79–93. MIT Press, 24–28 June 1991.
167. M. Codish, M. Garc a de la Banda, M. Bruynooghe, and M. Hermenegildo. Exploiting goal independence in the analysis of logic programs. *J. Logic Programming*, 32(3):247–261, 1997.
168. M. Codish, S. Debray, and R. Giacobazzi. Compositional analysis of modular logic programs. In *20th POPL*, pages 451–464, Charleston, SC, 1993. ACM Press.
169. M. Codish and B. Demoen. Analysing logic programs using `Prop`-ositional logic programs and a magic wand. In D. Miller, editor, *Proc. Int. Symp. ILPS '1993*, Vancouver, BC, CA, pages 114–129. MIT Press, 1993.
170. M. Codish and B. Demoen. Deriving polymorphic type dependencies for logic programs using multiple incarnations of Prop. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 281–296. Springer-Verlag, 1994.
171. M. Codish and B. Demoen. Analysing logic programs using `Prop`-ositional logic programs and a magic wand. *J. Logic Programming*, 25(3):249–274, 1995.
172. M. Codish, B. Demoen, and K. Sagonas. Semantic-based program analysis for logic-based languages using XSB. *STTT*, 2(1):29–45, 1998.
173. M. Codish, M. Falaschi, and K. Marriott. Suspension analysis of concurrent logic programs. In K. Furukawa, editor, *Proc. 8th ICLP '91*, Paris, FR, pages 331–345. MIT Press, 24–28 June 1991.
174. M. Codish, M. Falaschi, and K. Marriott. Suspension analyses for concurrent logic programs. *TOPLAS*, 16(3):649–686, May 1994.
175. M. Codish, M. Falaschi, and K. Marriott. Suspension analysis for concurrent logic programs. *TOPLAS*, 16(3):649–686, 1994.
176. M. Codish, M. Falaschi, and K. Marriott. Suspension analysis of concurrent logic programs. *TOPLAS*, 16(3):649–686, May 1994.
177. M. Codish, M. Falaschi, K. Marriott, and W. Winsborough. Efficient analysis of reactive properties of concurrent constraint logic programs. In *12th ICALP*, LNCS. SPRINGER, Jul. 1993.
178. M. Codish, M. Falaschi, K. Marriott, and W. Winsborough. A confluent semantic basis for the analysis of concurrent constraint logic programs. *J. Logic Programming*, 30(1):53–81, 1997.
179. M. Codish, J. Gallagher, and E. Shapiro. Using safe approximations of fixed points for analysis of logic programs. In H. Abramson and M.H. Rogers, editors, *Meta-programming in Logic Programming*, pages 233–262. MIT Press, 1989.
180. M. Codish and V. Lagoon. Type dependencies for logic programs using aci-unification. *Theoret. Comput. Sci.*, 238:131–159, 2000.
181. M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 68–82. Springer-Verlag, 1997.
182. M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. *J. Logic Programming*, 41(2):110–149, 2000.
183. M. Codish, K. Marriott, and C. Taboch. Improving program analyses by structure untupling. *J. Logic Programming*, 43(3):251–263, 2000.

184. M. Codish and G. Mashevitzky. Proving implications by algebraic approximation. *Theoret. Comput. Sci.*, 165:57–74, 1996.
185. M. Codish, A. Mulkers, M. Bruynooghe, M. Garc a de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 194–205. ACM Press, 1993.
186. M. Codish, A. Mulkers, M. Bruynooghe, M. Garc a de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. *TOPLAS*, 17(1):28–44, Jan. 1995.
187. M. Codish and H. Søndergaard. The boolean logic of set sharing analysis. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proc. 10^th Int. Symp. PLILP '98*, pages 89–101. Springer-Verlag, Pisa, IT, 16–18 Sep. 1998, LNCS 1490, 1998.
188. M. Codish, H. Søndergaard, and P.J. Stuckey. Sharing and groundness dependencies in logic programs. *TOPLAS*, 21(5):948–976, Sep. 1999.
189. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs goal. *J. Logic Programming*, 41(1):103–123, 1999.
190. P. Codognet and G. Filé. Computations, abstractions and constraints (abstract). *Actes JTASPEFL '91, Bordeaux, FR. BIGRE*, 74:70–71, Oct. 1991.
191. C. Colby. Analyzing the communication topology of concurrent languages. In *Proc. PEPM '95*, La Jolla, CA, 21–23 June 1995, pages 202–213. ACM Press, June 1995.
192. C. Colby. Determining storage properties of sequential and concurrent programs with assignment and structured data. In A. Mycroft, editor, *Proc. 2^nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 64–81. Springer-Verlag, 1995.
193. C. Colby and P. Lee. Trace-based program analysis. In *23^rd POPL*, pages 195–207, St. Petersburg Beach, FL, 1996. ACM Press.
194. M. Colón and T.E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10^th Int. Conf. CAV '98*, Vancouver, BC, CA,LNCS 1427, pages 293–304. Springer-Verlag, 28 June – 2 Jul. 1998.
195. M.A. Colón and T.E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10^th Int. Conf. CAV '98*, Vancouver, BC, CA, LNCS 1427, pages 293–304. Springer-Verlag, June /Jul. 1998.
196. M. Comini, R. Gori, G. Levi, and P. Volpe. Abstract interpretation based verification of logic programs. *ENTCS*, 30(1), 1999.
197. M. Comini, G. Levi, M. Chiara Meo, and G. Vitiello. Abstract diagnosis. *J. Logic Programming*, 39(1–3):43–93, 1999.
198. M. Comini, G. Levi, M.C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dam, editor, *Analysis and Verification of Multiple-Agent Languages, 5^th LOMAPS Workshop*, Stockhlom, SE, 24–26 June 1996, LNCS 1192, pages 22–50. Springer-Verlag, 1997.
199. M. Comini, G. Levi, and G. Vitiello. Abstract debugging of logic program. In L. Fribourg and F. Turini, editors, *Proc. Int. Work. LOPSTR '94 and Int. Symp. META '94*, Pisa, IT, 20–21 June 1994, LNCS 883, pages 440–450. Springer-Verlag, 1994.
200. M. Comini, G. Levi, and G. Vitiello. Efficient detection of incompleteness errors in the abstract debugging of logic programs. In A. Cortesi and G. Filé, editors, *Proc. 2^nd Int. Work. AADEBUG '95*, Saint Malo, FR, 22–24 May 1995, pages 159–174. IRISA-CNRS, Rennes, FR, 1995.

201. C. Consel. Polyvariant binding-time analysis for applicative languages. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 66–77. ACM Press, 1993.
202. C. Consel. Fast strictness analysis via symbolic fixpoint iteration. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 423–431. Springer-Verlag, 1994.
203. M. Coppo and A. Ferrari. Type inference, abstract interpretation and strictness analysis. *Theoret. Comput. Sci.*, 121:113–143, 1993.
204. Th. Coquand and G.P. The calculus of constructions. *Inform. and Comput.*, 76(2/3):95–120, Feb. /Mar. 1988.
205. A. Cortesi, D. Dams, G. Filé, and M. Bruynooghe. On the design of a correct freeness analysis for logic programs. *J. Logic Programming*, 28(3):181–206, 1996.
206. A. Cortesi and G. Filé. Abstract interpretation of logic programs: an abstract domain for groundness, sharing, freeness and compoundness analysis. In P. Hudak and N.D. Jones, editors, *Proc. PEPM '91*, Yale U., New Haven, CT, US, 17–19 June 1991, ACM SIGPLAN Not. 26(9), pages 52–61. ACM Press, Sep. 1991.
207. A. Cortesi and G. Filé. Sharing is optimal. *J. Logic Programming*, 38(3):371–386, 1999.
208. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 100–117. Springer-Verlag, 1995. Full version in [209].
209. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *TOPLAS*, 19(1):7–47, Jan. 1997.
210. A. Cortesi, G. Filé, and W. Winsborough. Prop revisited: propositional formulas as abstract domains for groundness analysis. In G. Kahn, editor, *Proc. 6th LICS'91, Amsterdam, NL*, pages 322–327. IEEE Comp. Soc. Press, 15–18 Jul. 1991.
211. A. Cortesi, G. Filé, and W.H. Winsborough. Comparison of abstract interpretations. In W. Kuich, editor, *19th ICALP*, Vienna, AT, LNCS 623, pages 521–532. Springer-Verlag, 13–17 Jul. 1992.
212. A. Cortesi, G. Filé, and W.H. Winsborough. Optimal groundness analysis using propositional logic. *J. Logic Programming*, 27(2):137–167, 1996.
213. A. Cortesi, B. Le Charlier, and P. van Hentenryck. Combinations of abstract domains for logic programming. In *24th POPL*, pages 227–239. ACM Press, 1994.
214. O. Coudert, C. Berthet, and J.C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proc. Int. Work. on Automatic Verification Methods for Finite State Systems, Grenoble, FR, June 1989*, LNCS 407, pages 365–373. Springer-Verlag, 1990.
215. O. Coudert, J.C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In E.M. Clarke and R.P. Kurshan, editors, *CAV '90*, number 3 in DIMACS Volume Series, pages 75–84. AMS, June 1990.
216. P. Cousot. Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice. Res. rep. R.R. 88, Laboratoire IMAG, Université scientifique et médicale de Grenoble, Grenoble, FR, Mar. 1978. 34 p.
217. P. Cousot. *M thodes it ratives de construction et d'approximation de points fixes d'op rateurs monotones sur un treillis, analyse s mantique de programmes.* Th se d'État s sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, FR, 21 Mar. 1978.

218. 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, 1981.
219. 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, 1990.
220. P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Comput. Surv.*, 28(2):324–328, 1996.
221. P. Cousot. Program analysis: The abstract interpretation perspective. *ACM Comput. Surv.*, 28A(4es):165–es, Dec. 1996.
222. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *ENTCS*, 6, 1997. `http://www.elsevier.nl/locate/entcs/volume6.html`, 25 pages.
223. P. Cousot. Design of semantics by abstract interpretation, invited address. In *Mathematical Foundations of Programming Semantics, 30th Annual Conf. (MFPS XIII)*, Carnegie Mellon University, Pittsburgh, PA, US, 23–26 Mar. 1997.
224. P. Cousot. Program analysis: The abstract interpretation perspective. *ACM SIGPLAN Not.*, 32:73–76, 1997.
225. P. Cousot. Types as abstract interpretations, invited paper. In *24th POPL*, pages 316–331, Paris, FR, Jan. 1997. ACM Press.
226. P. Cousot. The Marktoberdorf'98 generic abstract interpreter. `http://www.di.ens.fr/~cousot/Marktoberdorf98.shtml`, Nov. 1998.
227. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbr ggen, editors, *Calculational System Design*, volume 173, pages 421–505. NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, 1999.
228. P. Cousot. Directions for research in approximate system analysis. *ACM Comput. Surv.*, 31(3es), Sep. 1999.
229. P. Cousot. Abstract interpretation: Achievements and perspectives, invited paper. In *Proc. SSGRR 2000 Computer & eBusiness International Conference*, Compact disk paper 224, L'Aquila, IT, 31 Jul. – 6 Aug. 2000. Scuola Superiore G. Reiss Romoli.
230. P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 1–25. Springer-Verlag, 26–29 Jul. 2000.
231. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, To appear (Preliminary version in [222]).
232. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2nd Int. Symp. on Programming*, pages 106–130. Dunod, 1976.
233. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.
234. P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. In *ACM Symposium on Artificial Intelligence & Programming Languages*, Rochester, NY, ACM SIGPLAN Not. 12(8):1–12, 1977.
235. P. Cousot and R. Cousot. Static determination of dynamic properties of generalized type unions. In *ACM Symposium on Language Design for Reliable Software*, Raleigh, NC, ACM SIGPLAN Not. 12(3):77–94, 1977.

236. P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E.J. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, pages 237–277. North-Holland, 1977.
237. 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. *Portugal. Math.*, 38(2):185–198, 1979.
238. P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.*, 82(1):43–57, 1979.
239. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6^{th} POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.
240. P. Cousot and R. Cousot. Semantic analysis of communicating sequential processes. In J.W. de Bakker and J. van Leeuwen, editors, *7^{th} ICALP*, LNCS 85, pages 119–133. Springer-Verlag, Jul. 1980.
241. 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, 1984.
242. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. Logic Programming*, 13(2–3):103–179, 1992. (The editor of J. Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.).
243. P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, Aug. 1992.
244. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4^{th} Int. Symp. PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer-Verlag, 1992.
245. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *19^{th} POPL*, pages 83–94, Albuquerque, NM, 1992. ACM Press.
246. P. Cousot and R. Cousot. Galois connection based abstract interpretations for strictness analysis, invited paper. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. FMPA*, Akademgorodok, Novosibirsk, RU, LNCS 735, pages 98–127. Springer-Verlag, 28 June – 2 Jul. 1993.
247. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In *Proc. 1994 ICCL*, pages 95–112, Toulouse, FR, 16–19 May 1994. IEEE Comp. Soc. Press.
248. P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. 7^{th} FPCA*, pages 170–181, La Jolla, CA, 25–28 June 1995. ACM Press.
249. P. Cousot and R. Cousot. Parallel combination of abstract interpretation and model-based automatic analysis of software. In R. Cleaveland and D. Jackson, editors, *Proc. 1^{st} ACM SIGPLAN Workshop on Automatic Analysis of Software, AAS '97*, pages 91–98, Paris, FR, Jan. 1997. ACM Press.
250. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Aut . Soft . Eng.*, 6:69–95, 1999.
251. P. Cousot and R. Cousot. Abstract interpretation based program testing, invited paper. In *Proc. SSGRR 2000 Computer & eBusiness International Conference*, Compact disk paper 248, L'Aquila, IT, 31 Jul. – 6 Aug. 2000. Scuola Superiore G. Reiss Romoli.

252. P. Cousot and R. Cousot. Temporal abstract interpretation. In *27th POPL*, pages 12–25, Boston, MA, Jan. 2000. ACM Press.
253. P. Cousot, R. Cousot, A. Deutsch, C. Ferdinand, É. Goubault, N. Jones, D. Pilaud, F. Randimbivololona, M. Sagiv, H. Seidel, and R. Wilhelm. DAEDALUS: Validation of critical software by static analysis and abstract testing. Project IST-1999-20527 of the european 5th Framework Programme (FP5), Oct. 2000 – Oct. 2002.
254. P. Cousot, R. Cousot, and M. Riguidel. TUAMOTU: Tatouage électronique sémantique de code mobile Java. Project RNRT 1999 n° 95, Oct. 1999 – Oct. 2001.
255. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.
256. P. Crégut. Interprétation abstraite pour améliorer la représentation des environnements dans les langages fonctionnels. *Actes JTASPEFL '91, Bordeaux, FR. BIGRE*, 74:37–43, Oct. 1991.
257. B. Creusillet and F. Irigoin. Interprocedural array region analyses. In C.-H. Huang, P. Sadayappan, U. Banerjee, D. Gelernter, A. Nicolau, and D.A. Padua, editors, *Proc. 8th Int. Work. LCPC '95*, Columbus, OH, US, 10–12 Aug. 1995, LNCS 1033, pages 46–60. Springer-Verlag, 1996.
258. R. Cridlig. Semantic analysis of shared-memory concurrent languages using abstract model-checking. In *Proc. PEPM '95*, La Jolla, CA, 21–23 June 1995. ACM Press.
259. R. Cridlig. Semantic analysis of Concurrent ML by abstract model-checking. *ENTCS*, 5, 1996.
260. R. Cridlig. Semantic analysis of Concurrent ML by abstract model-checking. In B. Steffen and T. Margaria, editors, *Proc. Int. Work. on Verification of Infinite State Systems*. vol. MIP-9614, Universit t Passau, DE, Aug. 1996. To be published in Electronic Notes on Theoretical Computer Science, 1997.
261. R. Cridlig. Implementing a static analyzer of concurrent programs: Problems and perspectives. In M. Dam, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, Stockhlom, SE, 24–26 June 1996, LNCS 1192, pages 244–259. Springer-Verlag, 1997.
262. R. Cridlig and É. Goubault. Semantics and analysis of Linda-based languages. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 72–86. Springer-Verlag, 22–24 Sep. 1993.
263. D. Cyrluk. Inverting the abstraction mapping: A methodology for hardware verification. In M.S. Srivas and A.J. Camilleri, editors, *Proc. 1st Int. Conf. on Formal Methods in Computer-Aided Design, FMCAD '96*, number 1166 in LNCS, pages 172–186, Palo Alto, CA, US, 6–8 Nov. 1996. Springer-Verlag.
264. M. Dam. Fixed points of b chi automata. In R.K. Shyamasundar, editor, *Proc. 12th FST & TCS*, New Delhi, IN, 18–20 Dec. 1992, LNCS 652, pages 39–50. Springer-Verlag, 1992.
265. L. Damas and R. Milner. Principal type-schemes for functional programs. In *9th POPL*, pages 207–212, Albuquerque, NM, Jan. 1982. ACM Press.
266. D. Dams, R. Gerth, G. D hmen, R. Herrmann, P. Kelb, and H. Pargmann. Model checking using adaptive state and data abstraction. In D.L. Dill, editor, *Proc. 6th Int. Conf. CAV '94*, Stanford, CA, US, LNCS 818, pages 455–467. Springer-Verlag, 21–23 June 1994.

267. D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ∀CTL*, ∃CTL* and CTL*. In E.R. Olderog, editor, *Proc. IFIP WG2.1/WG2.2/WG2.3 Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET)*, IFIP Transactions. North-Holland/Elsevier, June 1994.
268. D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems. *TOPLAS*, 19(2):253–291, Mar. 1997.
269. S. Das, D.L. Dill, and S. Park. Experience with predicate abstraction. In N. Halbwachs and D. Peled, editors, *Proc. 11th Int. Conf. CAV '99*, Trento, IT, LNCS 1633, pages 160–171. Springer-Verlag, 6–10 Jul. 1999.
270. K. Davis. Higher order binding time analysis. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 80–87. ACM Press, 1993.
271. K. Davis and P. Wadler. Backwards strictness analysis: Proved and improved. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow 1989*, Proc. 1989 Glasgow Workshop, Fraserburgh, UK. Springer-Verlag and BCS, 12–30 Aug. 1989.
272. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III, Verification and Control*, LNCS 1066, pages 208–219. Springer-Verlag, 1996.
273. M.J. Garc a de la Banda and M.V. Hermenegildo. A practical application of sharing and freeness inference. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 118–125. IRISA, Rennes, FR, 23–25 Sep. 1992.
274. G.R. Renardel de Lavalette. Strictness analysis via abstract interpretation for recursively defined types. *Inform. and Comput.*, 99(2):154–177, Aug. 1992.
275. S.K. Debray. Formal bases for dataflow analysis of logic programs. In G. Levi, editor, *Advances in Logic Programming Theory*, Int. Schools for Computer Scientists, section 3, pages 115–182. Clarendon Press, 1994.
276. S.K. Debray, P. López-Garc a, M.V. Hermenegildo, and N.-W. Lin. Lower bound cost estimation for logic programs. In J. Małuszyński, editor, *Proc. Int. Symp. ILPS '1997*, Port Jefferson, Long Island, NY, US, pages 291–305. MIT Press, 13–16 Oct. 1997.
277. S.K. Debray, R. Muth, and M. Weippert. Alias analysis of executable code. In *25th POPL*, pages 12–24, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
278. S.K. Debray and T.A. Proebsting. Interprocedural control flow analysis of first-order programs with tail-call optimization. *TOPLAS*, 19(4):568–585, Jul. 1997.
279. S.K. Debray and D.S. Warren. Automatic mode inferencing for Prolog programs. In *Proc. 1986 Int. Symp. on Logic Programming*, Salt Lake City, UT, pages 78–88. IEEE Comp. Soc. Press, Sep. 1986.
280. S.K. Debray and D.S. Warren. Automatic mode inference of logic programs. *J. Logic Programming*, 5(3):207–229, 1988.
281. S. Decorte and D. De Schrey. Termination analysis: Some practical properties of the norm and level mapping space. In J. Jaffar, editor, *JICSLP '98, Workshop on Concurrent and Parallel Implementations*, Manchester, UK, pages 235–249. MIT Press, 15–19 June 1992.
282. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based termination analysis of logic programs. *TOPLAS*, 21(6):1137–1195, Nov. 1999.
283. G. DeFouw, D. Grove, and C. Chambers. Fast interprocedural class analysis. In *25th POPL*, pages 222–236, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.

284. G. Delzanno and A. Podelski. Model checking in CLP. In W.R. Cleaveland, editor, *Proc. 5^{th} Int. Conf. TACAS '99*, Amsterdam, NL, 22-25 Mar. 1999, LNCS 1579, pages 223–239. Springer-Verlag, 1999.
285. N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Automatic termination analysis of programs containing arithmetic predicates. *ENTCS*, 30(2), 1999.
286. A. Deutsch. On determining lifetime and aliasing of dynamically allocated data in higher-order functional specifications. In *17^{th} POPL*, pages 157–168, San Fransisco, CA, Jan. 1990. ACM Press.
287. A. Deutsch. A storeless model of aliasing and its abstraction using finite representations of right-regular equivalence relations. In *Proc. 1992 ICCL*, Oakland, CA, pages 2–13. IEEE Comp. Soc. Press, 20–23 Apr. 1992.
288. A. Deutsch. Interprocedural may-alias analysis for pointers: Beyond k-limiting. In *Proc. ACM SIGPLAN '94 Conf. PLDI. ACM SIGPLAN Not. 29(6)*, pages 230–241, Orlando, FL, US, 20–24 June 1994. ACM Press.
289. A. Deutsch. Semantic models and abstract interpretation techniques for inductive data structures and pointers, invited paper. In *Proc. PEPM '95*, pages 226–229, La Jolla, CA, 21–23 June 1995. ACM Press.
290. A. Deutsch, G. Gonthier, and M. Turin. La vérification des programmes d'ariane. *Pour la Science*, 243:21–22, Jan. 1998. (in French).
291. D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer-Verlag, 1989.
292. D.L. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In P. Wolper, editor, *Proc. 7^{th} Int. Conf. CAV '95*, Li ge, BE, LNCS 939, pages 409–422. Springer-Verlag, 3–5 Jul. 1995.
293. J. Dingel and T. Filkorn. Checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In P. Wolper, editor, *Proc. 7^{th} Int. Conf. CAV '95*, Li ge, BE, LNCS 939, pages 54–69. Springer-Verlag, 3–5 Jul. 1995.
294. N. Dor, M. Rodeh, and M. Sagiv. Checking cleanness in linked lists. In J. Palsberg, editor, *Proc. 7^{th} Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 115–134. Springer-Verlag, 29 June – 1 Jul. 2000.
295. V. Dornic, P. Jouvelot, and D.K. Gifford. Polymorphic time systems for estimating program complexity. *Actes JTASPEFL '91, Bordeaux, FR. BIGRE*, 74:9–17, Oct. 1991.
296. V. Dumortier, G. Janssens, M. Bruynooghe, and M. Codish. Freeness analysis in the presence of numerical constraints. In D.S. Warren, editor, *Proc. 10^{th} ICLP '93*, Budapest, HU, pages 100–115. MITpress, 21–25 June 1993.
297. P. Dybjer. Inverse image analysis. In T. Ottmann, editor, *14^{th} ICALP*, Karlsruhe, DE, LNCS 267, pages 21–30. Springer-Verlag, 13–17 Jul. 1987.
298. P. Dybjer. Inverse image analysis generalises strictness analysis. *Inform. and Comput.*, 90:194–216, 1991.
299. P.G. Emelianov. Analysis of the equality relations for the program terms. In R. Cousot and D.A. Schmidt, editors, *Proc. 3^{rd} Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 174–188. Springer-Verlag, 1996.
300. P.G. Emelianov and D.E. Baburin. Semantic analyzer of Modula-programs. In P. van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 361–363. Springer-Verlag, 1997.

301. P.G. Emelianov and V.K. Sabelfeld. Analyzer of semantic properties of Modula-programms. In *Software intellectualization and quality*, pages 100–107. Institute of Informatics Systems, Novosibirsk, RU, 1994. In russian.
302. E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" revisited: On branching time versus linear time. *TOPLAS*, 33:151–178, 1986.
303. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, editor, *Proc. 5th Int. Conf. CAV '93*, Elounda, GR, LNCS 697, pages 385–396. Springer-Verlag, 28 June –1 Jul. 1993.
304. V. Englebert, B. Le Charlier, D. Roland, and P. Van Hentenryck. Generic abstract interpretation algorithms for Prolog: Two optimizations techniques and their experimental evaluation. Tech. rep. CS-91-67, Department of Computer Science, Brown University, Providence, RI, Oct. 1991.
305. V. Englebert, B. Le Charlier, D. Roland, and P. Van Hentenryck. Generic abstract interpretation algorithms for Prolog: Two optimization techniques and their experimental evaluation. *Soft.–Pract. & Exp.*, 23(4):419–459, 1993.
306. C. Ernoult and A. Mycroft. Uniform ideals and strictness analysis. In J. Leach Albert, B. Monien, and M. Rodr guez Artalejo, editors, *18th ICALP*, LNCS 510, pages 47–59. Springer-Verlag, Jul. 1991.
307. R. Evertsz. The generation of 'critical problems' by abstract interpretations of student models. In N.S. Sridharan, editor, *Proc. 11th IJCAI '89*, pages 483–488, Detroit, MI, US, Aug. 1989. Morgan Kaufmann Pub.
308. M. F hndrich and A. Aiken. Program analysis using mixed term and set constraints. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 114–126. Springer-Verlag, 1997.
309. M. Falaschi, P. Hicks, and W.H. Winsboroug. Demand transformation analysis for concurrent constraint programs. In M.J. Maher, editor, *Proc. JICSLP '96*, Bonn, DE, pages 333–347. MIT Press, 2–6 Sep. 1996.
310. A. Fantechi, S. Gnesi, and D. Latella. Diego: Towards automatic temporal logic verification of value passing process algebra using abstract interpretation. In U. Montanari and V. Sassone, editors, *Proc. 7th Int. Conf. CONCUR '96*, number 1119 in LNCS, pages 563–578. Springer-Verlag, Pisa, IT, 26–29 Aug. 1996.
311. K.-P. Faxén. Optimizing lazy functional programs using flow inference. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 136–153. Springer-Verlag, 1995.
312. C. Fecht. GENA – a tool for generating Prolog analyzers from specifications. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 418–419. Springer-Verlag, 1995.
313. C. Fecht. An efficient and precise sharing domain for logic programs. In H. Kuchen and D.S. Swierstra, editors, *Proc. 8th Int. Symp. PLILP '96*, pages 469–470. Springer-Verlag, Aachen, DE, 24–27 Sep. 1996, LNCS 1140, 1996.
314. C. Fecht and H. Seidl. A faster solver for general systems of equations. *Sci. Comput. Programming, Special Issue on SAS'96*, 35(1):137–161, September 1999.
315. C. Ferdinand, F. Martin, R. Wilhelm, and M. Alt. Cache behavior prediction by abstract interpretation. *Sci. Comput. Programming, Special Issue on SAS'96*, 35(1):163–189, September 1999.
316. C. Ferdinand, F. Martin, R. Wilhelm, and M. Alt. Cache behavior prediction by abstract interpretation. *Sci. Comput. Programming*, 35(1):163–189, 1999.
317. C. Ferdinand and R. Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Syst.*, 17(2–3):131–181, 1999.

318. J. Feret. Occurrence counting analysis for the π-calculus. In P. Cousot, É. Goubault, J. Gunawardena, M. Herlihy, M. Raussen, and V. Sassone, editors, *Preliminary Proc. Workshop GETCO '00*, pages 99–116, State College, US, 21 Aug. 2000. BRICS Notes Series NS-00-3.
319. J. Feret. Confidentiality analysis of mobile systems. In J. Palsberg, editor, *Proc. 7^{th} Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 135–154. Springer-Verlag, 29 June – 1 Jul. 2000.
320. A. Ferguson and J. Hughes. Fast abstract interpretation using sequential algorithms. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3^{rd} Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 45–59. Springer-Verlag, 22–24 Sep. 1993.
321. J.-C. Fernandez. Abstract interpretation and verification of reactive systems. In P. Cousot, P. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3^{rd} Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 60–71. Springer-Verlag, 22–24 Sep. 1993.
322. J. Field, J. Heering, and T.B. Dinesh. Equations as a uniform framework for partial evaluation and abstract interpretation, 1. *Electronic Symposium on Partial Evaluation Principles, Foundations and Frameworks, ACM Comput. Surv.*, 30(3es), Sep. 1998.
323. G. Fil , R. Giacobazzi, and F. Ranzato. A unifying view on abstract domain design. *ACM Computing Surveys*, 28(2):333–336, 1996.
324. G. Filé and F. Ranzato. Improving abstract interpretations by systematic lifting to the powerset. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 655–669. MIT Press, 13–17 Nov. 1994.
325. G. Filé and F. Ranzato. Complementation of abstract domains made easy. In M.J. Maher, editor, *Proc. JICSLP '96*, Bonn, DE, pages 348–362. MIT Press, 2–6 Sep. 1996.
326. G. Filé and F. Ranzato. The powerset operator on abstract interpretations. *Theoret. Comput. Sci.*, 222(1-2):77–111, Jul. 1999.
327. G. Filé and S. Rossi. Static analysis of Prolog with cut. In A. Voronkov, editor, *Proc. 4^{th} Int. Conf. LPAR '93*, pages 134–145, St. Petersburg, RU, LNCS 698, 13–20 Jul. 1993. Springer-Verlag.
328. G. Filé and P. Sottero. Abstract interpretation for type checking. In J. Małuszyński and M. Wirsing, editors, *Proc. 3^{rd} Int. Symp. PLILP '91*, pages 311–322, Passau, DE, LNCS 528, 26–28 Aug. 1991. Springer-Verlag.
329. C. Flanagan and M. Felleisen. Componential set-based analysis. *TOPLAS*, 21(2):370–416, Feb. 1999.
330. C. Flanagan, M. Flatt S. Krishnamurthi, S. Weirich, and M. Felleisen. Static debugging: Browsing the web of program invariants. In *Proc. ACM SIGPLAN '96 Conf. PLDI. ACM SIGPLAN Not. 31(5)*, pages 23–32, Philadelphia, PA, US, 21–24, May 1996.
331. R.W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Proc. Symposium in Applied Mathematics*, volume 19, pages 19–32. AMS, 1967.
332. I.T. Foster and W.H. Winsborough. Copy avoidance through compile-time analysis and local reuse. In K. Ueda V.A. Saraswat, editor, *Proc. 1991 Int. Symp. ISLP '91*, San Diego, CA, US, pages 455–469. MIT Press, 28 Oct. – 1 Nov. 1997.
333. M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. 9^{th} ACM Symp. on App. Comp.*, Phoenix, AZ, Mar. 1994. ACM Press.
334. J. Gallagher, M. Codish, and E. Shapiro. Specialization of Prolog and FCP programs by abstract interpretation. *New Gen. Comp.*, 6:159–186, 1988.

335. M. Gengler and M. Rytz. A polyvariant binding time analysis handling partially known values. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 322–330. IRISA, Rennes, FR, 23–25 Sep. 1992.
336. R. Ghiya and L. Hendren. Is it a tree, a dag, or a cyclic graph? a shape analysis for heap-directed pointers in C. In *23^{rd} POPL*, pages 1–15, St. Petersburg Beach, FL, 1996. ACM Press.
337. R. Ghiya and L.J. Hendren. Putting pointer analysis to work. In *25^{th} POPL*, pages 121–133, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
338. R. Giacobazzi. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In C. Puech and R. Reischuk, editors, *Proc. Annual Symp. STACS '96*, LNCS 1046, pages 503–514. Springer-Verlag, 1996.
339. R. Giacobazzi. A tutorial on domain theory in abstract interpretation. In G. Levi, editor, *Proc. 5^{th} Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 349–350. Springer-Verlag, 1998.
340. R. Giacobazzi, S. Debray, and G. Levi. Joining abstract and concrete computations in constraint logic programmimg. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Proc. 3^{rd} Int. Conf. AMAST '93*, London, GB, Workshops in Comp., pages 109–126. Springer-Verlag, 1993.
341. R. Giacobazzi, S.K. Debray, and G. Levi. Generalized semantics and abstract interpretation for constraint logic programs. *J. Logic Programming*, pages 191–247, 1995.
342. R. Giacobazzi and F. Ranzato. Completeness in abstract interpretation: A domain perspective. In M. Johnson, editor, *Proc. 6^{th} Int. Conf. AMAST '97, Sydney, AU*, volume 1349 of *LNCS*, pages 231–245. Springer-Verlag, 13–18 Dec. 1997.
343. R. Giacobazzi and F. Ranzato. Compositional optimization of disJune ctive abstract interpretations. In H. Riis Nielson, editor, *Proc. 6^{th} ESOP '96*, Link ping, SE, LNCS 1058, pages 141–155. Springer-Verlag, 22–26 Apr. 1996.
344. R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24^{th} Int. Coll. ICALP '97*, volume 1256 of *LNCS*, pages 771–781. Springer-Verlag, 1997.
345. R. Giacobazzi and F. Ranzato. Optimal domains for disJune ctive abstract interpretation. *Sci. Comput. Programming*, 32(1–3):177–210, 1998.
346. R. Giacobazzi and F. Ranzato. The reduced relative power operation on abstract domains. *Theoret. Comput. Sci.*, 216:159–211, 1999.
347. R. Giacobazzi, F. Ranzato, and F. Scozzari. Building complete abstract interpretations in a linear logic-based setting. In G. Levi, editor, *Proc. 5^{th} Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 215–229. Springer-Verlag, 1998.
348. R. Giacobazzi, F. Ranzato, and F. Scozzari. Complete abstract interpretations made constructive. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proc. 23^{rd} Int. Symp. on Mathematical Foundations of Computer Science, MFCS'98*, volume 1450 of *LNCS*, pages 366–377. Springer-Verlag, 1998.
349. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.
350. R. Giacobazzi and L. Ricci. Detecting determinate computations by bottom-up abstract interpretation. In B. Krieg-Br ckner, editor, *Proc. 4^{th} ESOP '92*, Rennes, FR, LNCS 582, pages 167–181. Springer-Verlag, 26–28 Feb. 1992.
351. F. Giannotti and D. Latella. Using abstract interpretation for gate splitting in LOTOS specifications. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu,

and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 194–204. IRISA, Rennes, FR, 23–25 Sep. 1992.
352. F. Giannotti and D. Latella. Gate splitting in LOTOS specifications using abstract interpretation. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Int. J. Conf. TAPSOFT '93*, Orsay, FR, Volume 2 CAAP/FASE), LNCS 668, pages 437–452. Springer-Verlag, 13–17 Apr. 1993.
353. F. Giannotti and D. Latella. Gate splitting in LOTOS specifications using abstract interpretation. *Sci. Comput. Programming*, 23((2-3)):127–149, 1994.
354. J.-L. Giavitto, J.-P. Sansonnet, and O. Michel. Inférer rapidement la géométrie des collections. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 185–193. IRISA, Rennes, FR, 23–25 Sep. 1992.
355. J. Giesl. Termination analysis for functional programs using term orderings. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 154–171. Springer-Verlag, 1995.
356. F. Giunchiglia. Using Abstrips abstractions – where do we stand? *Art. Int. Rev.*, Jan. 1997.
357. F. Giunchiglia, R. Sebastiani, A. Villafiorita, and T. Walsh. A general purpose reasoner for abstraction. In G. McCalla, editor, *Advances in Artificial Intelligence, Proc. 11^{th} Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI '96*, Toronto, CA, LNCS 1081, pages 323–335. Springer-Verlag, 21–24 May 1996.
358. F. Giunchiglia and A. Villafiorita. ABSFOL: a proof checker with abstraction. In M.A. McRobbie and J.K. Slaney, editors, *Proc. 30^{th} Int. Conf. CADE '96*, volume 1104 of *New Brunswick, NJ, US, LNAI*, pages 136–140. Springer-Verlag, Jul. 30–Aug. 3 1996.
359. F. Giunchiglia and T. Walsh. Abstract theorem proving. In N.S. Sridharan, editor, *Proc. 11^{th} IJCAI '89*, pages 372–377, Detroit, MI, US, Aug. 1989. Morgan Kaufmann Pub.
360. F. Giunchiglia and T. Walsh. Using abstraction. In L. Steels and B. Smith, editors, *Proc. 8^{th} Conf. AISB '91*, pages 225–234, Leeds, GB, 1991. Springer-Verlag.
361. F. Giunchiglia and T. Walsh. A theory of abstraction. *Art. Int.*, 56(2–3):323–390, Oct. 1992.
362. F. Giunchiglia and T. Walsh. Tree subsumption: Reasoning with outlines. In B. Neumann, editor, *Proc. 10^{th} ECAI '92*, pages 77–81, Vienna, AT, Aug. 1992. Wiley & S.
363. F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. *J. Autom. Reason.*, 11(1):23–41, Aug. 1993.
364. PA. Godefroid. VeriSoft: A tool for the automatic analysis of concurrent reactive software. In O. Grumberg, editor, *Proc. 9^{th} Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 476–479. Springer-Verlag, 22–25 Jul. 1997.
365. É. Goubault. Schedulers as abstract interpretations of higher-dimensional automata. In *Proc. PEPM '95*, La Jolla, CA, pages 134–145. ACM Press, 21–23 June 1995.
366. É. Goubault and C. Hankin. A lattice for the abstract interpretation of term graph rewriting systems. In R. Sleep, R. Plasmeijer, and van M. Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 10, pages 131–140. Wiley & S., 1993.
367. É. Goubault, C. Hankin, M. van Eekelen, and E. Nocker. Abstract reduction: towards a theory via abstract interpretation. In R. Sleep, R. Plasmeijer, and

van M. Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 9, pages 117–129. Wiley & S., 1993.

368. J. Goubault. Generalized boxings, congruences and partial inlining. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 147–161. Springer-Verlag, 1994.
369. S. Graf. Verification of a distributed cache memory by using abstractions. In D.L. Dill, editor, *Proc. 6th Int. Conf. CAV '94*, Stanford, CA, US, LNCS 818, pages 207–219. Springer-Verlag, 21–23 June 1994.
370. S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In C. Courcoubetis, editor, *Proc. 5th Int. Conf. CAV '93*, Elounda, GR, LNCS 697, pages 71–84. Springer-Verlag, 28 June –1 Jul. 1993.
371. S. Graf and H. Sa di. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proc. 9th Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 72–83. Springer-Verlag, 22–25 Jul. 1997.
372. P. Granger. Static analysis of arithmetical congruences. *Int. J. Comput. Math.*, 30:165–190, 1989.
373. P. Granger. *Analyses s mantiques de congruence*. Th se de l'école polytechnique en informatique, LIX, École polytechnique, Palaiseau, FR, 12 Jul. 1991.
374. P. Granger. Static analysis of linear congruence equalities among variables of a program. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. Int. J. Conf. TAPSOFT '91, Volume 1 (CAAP '91)*, Brighton, GB, LNCS 493, pages 169–192. Springer-Verlag, 1991.
375. P. Granger. Static analyses of congruence properties on rational numbers. In P. van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 278–292. Springer-Verlag, 1997.
376. D. Cabeza Gras and M.V. Hermenegildo. Extracting non-strict independent and-parallelism using sharing and freeness information. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 297–313. Springer-Verlag, 1994.
377. I. Greif and A.R. Meyer. Specifying the semantics of **while** programs: A tutorial and critique of a paper by hoare and lauer. *TOPLAS*, 3(4):484–507, Oct. 1981.
378. C.A. Gunter, E.L. Gunter, and D.B. MacQueen. Computing ML equality kinds using abstract interpretation. *Inform. and Comput.*, 107(2):303–323, Dec. 1993.
379. M. Hagiya and A. Tozawa. On a new method for dataflow analysis of Java Virtual Machine subroutines. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 17–32. Springer-Verlag, 1998.
380. N. Halbwachs. *D termination automatique de relations lin aires v rifi es par les variables d'un programme*. Th se de 3ème cycle d'informatique, Université scientifique et médicale de Grenoble, Grenoble, FR, 12 Mar. 1979.
381. N. Halbwachs. Delays analysis in synchronous programs. In C. Courcoubatis, editor, *Proc. 5th Int. Conf. CAV '93*, Elounda, GR, LNCS 697, pages 333–346. Springer-Verlag, 28 June –1 Jul. 1993.
382. N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Acad. Pub., 1993.
383. N. Halbwachs. About synchronous programming and abstract interpretation. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 179–192. Springer-Verlag, 1994.
384. N. Halbwachs. About synchronous programming and abstract interpretation. *Sci. Comput. Programming*, 31(1):75–89, May 1998.
385. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proc. of the IEEE*, 79(9):1305–1320, Sep. 1991.

386. N. Halbwachs, J.-É. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 223–237. Springer-Verlag, 1994.
387. N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, Aug. 1997.
388. M.W. Hall, B.R. Murphy, S.P. Amarasinghe, S.-W. Liao, and M.S. Lam. Interprocedural analysis for parallelization. In C.-H. Huang, P. Sadayappan, U. Banerjee, D. Gelernter, A. Nicolau, and D.A. Padua, editors, *Proc. 8th Int. Work. LCPC '95*, Columbus, OH, US, 10–12 Aug. 1995, LNCS 1033, pages 61–80. Springer-Verlag, 1996.
389. G.W. Hamilton. Sharing analysis of lazy first-order functional programs. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 68–78. IRISA, Rennes, FR, 23–25 Sep. 1992.
390. M. Handjieva. STAN: A static analyzer for CLP($\Re$) based on abstract interpretation. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 383–384. Springer-Verlag, 1996.
391. M. Handjieva and S. Tzolovski. Refining static analyses by trace-based partitioning using control flow. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 200–214. Springer-Verlag, 1998.
392. C. Hankin and S. Hunt. Fixed points and frontiers: A new perspective. *JFP*, 1(1):91–120, 1991.
393. C. Hankin and S. Hunt. Approximate fixed points in abstract interpretation. In B. Krieg-Br ckner, editor, *Proc. 4th ESOP '92*, Rennes, FR, LNCS 582, pages 219–232. Springer-Verlag, 26–28 Feb. 1992.
394. C. Hankin and S. Hunt. Approximate fixed points in abstract interpretation. *Sci. Comput. Programming*, 22(3):283–306, 1994. Erratum: Sci. Comput. Programming 23(1): 103 (1994).
395. C. Hankin and D. Le Métayer. A type-based framework for program analysis. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 380–394. Springer-Verlag, 1994.
396. C. Hankin, F. Nielson, and H. Riis Nielson. *Principles of Program Analysis*. Springer-Verlag, 1998.
397. J. Hannan. A type-based analysis for stack allocation in functional languages. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 172–188. Springer-Verlag, 1995.
398. J. Hannan. Program analysis in Lambda-Prolog. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proc. 10th Int. Symp. PLILP '98*, pages 353–354. Springer-Verlag, Pisa, IT, 16–18 Sep. 1998, LNCS 1490, 1998.
399. R.R. Hansen, J.G. Jensen, F. Nielson, and H. Riis Nielson. Abstract interpretation of mobile ambients. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 134–138. Springer-Verlag, 1999.
400. M. Hanus. Analysis of nonlinear constraints in CLP(R). In D.S. Warren, editor, *Proc. 10th ICLP '93*, Budapest, HU, pages 83–99. MITpress, 21–25 June 1993.
401. M. Hanus. Towards the global optimization of functional logic programs. In P.A. Fritzson, editor, *Proc. 5th Int. Conf. CC '94*, Edinburg, UK, LNCS 786, pages 68–82. Springer-Verlag, Apr. 1994.

402. M. Hanus. Compile-time analysis of nonlinear constraints in CLP(R). *New Gen. Comp.*, 13(2):155–186, 1995.
403. M. Hanus and S. Lucas. A semantics for program analysis in narrowing-based functional logic languages. In A. Middeldorp and T. Sato, editors, *4th FLOPS '99*, Tsukuba, JP, 11–13 Nov. 1999, LNCS 1722, pages 353–368. Springer-Verlag, 1999.
404. M. Hanus and F. Zartmann. Mode analysis of functional logic programs. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 26–42. Springer-Verlag, 1994.
405. R. Harper. A simplified account of polymorphic references. *Inf. Process. Lett.*, 54(4):201–206, 1994.
406. R. Harper. A note on "a simplified account of polymorphic references". *Inf. Process. Lett.*, 57(1):15–16, 1996.
407. R. Harper, R. Milner, and M. Tofte. A type discipline for program modules. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proc. Int. J. Conf. TAPSOFT '87, Volume 2 (AFISD/CFLP)*, Pisa, IT, LNCS 250, pages 308–319. Springer-Verlag, 23–27 Mar. 1987.
408. R. Harper and J.C. Mitchell. On the type structure of Standard ML. *TOPLAS*, 15(2):211–252, 1993.
409. R. Harper and J.C. Mitchell. ML and beyond. *ACM SIGPLAN Not.*, 32(1):8085, 1997.
410. W.L. Harrison. Can abstract interpretation become a main stream compiler technology? (abstract). In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, page 395. Springer-Verlag, 1997.
411. V. Hartonas-Garmhausen, A. Campos, S.V.A. Cimatti, E.M. Clarke, and F. Giunchiglia. Verification of a safety-critical railway interlocking system with real-time constraints. In *Proc. FTCS '28*, Munchen, DE, pages 458–463. IEEE Comp. Soc. Press, 23–25 June 1998.
412. V. Hartonas-Garmhausen, S.V.A. Campos, and E.M. Clarke. ProbVerus: Probabilistic symbolic model checking. In J.-P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th Int. Symp. AMAST Workshop, ARTS '99*, Bamberg, DE, 26–28 May 1999, LNCS 1601, pages 96–110. Springer-Verlag, 1993.
413. J. Hatcliff, M.B. Dwyer, and S. Laubach. Staging static analyses using abstraction-based program specialization. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proc. 10th Int. Symp. PLILP '98*, pages 134–151. Springer-Verlag, Pisa, IT, 16–18 Sep. 1998, LNCS 1490, 1998.
414. K. Havelund, K.G. Larsen, and A. Skou. Formal verification of an audio/video power controller using the real-time model checker UPPAAL. In *ARTS'99*, 1999.
415. K. Havelund, A. Skou, K.G. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *RTSS'97*, 1997.
416. A. Heaton, M. Abo-Zaed, M. Codish, and A. King. Simple, efficient and scalable groundness analysis of logic programs. *J. Logic Programming*, 45(1–3):143–156, 2000.
417. N. Heintze. Practical aspects of set based analysis. In K.R. Apt, editor, *Proc. JICSLP '92*, Washington, DC, US, pages 765–779. MIT Press, Nov. 1992.
418. N. Heintze. *Set Based Program Analysis*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, US, Oct. 1992.
419. N. Heintze. Set-based analysis of ML programs. In *Proc. ACM Conf. Lisp & Func. Prog.*, Orlando, FL, US, pages 306–317. ACM Press, 27–29 June 1994.

420. N. Heintze. Control-flow analysis and type systems. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 189–206. Springer-Verlag, 1995.
421. F. Henglein. Efficient type inference for higher-order binding-time analysis. In J. Hughes, editor, *Proc. 5th FPCA*, LNCS 523, pages 448–472. Springer-Verlag, Aug. 1991.
422. F. Henglein. Iterative fixed point computation for type-based strictness analysis. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 395–407. Springer-Verlag, 1994.
423. F. Henglein and D. Sands. A semantic model of binding times for safe partial evaluation. In M.V. Hermenegildo and S.D. Swierstra, editors, *Proc. 7th Int. Symp. PLILP '95*, Utrecht, NL, 20–22, Sep. 1995, LNCS 982, pages 299–320. Springer-Verlag, 1995.
424. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 5th LICS '92*. IEEE Comp. Soc. Press, June 1992.
425. T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proc. 7th Int. Conf. CAV '95*, Li ge, BE,LNCS 939, pages 225–238. Springer-Verlag, 3–5 Jul. 1995.
426. T.A. Henzinger, R. Majumbar, F. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 220–239. Springer-Verlag, 29 June – 1 Jul. 2000.
427. T.A. Henzinger and R. Majumdar. Symbolic model checking for rectangular hybrid systems. In S. Graf and M.I. Schwartzbach, editors, *Proc. 6th Int. Conf. TACAS '2000*, Berlin, DE, 25 Mar. – 2 Apr. 2000, LNCS 1785, pages 142–156. Springer-Verlag, 2000.
428. P.M. Hill, R. Bagnara, and E. Zaffanella. The correctness of set-sharing. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 99–114. Springer-Verlag, 1998.
429. P.M. Hill and F. Spoto. Freeness analysis through refinement. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 85–100. Springer-Verlag, 1999.
430. M. Hind and A. Pioli. Assessing the effects of flow-sensitivity on pointer alias analyses. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 57–81. Springer-Verlag, 1998.
431. R. Hindley. The principal type-scheme of an object in combinatory logic. *Trans. Amer. Math. Soc.*, 146:29–60, 1969.
432. P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proc. 7th Int. Conf. CAV '95*, Li ge, BE,LNCS 939, pages 381–394. Springer-Verlag, 3–5 Jul. 1995.
433. C.A.R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580, Oct. 1969.
434. C.A.R. Hoare and P.E. Lauer. Consistent and complementary formal theories of the semantics of programming languages. *Acta Informat.*, 3(2):135–153, 1974.
435. K. Horiuchi and T. Kanamori. Polymorphic type inference in Prolog by abstract interpretation. In K. Furukawa, H. Tanaka, and T. Fujisaki, editors, *Proc. 6th Conf. on Logic Programming '87*, Tokyo, JP, LNCS 315, pages 195–214. Springer-Verlag, June 1987.
436. N.R. Horspool and J. Vitek. Static analysis of PostScript code. In *Proc. 1992 ICCL*, Oakland, CA, pages 14–23. IEEE Comp. Soc. Press, 20–23 Apr. 1992.

437. N.R. Horspool and J. Vitek. Static analysis of PostScript code. *Comput. Lang.*, 19(2):65–78, 1993.
438. S. Horwitz. Precise flow-insensitive may-alias analysis is NP-hard. *TOPLAS*, 19(1):1–6, Jan. 1997.
439. F. Huch. Verification of Erlang programs using abstract interpretation and model checking. In *Proc. 4^{th} ACM SIGPLANInt. Conf. ICFP '99, ACM SIGPLAN Not. 34(9)*, pages 261–272, Paris, FR, 27–29 Sep. 1999. ACM Press.
440. P. Hudak and J. Young. Higher-order strictness analysis in untyped lambda calculus. In *12^{th} POPL*, pages 97–109. ACM Press, Jan. 1986.
441. P. Hudak and J. Young. Collecting interpretations of expressions. *TOPLAS*, 13(2):269–290, Apr. 1991.
442. J. Hughes and J. Launchbury. Relational reversal of abstract interpretations. *J. Logic and Comp.*, 2(4):465–509, Aug. 1992.
443. J. Hughes and J. Launchbury. Reversing abstract interpretations. In B. Krieg-Br ckner, editor, *Proc. 4^{th} ESOP '92*, Rennes, FR, LNCS 582, pages 269–286. Springer-Verlag, 26–28 Feb. 1992.
444. J. Hughes and J. Launchbury. Reversing abstract interpretations. *Sci. Comput. Programming*, 22(3):307–326, June 1994.
445. R.J.M. Hughes. Strictness detection in non-flat domains. In H. Ganzinger and N.D. Jones, editors, *Programs as Data Objects, Proceedings of a Workshop*, Copenhagen, DK, 17–19 Oct. 1985, LNCS 217, pages 112–135. Springer-Verlag, 1986.
446. R.J.M. Hughes. Backwards analysis of functional programs. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Partial Evaluation and Mixed Computation*, Proceedings IFIP TC2 Workshop, Gl Avernæs, Ebberup, DK, 18–24 Oct. 1987, pages 187–208. Elsevier, 1988.
447. R.J.M. Hughes. Projections for polymorphic strictness analysis. In D.H. Pitt, D.E. Rydeheard, P. Dybjer, A.M. Pitts, and A. Poigné, editors, *Category Theory and Computer Science*, LNCS 389, pages 82–100. Springer-Verlag, 1989.
448. R.J.M. Hughes and J. Launchbury. Projections for polymorphic first-order strictness analysis. *MSCS*, 2:301–326, 1993.
449. S. Hughes. Compile-time garbage collection for higher-order functional languages. *J. Logic and Comp.*, 2(4):483–464, Aug. 1992.
450. S. Hunt. PERs generalize projections for strictness analysis. Tech. rep. DOC 14/90, Department of Computing, Imperial College, London, GB, Aug. 1990.
451. S. Hunt. PERs generalize projections for strictness analysis. In S.L. Peyton Jones, G. Hutton, and C. Kehler Holst, editors, *Functional Programming, Glasgow 1990*, Proc. 1990 Glasgow Workshop on Functional Programming, Ullapool, UK. Springer-Verlag and BCS, 13–15 Aug. 1990.
452. S. Hunt. Frontiers and open sets in abstract interpretation. In *Proc. 3^{rd} FPCA*, volume 523 of *LNCS*, pages 1–13. ACM Press, Imperial College, London, UK, 11-13 Sep. 1989.
453. S. Hunt and D. Sands. Binding time analysis: A new PERspective. In P. Hudak and N.D. Jones, editors, *Proc. PEPM '91*, Yale U., New Haven, CT, US, 17–19 June 1991, ACM SIGPLAN Not. 26(9), pages 154–165. ACM Press, Sep. 1991.
454. P. Jouvelot J.-P. Talpin. Polymorphic type, region and effect inference. *Actes JTASPEFL '91, Bordeaux, FR. BIGRE*, 74:26–32, Oct. 1991.
455. P. Jouvelot J.-P. Talpin. Polymorphic type, region and effect inference. *J. Func. Prog.*, 2(3):245–271, Jul. 1992.
456. I-P. Lin J. Tan. Recursive modes for precise analysis of logic programs. In J. Małuszyński, editor, *Proc. Int. Symp. ILPS '1997*, Port Jefferson, Long Island, NY, US, pages 277–290. MIT Press, 13–16 Oct. 1997.

457. D. Jackson. Abstract model checking of infinite specifications. In M. Naftalin, T. Denvir, and M. Bertran, editors, *2^{nd} Int. Symp. of Formal Methods Europe FME '94: Industrial Benefit of Formal Methods*, Barcelona, ES, LNCS 873, pages 519–531. Springer-Verlag, Oct. 1994.
458. D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E.L. Lusk and R.A. Overbeek, editors, *NACLP 1989, Volume 1*, Cleaveland, OH, US, pages 154–165. MIT Press, 16–20 Oct. 1989.
459. S. Jagannathan, P. Thiemann, S. Weeks, and A.K. Wright. Single and loving it: Must-alias analysis for higher-order languages. In *25^{th} POPL*, pages 329–341, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
460. S. Jagannathan, S. Weeks, and A.K. Wright. Type-directed flow analysis for typed intermediate languages. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 232–249. Springer-Verlag, 1997.
461. 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, BE, Mar. 1990. (special issue of the Journal of Logic Programming on abstract interpretation).
462. G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *J. Logic Programming*, 13(1,2,3&4):205–258, 1992. published version of [461].
463. G. Janssens, M. Bruynooghe, and V. Dumortier. A blueprint for an abstract machine for abstract interpretation of (constraint) logic programs. In J.W. Lloyd, editor, *Proc. Int. Symp. ILPS '1995*, Portland, OR, US, pages 336–350. MIT Press, 4–7 Dec. 1995.
464. G. Janssens and W. Simoens. On the implementation of abstract interpretation systems for (constraint) logic programming. In P.A. Fritzson, editor, *Proc. 5^{th} Int. Conf. CC '94*, Edinburg, UK, LNCS 786, pages 172–187. Springer-Verlag, Apr. 1994.
465. B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In A. Cortesi and G. Filé, editors, *Proc. 6^{th} Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 18–38. Springer-Verlag, 1999.
466. T.P. Jensen. Strictness analysis in logical form. In J. Hughes, editor, *Proc. 5^{th} FPCA*, LNCS 523, pages 352–366. Springer-Verlag, Aug. 1991.
467. T.P. Jensen. Axiomatising uniform properties of recursive data structures. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2^{nd} Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 144–151. IRISA, Rennes, FR, 23–25 Sep. 1992.
468. T.P. Jensen. Disjunctive strictness analysis. In *Proc. 7^{th} LICS*, pages 174–185. IEEE Comp. Soc. Press, 1992.
469. T.P. Jensen. Clock analysis of synchronous dataflow programs. In *Proc. PEPM '95*, La Jolla, CA, pages 156–167. ACM Press, 21–23 June 1995.
470. T.P. Jensen. Disjunctive program analysis for algebraic data types. *TOPLAS*, 19(5):751–803, Sep. 1997.
471. T.P. Jensen. Inference of polymorphic and conditional strictness properties. In *25^{th} POPL*, pages 209–221, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
472. C. Consel J.M. Ashley. Fixpoint computation for polyvariant static analyses of higher-order applicative programs. *TOPLAS*, 16(5):1331–1448, Sep. 1994.

473. N. Jø rgensen. Chaotic fixpoint iteration guided by dynamic dependency. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 27–44. Springer-Verlag, 22–24 Sep. 1993.
474. T. Johnsson. Detecting when call-by-value can be used instead of call-by-need. Res. rep. LPM MEMO 14, Laboratory for Programming Methodology, Department of Computer Science, Chalmers University of Technology, S-412 96 G teborg, SE, Oct. 1981.
475. N.D. Jones. Abstract interpretation and partial evaluation in functional and logic programming. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, pages 17–22. MIT Press, 13–17 Nov. 1994.
476. N.D. Jones. An introduction to partial evaluation. *ACM Comput. Surv.*, 28(3):480–504, Sep. 1996.
477. N.D. Jones. Combining abstract interpretation and partial evaluation (brief overview). In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 396–405. Springer-Verlag, 1997.
478. N.D. Jones, Gomard C.K., Sestoft P., L.O. (Andersen, and T.) Mogensen. *Partial Evaluation and Automatic Program Generation*. Int. Series in Computer Science. Prentice-Hall, June 1993.
479. N.D. Jones and S.S. Muchnick. Flow analysis and optimization of LISP-like structures. In *6th POPL*, pages 244–256, San Antonio, TX, 1979. ACM Press.
480. 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, 1981.
481. N.D. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs: abridged version. In *30th POPL*, pages 296–306, St. Petersburg Beach, FL, 1986. ACM Press.
482. N.D. Jones and M. Rosendahl. Higher-order minimal function graphs. *J. Func. and Logic Prog.*, 1997(2), 1997.
483. S.B. Jones and D. Le Métayer. A new method for strictness analysis on non-flat domains. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow 1989*, Proc. 1989 Glasgow Workshop, Fraserburgh, UK. Springer-Verlag and BCS, 1–11 Aug. 1989.
484. N. Jørgensen. Finding fixpoints in finite function spaces using neededness analysis and chaotic iteration. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 329–345. Springer-Verlag, 1994.
485. N. Jørgensen, K. Marriott, and S. Michaylov. Some global compile-time optimizations for CLP(R). In K. Ueda V.A. Saraswat, editor, *Proc. 1991 Int. Symp. ISLP '91*, San Diego, CA, US, pages 420–434. MIT Press, 28 Oct. – 1 Nov. 1997.
486. P. Jouvelot. Semantic parallelization: a practical exercise in abstract interpretation. In *14th POPL*, pages 39–48, Munich, DE, 21–23 Jan. 1987. ACM Press.
487. P. Jouvelot and D.K. Gifford. Algebraic reconstruction of types and effects. In *18th POPL*, pages 303–310, Orlando, FL, 1991. ACM Press.
488. M.H. Sørensen K. Nielsen. Call-by-name CPS-translation as a binding-time improvement. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 296–313. Springer-Verlag, 1995.
489. G. Kahn. Natural semantics. In K. Fuchi and M. Nivat, editors, *Programming of Future Generation Computers*, pages 237–258. Elsevier, 1988.
490. S. Kalogeropulos. Identifying the available parallelism using static analysis. In J. Volkert, editor, *Proc. 2nd Int. ACPC Conf. on Parallel Computation*, pages 151–165, Gmunden, AT, Oct. 1993, LNCS 734, 1993. Springer-Verlag.

491. T. Kanamori, K. Horiuchi, and T. Kawamura. Detecting termination of logic programs based on abstract hybrid interpretation. Tech. rep. 398, ICOT, Tokyo, JP, 1987.
492. M.T. Kandemir, P. Banerjee, A.N. Choudhary, J. Ramanujam, and N. Shenoy. A global communication optimization technique based on data-flow analysis and linear algebra. *TOPLAS*, 21(6):1251–1297, Nov. 1999.
493. M. Karr. Affine relationships among variables of a program. *Acta Informat.*, 6:133–151, 1976.
494. P. Kelb. Model checking and abstraction: A framework approximating both truth and failure information. Tech. rep., University of Oldenburg, 1994.
495. R.M. Keller. Formal verification of parallel programs. *Comm. ACM*, 19(7):371–384, Jul. 1977.
496. A.D. Kelly, A.D. Macdonald, K. Marriott, P.J. Stuckey, and R.H.C. Yap. Effectiveness of optimizing compilation for CLP(R). In M.J. Maher, editor, *Proc. JICSLP '96*, Bonn, DE, pages 37–51. MIT Press, 2–6 Sep. 1996.
497. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proc. 9^{th} Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 424–435. Springer-Verlag, 22–25 Jul. 1997.
498. Y. Kesten and A. Pnueli. Modularization and abstraction: The keys to formal verification. In L. Brim, J. Gruska, and J. Zlatuska, editors, *23^{rd} Int. Symp. MFCS '98*, LNCS 1450, pages 54–71. Springer-Verlag, 1998.
499. Y. Kesten and A. Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *STTT*, 2(4):328–342, 2000.
500. G. Kildall. A unified approach to global program optimization. In *1^{st} POPL*, pages 194–206, Boston, MA, Oct. 1973. ACMpress.
501. A. King and P. Sober. Schedule analysis of concurrent logic programs. Tech. rep. CSTR 90-22, Department of Electronics and Computer Science, University of Southampton, Southampton, UK, 1990.
502. A. King and P. Sober. Producer and consumer analysis of concurrent logic programs. Tech. rep. CSTR 91-8, Department of Electronics and Computer Science, University of Southampton, Southampton, GB, 1991.
503. A. King and . Soper. Schedule analysis of concurrent logic programs. In K.R. Apt, editor, *Proc. JICSLP '92*, Washington, DC, US, pages 478–492. MIT Press, Nov. 1992.
504. N. Kobayashi, M. Nakade, and A. Yonezawa. Static analysis of communication for asynchronous concurrent programming languages. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 225–242. Springer-Verlag, 1995.
505. J. K ller and M. Mohnen. A new class of functions for abstract interpretation. In A. Cortesi and G. Filé, editors, *Proc. 6^{th} Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 248–263. Springer-Verlag, 1999.
506. D. Kozen, J. Palsberg, and M.I. Schwartzbach. Efficient inference of partial types. *J. Comput. System Sci.*, 49(2):306–324, 1994.
507. D. Kozen, J. Palsberg, and M.I. Schwartzbach. Efficient recursive subtyping. *MSCS*, 5(1):113–125, 1995.
508. S. Kripke. A semantical analysis of modal logic I: normal modal propositional calculi. *Z. Math. Logik Grundlagen Math.*, 9:67–96, 1963.
509. T.M. Kuo and P. Mishra. On strictness and its analysis. In *14^{th} POPL*, pages 144–155, Munchen, DE, 1987. ACM Press.

510. T.M. Kuo and P. Mishra. Strictness analysis: A new perspective based on type inference. In *Proc. 3rd FPCA*, pages 260–272. ACM Press, Sep. 1989.
511. P. Lacan, J.N. Monfort, L.V.Q. Ribal, A. Deutsch, and G. Gonthier. The software reliability verification process: The ARIANE 5 example. In *Proceedings DASIA 98 – DAta Systems In Aerospace*, Athens, GR. ESA Publications, SP-422, 25–28 May 1998.
512. J. Launchbury. Projections for specialization. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Partial Evaluation and Mixed Computation*, Proceedings IFIP TC2 Workshop, Gl Avernæs, Ebberup, 18–24 Oct. 1987, DK, pages 299–315. Elsevier, 1988.
513. J. Launchbury. Dependent sums express separation of binding times. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow 1989*, Proc. 1989 Glasgow Workshop, Fraserburgh, UK. Springer-Verlag and BCS, 238–253 Aug. 1989.
514. J. Launchbury. *Projection Factorizations in Partial Evaluation*, volume 1 of *Distinguished Dissertations in Computer Science*. Cambridge U. Press, 1991.
515. B. Le Charlier, O. Degimbe, L. Michel, and P. Van Hentenryck. Optimization techniques for general purpose fixpoint algorithms — practical efficiency for abstract interpretation of Prolog. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 15–26. Springer-Verlag, 22–24 Sep. 1993.
516. B. Le Charlier, K. Musumbu, and P. Van Hentenryck. A generic abstract interpretation algorithm and its complexity analysis. In K. Furukawa, editor, *Proc. ICLP '91*, Paris, FR, pages 64–78. MIT Press, 24–28 June 1991.
517. B. Le Charlier and P. Van Hentenryck. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. In *Proc. 1992 ICCL*, Oakland, CA, pages 137–146. IEEE Comp. Soc. Press, 20–23 Apr. 1992.
518. C. Lecoutre, S. Merchez, F. Boussemart, and É. Gr^goire. A CSP abstraction framework. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 164–184. Springer-Verlag, 26–29 Jul. 2000.
519. K.R.M. Leino and G. Nelson. An extended static checker for Modula-3. In K. Koskimies, editor, *Proc. 7th Int. Conf. CC '98*, Lisbon, PT, LNCS 1383, pages 302–305. Springer-Verlag, 28 Mar. – 4 Apr. 1998.
520. X. Leroy and F. Rouaix. Security properties of typed applets. In *25th POPL*, pages 391–403, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
521. M. Leuschel. Program specialisation and abstract interpretation reconciled. In J. Jaffar, editor, *JICSLP '98, Workshop on Concurrent and Parallel Implementations*, Manchester, UK, pages 220–234. MIT Press, 15–19 June 1992.
522. M. Leuschel. On the power of homeomorphic embedding for online termination. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 200–214. Springer-Verlag, 1998.
523. T. Lev-Ami and M. Sagiv. TVLA: A system for implementing static analyses. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 280–301. Springer-Verlag, 29 June – 1 Jul. 2000.
524. G. Levi and F. Spoto. Non pair-sharing and freeness analysis through linear refinement. In *Proc. PEPM '00, ACM SIGPLAN Not. 34(11)*, Boston, MA, US, 22–23 Jan. 2000, pages 202–213. ACM Press, Nov. 1999.
525. G. Levi and P. Volpe. Derivation of proof methods by abstract interpretation. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proc. 10th Int. Symp.*

PLILP '98, pages 102–117. Springer-Verlag, Pisa, IT, 16–18 Sep. 1998, LNCS 1490, 1998.

526. Y.A. Liu and S.D. Stroller. Eliminating dead code on recursive data. In A. Cortesi and G. Filé, editors, *Proc. 6^{th} Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 179–193. Springer-Verlag, 1999.
527. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1), 1995.
528. D.E. Long, A. Browne, E.M. Clarke, S. Jha, and W.R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D.L. Dill, editor, *Proc. 6^{th} Int. Conf. CAV '94*, Stanford, CA, US, LNCS 818, pages 338–350. Springer-Verlag, 21–23 June 1994.
529. D.E. Long, A. Browne, E.M. Clarke, S. Jha, and W.R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoret. Comput. Sci.*, 178(1-2):237–255, 1997.
530. L. Lu. A mode analysis of logic programs by abstract interpretation. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. Perspectives of System Informatics, 2^{nd} Intl. Andrei Ershov Memorial Conf.*, Akademgorodok, Novosibirsk, RU, LNCS 1181, pages 362–373. Springer-Verlag, 25–28 June 1996.
531. I. Mackie. Static analysis of interaction nets for distributed implementations. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 217–231. Springer-Verlag, 1997.
532. F. Malésieux, O. Ridoux, and P. Boizumault. Abstract compilation of Lambda-Prolog. In J. Jaffar, editor, *JICSLP '98*, Manchester, UK, pages 130–144. MIT Press, 15–19 June 1992.
533. Z. Manna, A. Browne, H. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A.M. Haeberer, editor, *Proc. 7^{th} Int. Conf. AMAST '98*, Amazonia, BR, 4–8 Jan. 1999, LNCS 1548, pages 28–41. Springer-Verlag, 1999.
534. E. Marchiori and F. Teusink. Proving termination of logic programs with delay declarations. In J.W. Lloyd, editor, *Proc. Int. Symp. ILPS '1995*, Portland, OR, US, pages 447–461. MIT Press, 4–7 Dec. 1995.
535. T. Margaria and B. Steffen, editors. *A Tool for Proving Invariance Properties of Concurrent Systems Automatically*, Passau, DE, LNCS 1055. Springer-Verlag, 27–29 Mar. 1996.
536. T.J. Marlowe and B.G. Ryder. Properties of data flow frameworks: A unified model. *Acta Informat.*, 28:121–163, 1990.
537. K. Marriott. Frameworks for abstract interpretation. *Acta Informat.*, 30:103–129, 1993.
538. K. Marriott and H. Søndergaard. On propagation-based analysis of logic programs. In S. Michaylov and W. Winsborough, editors, *Proc. Int. Symp. ILPS '1993 Workshop on Global Compilation, Vancouver, CA*, pages 47–65, 1993.
539. K. Marriott and P.J. Stuckey. Approximating interaction between linear arithmetic constraints. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 571–585. MIT Press, 13–17 Nov. 1994.
540. F. Martin. *Generating Program Analyzers.* Pirrot Verlag, Saarbr cken, DE, 1999.
541. F. Masdupuy. Using abstract interpretation to detect array data dependencies. In *Proc. Int. Symp. on Supercomputing*, pages 19–27, Fukuoka, JP, Nov. 1991. Kyushu U. Press.
542. F. Masdupuy. Array operations abstraction using semantic analysis of trapezoid congruences. In *Proc. ACM Int. Conf. on Supercomputing, ICS '92*, pages 226–235, Washington D.C., Jul. 1992.

543. F. Masdupuy. Semantic analysis of interval congruences. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. FMPA*, Akademgorodok, Novosibirsk, RU, LNCS 735, pages 142–155. Springer-Verlag, 28 June – 2 Jul. 1993.
544. L. Mauborgne. Abstract interpretation using TDGs. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 363–379. Springer-Verlag, 1994.
545. L. Mauborgne. Abstract interpretation using typed decision graphs. *Sci. Comput. Programming*, 31(1):91–112, May 1998.
546. L. Mauborgne. Binary decision graphs. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 101–116. Springer-Verlag, 1999.
547. L. Mauborgne. *Representation of Sets of Trees for Abstract Interpretation*. Phd thesis in computer science, École polytechnique, Palaiseau, FR, 25 Nov. 1999.
548. L. Mauborgne. Tree schemata and fair termination. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 302–321. Springer-Verlag, 29 June – 1 Jul. 2000.
549. L. Mauborgne. Improving the representation of infinite trees to deal with sets of trees. In G. Smolka, editor, *Programming Languages and Systems, Proc. 9th ESOP '2000*, Berlin, DE, LNCS 1782, pages 275–289. Springer-Verlag, Mar. – Apr. 2000.
550. H.G. Mayer and M. Wolfe. Interprocedural alias analysis: Implementation and empirical results. *Soft.–Pract. & Exp.*, 23(11):1201–1233, Nov. 1993.
551. K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Acad. Pub., 1993.
552. T.S. McNerney. Verifying the correctness of compiler transformations on basic blocks using abstract interpretation. In P. Hudak and N.D. Jones, editors, *Proc. PEPM '91*, Yale U., New Haven, CT, US, 17–19 June 1991, ACM SIGPLAN Not. 26(9), pages 106–115. ACM Press, Sep. 1991.
553. C.S. Mellish. The automatic generation of mode Dec. laration for Prolog programs. DAI research paper 163, Department of Artificial Intelligence, University of Edinburgh, Edinburg, UK, 1981.
554. C.S. Mellish. Some global optimizations for a Prolog program. *J. Logic Programming*, 2(1):43–66, 1985.
555. C.S. Mellish. Abstract interpretation of Prolog programs. In E. Shapiro, editor, *3rd ICLP '86*, London, GB, LNCS 225, pages 463–474. Springer-Verlag, 14–18 Jul. 1986.
556. C.S. Mellish. Abstract interpretation of Prolog programs. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Dec. larative Languages*, pages 181–198. Ellis Horwood, 1987.
557. N. Mercouroff. An algorithm for analyzing communicating processes. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proc. 7th Int. Conf. on Mathematical Foundations of Programming Semantics*, Pittsburgh, PA, pages 312–325. Springer-Verlag, 25–28 Mar. 1991.
558. S. Michaylov and B. Pippin. Optimizing compilation of linear arithmetic in a class of constraint logic programs. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 586–600. MIT Press, 13–17 Nov. 1994.
559. S.P. Midkiff. Dependence analysis in parallel loops with $i \pm k$ subscripts. In C.-H. Huang, P. Sadayappan, U. Banerjee, D. Gelernter, A. Nicolau, and D.A. Padua, editors, *Proc. 8th Int. Work. LCPC '95*, Columbus, OH, US, 10–12 Aug. 1995, LNCS 1033, pages 331–345. Springer-Verlag, 1996.

560. R. Milner. A theory of polymorphism in programming. *J. Comput. System Sci.*, 17(3):348–375, Dec. 1978.
561. R. Milner and M. Tofte. Co-induction in relational semantics. *Theoret. Comput. Sci.*, 87:209–220, 1991.
562. M. Minasi. *The Software Conspiracy: What You Don't Know About the Software Industry and How It's Taking Control of Your Life*. McGraw-Hill, 1999.
563. J.C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 8, pages 365–458. Elsevier, 1990.
564. T.Æ . Mogensen. A semantics-based determinacy analysis for prolog with cut. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. Perspectives of System Informatics, 2^{nd} Intl. Andrei Ershov Memorial Conf.*, Akademgorodok, Novosibirsk, RU, LNCS 1181, pages 374–385. Springer-Verlag, 25–28 June 1996.
565. M. Mohnen. Efficient closure utilisation by higher-order inheritance analysis. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 261–278. Springer-Verlag, 1995.
566. U. M ncke and R. Wilhelm. Grammar flow analysis. In H. Alblas and B. Melichar, editors, *PROC Attribute Grammars, Applications and Systems, Int. Summer School SAGA*, Prague, CZ, LNCS 545, pages 151–186. Springer-Verlag, 1991.
567. D. Monniaux. Abstracting cryptographic protocols with tree automata. In A. Cortesi and G. Filé, editors, *Proc. 6^{th} Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 149–163. Springer-Verlag, 1999.
568. D. Monniaux. Abstract interpretation of probabilistic semantics. In J. Palsberg, editor, *Proc. 7^{th} Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 322–339. Springer-Verlag, 29 June – 1 Jul. 2000.
569. B. Monsuez. Polymorphic typing by abstract interpretation. In R. Shyamasundar, editor, *Proc. 12^{th} FST & TCS*, pages 127–138, New Delhi, IN, 18–20 Dec. 1992, LNCS 652, 1992. Springer-Verlag.
570. B. Monsuez. Polymorphic types and widening operators. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3^{rd} Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 267–281. Springer-Verlag, 22–24 Sep. 1993.
571. B. Monsuez. Polymorphic typing for call-by-name semantics. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Proc. FMPA*, Akademgorodok, Novosibirsk, RU, LNCS 735, pages 156–169. Springer-Verlag, 28 June – 2 Jul. 1993.
572. B. Monsuez. System F and abstract interpretation. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 279–295. Springer-Verlag, 1995.
573. B. Monsuez. Using abstract interpretation to define a strictness type inference system. In *Proc. PEPM '95*, La Jolla, CA, pages 122–133. ACM Press, 21–23 June 1995.
574. G. Morrisett, D. Tarditi, P. Cheng, C. Stone, R. Harper, and P. Lee. The TIL/ML compiler: Performance and safety through types. In *Workshop on Compiler Support for Systems Software, WCSSS '96*, Feb. 1996.
575. P.D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 11, pages 575–631. Elsevier, 1990.
576. C. Mossin. Exact flow analysis. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 250–264. Springer-Verlag, 1997.

577. C. Mossin. Higher-order value flow graphs. In H. Glaser, P.H. Hartel, and H. Kuchen, editors, *Proc. 9th Int. Symp. PLILP '97*, pages 159–173. Springer-Verlag, Southampton, UK, 3–5 Sep. 1997, LNCS 1292, 1997.
578. M. M ller, T. Glaß, and K. Stroetmann. Automated modular termination proofs for real Prolog programs. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 220–237. Springer-Verlag, 1996.
579. R. Muller and Y. Zhou. Abstract interpretation in weak powerdomains. *LISP Pointers*, 5(1):119–126, Jan. – Mar. 1992.
580. M. M ller-Olm, D.A. Schmidt, and B. Steffen. Model checking: a tutorial introduction. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 330–354. Springer-Verlag, 1999.
581. K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In K. Furukawa, editor, *Proc. 8th ICLP '91*, Paris, FR, pages 49–63. MIT Press, 24–28 June 1991.
582. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *J. Logic Programming*, 13(2–3):315–347, Jul. 1992.
583. A. Mycroft. The theory and practice of transforming call-by-need into call-by-value. In B. Robinet, editor, *Proc. 4th Int. Symp. on Programming*, Paris, FR, 22–24 Apr. 1980, LNCS 83, pages 270–281. Springer-Verlag, 1980.
584. A. Mycroft. *Abstract Interpretation and Optimising Transformations for Applicative Programs*. Ph.D. Dissertation, CST-15-81, Department of Computer Science, University of Edinburgh, Edinburg, UK, Dec. 1981.
585. A. Mycroft. Polymorphic type schemes and recursive definitions. In M. Paul and B. Robinet, editors, *Proc. 6th Int. Symp. on Programming*, Toulouse, FR, Apr. 1984, LNCS 167, pages 217–228. Springer-Verlag, 1984.
586. A. Mycroft. Incremental polymorphic type checking with update. In A. Nerode and M. Taitslin, editors, *Proc. LFCS – Tver'92*, LNCS 620. Springer-Verlag, 1992.
587. A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proc. PEPM '93*, Copenhagen, DK, 14–16 June 1993, pages 80–87. ACM Press, 1993.
588. A. Mycroft and N.D. Jones. A relational framework for abstract interpretation. In N.D. Jones and H. Ganzinger, editors, *Programs as Data Objects, Proceedings of a Workshop*, Copenhagen, DK, 17-19 Oct. 1985, LNCS 215, pages 156–171. Springer-Verlag, 1986.
589. A. Mycroft and F. Nielson. Strong abstract interpretation using power domains (extended abstract). In J. Diaz, editor, *10th ICALP, Barcelona, ES*, LNCS 154, pages 536–547. Springer-Verlag, 18–22 Jul. 1983.
590. A. Mycroft and M. Rosendahl. Minimal function graphs are not instrumented. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 60–67. IRISA, Rennes, FR, 23–25 Sep. 1992.
591. A. Mycroft and K.L. Solberg. Uniform PERs and comportment analysis. In M.V. Hermenegildo and S.D. Swierstra, editors, *Proc. 7th Int. Symp. PLILP '95*, Utrecht, NL, 20–22, Sep. 1995, LNCS 982, pages 169–187. Springer-Verlag, 1995.
592. P. Naur. Proofs of algorithms by general snapshots. *BIT*, 6:310–316, 1966.
593. G.C. Necula. Proof-carrying code. In *24th POPL*, pages 106–119, Paris, FR, Jan. 1997. ACM Press.
594. M. Neuberger and P. Mishra. A precise relationship between the deductive power of forward and backward strictness analysis. *LISP Pointers*, 5(1):127–138, Jan. – Mar. 1992.

595. F. Nielson. A denotational framework for data flow analysis. *Acta Informat.*, 18:265–287, 1982.
596. F. Nielson. Program transformation in a denotational setting. *TOPLAS*, 7(3):359–379, 1985.
597. F Nielson. Tensor product generalize the relational data flow analysis method. In *Proc. 4th Hungarian Computer Science Conference*, pages 211–225, 1985.
598. F. Nielson. Abstract interpretation of denotational definions (a survey). In B. Monien and G. Vidal-Naquet, editors, *Proc. 3rd Annual Symp. STACS '86*, Orsay, FR, LNCS 210, pages 1–20. Springer-Verlag, 16–18 Jan. 1986.
599. F. Nielson. Strictness analysis and denotational abstract interpretation. In *14th POPL*, pages 120–131, Munchen, DE, 1987. ACM Press.
600. F. Nielson. Strictness analysis and denotational abstract interpretation. *Inform. and Comput.*, 76(1):29–92, 1988.
601. F. Nielson and H. Riis Nielson. Finiteness conditions for fixed point iteration. *LISP Pointers*, 5(1):96–108, Jan. – Mar. 1992.
602. F. Nielson and H. Riis Nielson. Type and effect systems. In E.-R. Olderog and B. Steffen, editors, *Correct System Design, Recent Insight and Advances, (to Hans Langmaack on the occasion of his retirement from his professorship at the University of Kiel)*, pages 114–136. Springer-Verlag, 1999.
603. H.R. Nielson and F. Nielson. Automatic binding time analysis for a typed λ-calculus. *Sci. Comput. Programming*, 10:139–176, 1988.
604. H.R. Nielson and F. Nielson. Using transformmations in the implementation of higher-order functions. *J. Func. Prog.*, 1(4):459–494, 1991.
605. H.R. Nielson and F. Nielson. Bounded fixed-point iteration. *J. Logic and Comp.*, 2(4):437–464, Aug. 1992.
606. H.R. Nielson and F. Nielson. Bounded fixed point iteration (extended abstract). In *19th POPL*, pages 71–82, Albuquerque, NM, 1992. ACM Press.
607. H.R. Nielson and F. Nielson. The tensor product in Wadler's analysis of lists. In B. Krieg-Br ckner, editor, *Proc. 4th ESOP '92*, Rennes, FR, LNCS 582, pages 351–370. Springer-Verlag, 26–28 Feb. 1992.
608. H.R. Nielson and F. Nielson. Finiteness conditions for strictness analysis. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 194–205. Springer-Verlag, 22–24 Sep. 1993.
609. T. Nipkow and D. von Oheimb. Javalight is type-safe – definitely. In *25th POPL*, pages 161–170, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
610. M. Nordin, T. Lindgren, and H. Millroth. IGOR: A tool for developing Prolog dataflow analyzers. In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 420–421. Springer-Verlag, 1995.
611. M. Ogawa. Automatic verification based on abstract interpretation. In A. Middeldorp and T. Sato, editors, *4th FLOPS '99*, Tsukuba, JP, 11–13 Nov. 1999, LNCS 1722, pages 131–146. Springer-Verlag, 1999.
612. P.M. O'Keefe and J. Palsberg. A type system equivalent to flow analysis. In *22nd POPL*, pages 367–378, San Francisco, CA, 1995. ACM Press.
613. P.M. O'Keefe and J. Palsberg. A type system equivalent to flow analysis. *TOPLAS*, 17(4):576–599, 1995.
614. A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D.L. Dill, editor, *Proc. 6th Int. Conf. CAV '94*, Stanford, CA, US, LNCS 818, pages 81–94. Springer-Verlag, 21–23 June 1994.

615. M.-A. Oros and P.Y. Gloess. Inheritance in Datalog. In M. Ducassé, B. Le Charlier, Y.-J. Lin, and L. . Yalçinalp, editors, *Proc. 5th Workshop LPE 1993*, Vancouver, BC, CA, pages 52–58. IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, FR, 29–30 Oct. 1993.
616. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Proc. 8th Int. Conf. CAV '96*, New Brunswick, NJ, US, LNCS 1102, pages 411–414. Springer-Verlag, 31 Jul. –3 Aug. 1996.
617. S. Owre, J. Rushby, and N. Shankar. Integration in PVS: Tables, types, and model checking. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems, 3rd Int. Work. , TACAS '97*, number 1217 in LNCS, pages 366–383, Enschede, NL, 2–4 Apr. 1997. Springer-Verlag.
618. S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proc. 11th Int. Conf. CADE '92*, Saratoga Springs, NY, US, LNCS 607, pages 748–752. Springer-Verlag, 15–18 June 1992.
619. S. Owre, N. Shankar, and D.W.J. Stringer-Calvert. PVS: An experience report. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *PROC Applied Formal Methods - FM-Trends'98, International Workshop on Current Trends in Applied Formal Method*, Boppard, DE, LNCS 1641, pages 338–345. Springer-Verlag, 7–9 Oct. 1999.
620. É. Villemonte de la Clergerie P. Lef bvre. How to build quickly an efficient implementation of the domain Prop with DyALog. In M. Ducassé, B. Le Charlier, Y.-J. Lin, and L. . Yalçinalp, editors, *Proc. 5th Workshop LPE 1993*, Vancouver, BC, CA, pages 33–38. IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, FR, 29–30 Oct. 1993.
621. J. Palsberg. Correctness of binding-time analysis. *J. Func. Prog.*, 3(3):347–363, 1993.
622. J. Palsberg. Closure analysis in constraint form. *TOPLAS*, 17(1):47–62, Jan. 1995.
623. J. Palsberg. Efficient inference of object types. *Inform. and Comput.*, 123(2):198–209, Jan. 1995.
624. J. Palsberg and C. Pavlopoulou. From polyvariant flow information to intersection and union types. In *25th POPL*, pages 197–208, San Diego, CA, US, 19–21 Jan. 1998. ACM Press.
625. J. Palsberg and M.I. Schwartzbach. Static typing for object-oriented programming. *Sci. Comput. Programming*, 23(1):19–53, 1994.
626. S.E. Panitz and M. Schmidt-Schauß. TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 345–360. Springer-Verlag, 1997.
627. A. Pardo and G.D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In O. Grumberg, editor, *Proc. 9th Int. Conf. CAV '97*, Haifa, IL,LNCS 1254, pages 12–23. Springer-Verlag, 22–25 Jul. 1997.
628. S. Park, S. Das, and D.L. Dill. Automatic checking of aggregation abstractions through state enumeration. In *IFIP TC6/WG6.1 Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, pages 207–222, Nov. 1997.
629. R. Paterson. Compiling laziness using projections. In R. Cousot and D.A. Schmidt, editors, *Proc. 3rd Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 255–269. Springer-Verlag, 1996.

630. R. Paterson. Transforming lazy functions using comportment properties. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Programming Language Implementation and Logics of Programs*, volume 1292 of *LNCS*, pages 111–125. Springer-Verlag, September 1997.
631. R. Paterson. Transforming lazy functions using comportment properties. In H. Glaser, P.H. Hartel, and H. Kuchen, editors, *Proc. 9th Int. Symp. PLILP '97*, pages 111–125, Southampton, UK, 3–5 Sep. 1997, LNCS 1292, 1997. Springer-Verlag.
632. C. Paulin-Mohring and B. Werner. Synthesis of ML programs in the system Coq. *J. Symbolic Logic*, 15(5/6):607–640, 1993.
633. D. Peled. Ten years of partial order reduction. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th Int. Conf. CAV '98*, Vancouver, BC, CA,LNCS 1427, pages 17–28. Springer-Verlag, 28 June – 2 Jul. 1998.
634. C. Pixley and V. Singhal. Model checking: A hardware design perspective. *STTT*, 2(3):288–306, 1999.
635. G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, DK, Sep. 1981.
636. L. Pl mer. Automatic termination proofs for prolog programs operating on nonground terms. In K. Ueda V.A. Saraswat, editor, *Proc. 1991 Int. Symp. ISLP '91*, San Diego, CA, US, pages 503–517. MIT Press, 28 Oct. – 1 Nov. 1997.
637. A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, Providence, RI, Nov. 1977.
638. K. Post. Mutually exclusive rules in logic programming. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 472–486. MIT Press, 13–17 Nov. 1994.
639. G. Puebla, M.V. Hermenegildo, and J.P. Gallagher. An integration of partial evaluation in a generic abstract interpretation framework. In O. Danvy, editor, *Proc. PEPM '99*, San Antonio, TX, US, 17–19 June 1999, Tech. rep. BRICS-NS-99-1, pages 75–84. University of rhus, DK, 1999.
640. W. Pugh and D. Wonnacott. Static analysis of upper and lower bounds on dependences and parallelism. *TOPLAS*, 16(4):1248–1278, Jul. 1994.
641. J.-P. Queille and J. Sifakis. Verification of concurrent systems in Cesar. In *Proc. Int. Symp. on Programming*, LNCS 137, pages 337–351. Springer-Verlag, 1982.
642. G. Ramalingam. The undecidability of aliasing. *TOPLAS*, 16(5):1467–1471, Sep. 1994.
643. F. Randimbivololona, J. Souyris, and A. Deutsch. Improving avionics software verification cost-effectiveness: Abstract interpretation based technology contribution. In *Proceedings DASIA 2000 – DAta Systems In Aerospace*, Montreal, CA. ESA Publications, 22–26 May 2000.
644. U.S. Reddy and S.N. Kamin. On the power of abstract interpretation. In *Proc. 1992 ICCL*, Oakland, CA, pages 24–33. IEEE Comp. Soc. Press, 20–23 Apr. 1992.
645. U.S. Reddy and S.N. Kamin. On the power of abstract interpretation. *Comput. Lang.*, 19(2):79–89, 1993.
646. B. Reistad and D.K. Gifford. Static dependent costs for estimating execution time. In *Proc. ACM Conf. Lisp & Func. Prog.*, Orlando, FL, US, pages 65–78. ACM Press, 27–29 June 1994.
647. O. Ridoux, P. Boizumault, and F. Malésieux. Typed static analysis: Application to groundness analysis of PROLOG and lambda-PROLOG. In A. Middeldorp and T. Sato, editors, *4th FLOPS '99*, Tsukuba, JP, 11–13 Nov. 1999, LNCS 1722, pages 267–28. Springer-Verlag, 1999.

648. H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *27th POPL*, pages 142–154, Boston, MA, Jan. 2000. ACM Press.
649. M.C. Rinard and P.C. Diniz. Commutativity analysis: A new analysis technique for parallelizing compilers. *TOPLAS*, 19(6):942–991, Nov. 1997.
650. E. Rohwedder and F. Pfenning. Mode and termination checking for higher-order logic programs. In H. Riis Nielson, editor, *Proc. 6th ESOP '96*, Link ping, SE, LNCS 1058, pages 298–310. Springer-Verlag, 22–26 Apr. 1996.
651. M. Rosendahl. Higher order chaotic iteration sequences. In M. Bruynooghe and J. Penjam, editors, *Proc. 5th Int. Symp. PLILP '93*, Tallinn, EE, 25–27, Aug. 1993, LNCS 714, pages 332–345. Springer-Verlag, 1993.
652. E. Ruf and D. Weise. Improving the accuracy of higher-order specialization using control flow analysis. In *Proc. PEPM '92*, San Francisco, CA, US, pages 67–74. Yale University, Tech. rep. TR YALEU/DCS/RR-90, 19–20 June 1995.
653. J. Rushby. Automated deduction and formal methods. In R. Alur and T.A. Henzinger, editors, *Proc. 8th Int. Conf. CAV '96*, number 1102 in LNCS, pages 169–183, New Brunswick, NJ, Jul. /Aug. 1996. Springer-Verlag.
654. B.G. Ryder. Practical compile-time analysis. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 406–412. Springer-Verlag, 1997.
655. B. Rytz and M. Gengler. A polyvariant binding time analysis. In *Proc. PEPM '92*, San Francisco, CA, US, pages 21–28. Yale University, Tech. rep. TR YALEU/DCS/RR-90, 19–20 June 1995.
656. M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. In *23rd POPL*, pages 16–31, St. Petersburg Beach, FL, Jan. 1996. ACM Press.
657. M. Sagiv, T. Reps, and R. Wilhelm. Shape analysis. In *Proc. Int. Conf. CC '00*, 2000. To appear.
658. S. Sagiv, N. Francez, M. Rodeh, . Sagiv, and R. Wilhelm. A logic-based approach to data flow analysis problems. In P. Deransart and J. Małuszyński, editors, *Proc. 2nd Int. Work. PLILP '90*, pages 277–292, Link ping, SE, LNCS 456, 20–22 Aug. 1990. Springer-Verlag.
659. H. Sa di and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *Proc. 11th Int. Conf. CAV '99*, Trento, IT, LNCS 1633, pages 443–454. Springer-Verlag, 6–10 Jul. 1999.
660. S. Sa di. Model checking guided abstraction and analysis. In J. Palsberg, editor, *Proc. 7th Int. Symp. SAS '2000*, Santa Barbara, CA, US, LNCS 1824, pages 377–396. Springer-Verlag, 29 June – 1 Jul. 2000.
661. D. Sands. Complexity analysis for a lazy higher-order language. In K. Davis and J. Hughes, editors, *Functional Programming, Glasgow 1989*, Proc. 1989 Glasgow Workshop, Fraserburgh, UK, pages 56–79. Springer-Verlag and BCS, Aug. 1989.
662. C.F. Schaefer and G.N. Bundy. Static analysis of exception handling in Ada. *Soft.–Pract. & Exp.*, 23(10):1157–1174, Oct. 1993.
663. D.A. Schmidt. Natural-semantics-based abstract interpretation (preliminary version). In A. Mycroft, editor, *Proc. 2nd Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 1–18. Springer-Verlag, 1995.
664. D.A. Schmidt. Abstract interpretation of small-step semantics. In M. Dam, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, Stockhlom, SE, 24–26 June 1996, LNCS 1192, pages 76–99. Springer-Verlag, 1997.
665. D.A. Schmidt. Data-flow analysis is model checking of abstract interpretations. In *25th POPL*, pages 38–48, San Diego, CA, 19–21Jan. 1998. ACM Press.

666. D.A. Schmidt and B. Steffen. Program analysis *as* model checking of abstract interpretations. In G. Levi, editor, *Proc. 5^{th} Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 351–380. Springer-Verlag, 1998.
667. J. Schneider and C. Ferdinand. Pipeline behavior prediction for superscalar processors by abstract interpretation. In *Proc. LCTES '99, ACM SIGPLAN Not. 34(79)*, pages 35–44, Atlanta, GE, US, 5 May 1999, Jul. 1999. ACM Press.
668. P. Schnorf, M. Ganapathi, and J.L. Hennessy. Compile-time copy elimination. *Soft.–Pract. & Exp.*, 23(11):1175–1200, Nov. 1993.
669. B. Scholz, J. Blieberger, and T. Fahringer. Symbolic pointer analysis for detecting memory leaks. In *Proc. PEPM '00, ACM SIGPLAN Not. 34(11)*, Boston, MA, US, 22–23 Jan. 2000, pages 104–113. ACM Press, Nov. 1999.
670. E. Sch n. On the computation of fixpoints in static program analysis with an application to analysis of AKL. Res. rep. R95:06, Swedish Institute of Computer Science, SICS, 1995.
671. D. De Schreye and K. Verschaetse. Deriving linear size relations for logic programs by abstract interpretation. *New Gen. Comp.*, 13(2):117–154, 1995.
672. D. De Schreye, K. Verschaetse, and M. Bruynooghe. A framework for analyzing the termination of definite logic programs with respect to call patterns. In ICOT Staff, editor, *Proc. Int. Conf. on 5^{th} Generation Computer Systems 92*, Tokyo, JP, 1-5 June 1992, pages 481–488. IOS Press, 1992.
673. D. Scott. Lambda calculus: Some models, some philosophy. In J. Barwise, H.J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 223–265. North-Holland, 1980.
674. F. Scozzari. Logical optimality of groundness analysis. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 83–97. Springer-Verlag, 1997.
675. H. Seidel. Least solutions of equations over N. In S. Abiteboul and E. Shamir, editors, *Proc. ICALP '96*, LNCS 820, pages 400–411. Springer-Verlag, 1995.
676. R.C. Sekar, P. Mishra, and I.V. Ramakrishnan. On the power and limitation of strictness analysis based on abstract interpretation. In *18^{th} POPL*, pages 37–48, Orlando, FL, 1991. ACM Press.
677. R.C. Sekar, Shaunak Pawagi, and I.V. Ramakrishnan. Small domains spell fast strictness analysis. In *17^{th} POPL*, pages 169–183, San Francisco, CA, 1990. ACM Press.
678. R.C. Sekar and I.V. Ramakrishnan. Fast strictness analysis based on demand propagation. *TOPLAS*, 17(6):896–937, Nov. 1995.
679. J. Seward. Beyond prototype implementations: Polymorphic projection analysis for Glasgow Haskell. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 382–399. Springer-Verlag, 1995.
680. A. Shamir and W.W. Wadge. Data types as objects. In A. Salomaa and M. Steinby, editors, *4^{th} ICALP, Turku*, LNCS 52, pages 465–479. Springer-Verlag, Jul. 1977.
681. N. Shankar. PVS: Combining specification, proof checking, and model checking. In M.S. Srivas and A.J. Camilleri, editors, *Proc. 1^{st} Int. Conf. on Formal Methods in Computer-Aided Design, FMCAD '96*, number 1166 in LNCS, pages 257–264, Palo Alto, CA, US, 6–8 Nov. 1996. Springer-Verlag.
682. N. Shankar. Unifying verification paradigms. In *FTRTFT'96*, 1996.
683. O. Shivers. The semantics of scheme control-flow analysis. In P. Hudak and N.D. Jones, editors, *Proc. PEPM '91*, Yale U., New Haven, CT, US, 17–19 June 1991, ACM SIGPLAN Not. 26(9), pages 190–198. ACM Press, Sep. 1991.

684. O. Shivers. Useless-variable elimination. *Actes JTASPEFL '91, Bordeaux, FR. BIGRE*, 74:197–201, Oct. 1991.
685. P.A. Bigot S.K. Debray, D. Gudeman. Detection and optimization of suspension-free logic programs. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, Ithaca, NY, US, pages 487–501. MIT Press, 13–17 Nov. 1994.
686. K. Sohn. Constraints among argument sizes in logic programs. In *Proc. 30th PODS '94*, pages 68–76, Minneapolis, MN, 1994. ACM Press.
687. K. Lackner Solberg. Strictness and totality analysis. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 395–407. Springer-Verlag, 1994.
688. K. Lackner Solberg, H. Riis Nielson, and F. Nielson. Inference systems for binding time analysis. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 247–254. IRISA, Rennes, FR, 23–25 Sep. 1992.
689. K.L. Solberg Gasser, H. Riis Nielson, and F. Nielson. Strictness and totality analysis. *Sci. Comput. Programming*, 31(1):113–145, May 1998.
690. H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP '86*, Saarbr cken, DE, 17-19 Mar. 1986, LNCS 213, pages 327–338. Springer-Verlag, 1986.
691. C. Speirs, Z. Somogyi, and H. Søndergaard. Termination analysis for Mercury. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 160–171. Springer-Verlag, 1997.
692. D. Stefanescu and Y. Zhou. An equational framework for the flow analysis of higher order functional programs. In *Proc. ACM Conf. Lisp & Func. Prog.*, Orlando, FL, US, pages 318–327. ACM Press, 27–29 June 1994.
693. B. Steffen, editor. *Evaluation of Alternating Fixed Points: Fully Local and Efficient*, Lisbon, PT, LNCS 1384. Springer-Verlag, 28 Mar. – 4 Apr. 1998.
694. B. Steffen, editor. *A Proof of Burns N-Process Mutual Exclusion Algorithm Using Abstraction*, Lisbon, PT, LNCS 1384. Springer-Verlag, 28 Mar. – 4 Apr. 1998.
695. B. Steffen, editor. *Ten Years of Partial Order Reduction*, Lisbon, PT, LNCS 1384. Springer-Verlag, 28 Mar. – 4 Apr. 1998.
696. J. Stransky. A lattice for abstract interpretation of dynamic (LISP-like) structures. *Inform. and Comput.*, 101(1):70–102, Nov. 1992.
697. R. Sundararajan and J.S. Conery. An abstract interpretation scheme for groundedness, freeness, and sharing analysis of logic programs. In R. Shyamasundar, editor, *Proc. 12th FST & TCS*, New Delhi, IN, 18–20 Dec. 1992, LNCS 652, pages 203–216. Springer-Verlag, 1992.
698. M. Tadjouddine, F. Eyssette, and C. Faure. Sparse jacobian computation in automatic differentiation by static program analysis. In G. Levi, editor, *Proc. 5th Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 311–326. Springer-Verlag, 1998.
699. Y. Takayama. Extraction of concurrent processes from higher dimensional automata. In H. Kirchner, editor, *Proc. 21st CAAP '96*, Link ping, SE, LNCS 1059, pages 72–86. Springer-Verlag, 22–26 Apr. 1996.
700. J. Tan and I-P. Lin. Type synthesis for logic programs. In M.J. Maher, editor, *Proc. JICSLP '96*, Bonn, DE, pages 200–214. MIT Press, 2–6 Sep. 1996.
701. Y.M. Tang and P. Jouvelot. Control-flow effects for escape analysis. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Proc. 2nd Int. Work. WSA '92, Bordeaux, FR. BIGRE*, volume 81–82, pages 313–321. IRISA, Rennes, FR, 23–25 Sep. 1992.

702. Y.M. Tang and P. Jouvelot. Separate abstract interpretation for control-flow analysis. In M. Hagiya and J.C. Mitchell, editors, *Proc. Int. Conf. TACS '95, LNCS 789*, Sendai, JP, pages 224–243. Springer-Verlag, 19–22 Apr. 1994.
703. D. Tarditi, J.G. Morrisett, P. Cheng, R. Harper, and P. Lee. TIL: A type-directed optimizing compiler for ML. In *Proc. ACM SIGPLAN '96 Conf. PLDI. ACM SIGPLAN Not. 31(5)*, pages 181–192, Philadephia, PA, US, 21–24 May 1996.
704. A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–310, 1955.
705. R.D. Tennent. Denotational semantics. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Semantic Structures*, volume 3 of *Handbook of Logic in Computer Science*, chapter 2, pages 169–332. Clarendon Press, 1994.
706. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier, 1990.
707. M. Tofte. Region inference for higher-order functional languages. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 19–20. Springer-Verlag, 1995.
708. D. Toman. Constraint databases and program analysis using abstract interpretation. In V. Gaede, A. Brodsky, O. G nther, D. Srivastava, V. Vianu, and M. Wallace, editors, *Proc. 2^{nd} Int. Work. CDBS '97*, Delphi, GR, 11–12 Jan. 1997, LNCS 1191, pages 303–311. Springer-Verlag, 1997.
709. H. Touati, H. Savoj, B. Lin, R.H. Brayton, and S. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDDs. In A. Sangiovanni-Vincentelli and S. Goto, editors, *Proc. Int. Conf. ICCAD '90*, pages 130–133, Santa Clara, CA, Nov. 1990. IEEE Comp. Soc. Press.
710. K.R. Traub, D.E. Culler, and K.E. Schauser. Global analysis for partitioning non-strict programs into sequential threads. *LISP Pointers*, 5(1):324–334, Jan. – Mar. 1992.
711. R. Triolet, P. Feautrier, and F. Irigoin. Automatic parallelization of fortran programs in the presence of procedure calls. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP '86*, Saarbr cken, DE, 17-19 Mar. 1986, LNCS 213, pages 210–222. Springer-Verlag, 1986.
712. S. Tzolovski. Data dependence as abstract interpretations. In P. Van Hentenryck, editor, *Proc. 4^{th} Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, page 366. Springer-Verlag, 1997.
713. K. Ueda. Linearity analysis of concurrent logic programs. *ENTCS*, 30(2), 1999.
714. J.S. Uhl and R.N. Horspool. Flow grammars — a flow analysis methodology. In P.A. Fritzson, editor, *Proc. 5^{th} Int. Conf. CC '94*, Edinburg, UK, LNCS 786, pages 203–217. Springer-Verlag, Apr. 1994.
715. R. Vallée-Rai, H. Hendren, P. Lam, É Gagnon, and P. Co. Soot - a Java^{tm} optimization framework. In *CASCON '99*, Sep. 1999.
716. R.J. van Glabbeek. The linear time – branching time spectrum (extended abstract). In J.C.M. Baeten and J.W. Klop, editors, *Proc. CONCUR'90, Theories of Concurrency: Unification and Extension, Amsterdam, Aug. 1990*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, 1990.
717. M.Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In J.-P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5^{th} Int. Symp. AMAST Workshop, ARTS '99*, Bamberg, DE, 26–28 May 1999, LNCS 1601, pages 265–276. Springer-Verlag, 1993.

718. F. Védrine. Binding-time analysis and strictness analysis by abstract interpretation. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 400–417. Springer-Verlag, 1995.
719. A. Venet. Abstract cofibred domains: Application to the alias analysis of untyped programs. In R. Cousot and D.A. Schmidt, editors, *Proc. 3^{rd} Int. Symp. SAS '96*, Aachen, DE, 24–26 Sep. 1996, LNCS 1145, pages 368–382. Springer-Verlag, 1996.
720. A. Venet. Abstract interpretation of the π-calculus. In M. Dam, editor, *Analysis and Verification of Multiple-Agent Languages, 5^{th} LOMAPS Workshop*, Stockhlom, SE, 24–26 June 1996, LNCS 1192, pages 51–75. Springer-Verlag, 1997.
721. A. Venet. Automatic determination of communication topologies in mobile systems. In G. Levi, editor, *Proc. 5^{th} Int. Symp. SAS '98*, Pisa, IT, 14–16 Sep. 1998, LNCS 1503, pages 152–167. Springer-Verlag, 1998.
722. A. Venet. Automatic analysis of pointer aliasing for untyped programs. *Sci. Comput. Programming, Special Issue on SAS'96*, 35(1):223–248, September 1999.
723. S. Verbaeten, K.F. Sagonas, and D. De Schreye. Modular termination proofs for prolog with tabling. In *Proc. Int. Conf. PPDP '99*, Paris, FR, 29 Sep. – 1 Oct. 1999, LNCS 1702, pages 342–359. Springer-Verlag, 1999.
724. S. Verbaeten and D. De Schreye. Termination analysis of tabled logic programs using mode and type information. In A. Middeldorp and T. Sato, editors, *4^{th} FLOPS '99*, Tsukuba, JP, 11–13 Nov. 1999, LNCS 1722, pages 163–178. Springer-Verlag, 1999.
725. B. Vergauwen, J. Wauman, and J. Lewi. Efficient fixpoint computation. In B. Le Charlier, editor, *Proc. 1^{st} Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 314–328. Springer-Verlag, 1994.
726. K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In K. Furukawa, editor, *Proc. 8^{th} Int. Conf. on Logic Programming*, Paris, FR, pages 301–315. MIT Press, 24–28 June 1991.
727. K. Verschaetse, S. Decorte, and D. De Schreye. Automatic termination analysis. In K.-K. Lau and T.P. Clement, editors, *Proc. 5^{th} Int. Work. LOPSTR '92*, Manchester, UK, 2–3 Jul. 1992, Workshops in Comp., pages 168–183. Springer-Verlag, 1993.
728. J. Vitek, R.N. Horspool, and J.S. Uhl. Compile-time analysis of object-oriented programs. In U. Kastens and P. Pfahler, editors, *Proc. 4^{th} Int. Conf. CC '98*, Paderborn, DE, LNCS 641, pages 236–250. Springer-Verlag, 5-7 Oct. 1998.
729. P. Wadler. Strictness analysis aids time analysis. In *15^{th} POPL*, pages 119–132, San Diego, CA, Jan. 1988. ACM Press.
730. P.L. Wadler. Strictness analysis on non-flat domains (by abstract interpretation over finite domains). In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, Computers and their Applications, chapter 12, pages 266–275. Ellis Horwood, 1987.
731. P.L. Wadler and R.J.M. Hughes. Projections for strictness analysis. In G. Kahn, editor, *Proc. 2^{nd} FPCA*, LNCS 274, pages 385–407. Springer-Verlag, Portland, OR, Sep. 1987.
732. F. Wang. Efficient data structure for fully symbolic verification of real-time software systems. In S. Graf and M.I. Schwartzbach, editors, *Proc. 6^{th} Int. Conf. TACAS '2000*, Berlin, DE, 25 Mar. – 2 Apr. 2000, LNCS 1785, pages 157–171. Springer-Verlag, 2000.
733. P. Wolper and B. Boigelot. An automata-theoretic approach to presburger arithmetic constraints. In A. Mycroft, editor, *Proc. 2^{nd} Int. Symp. SAS '95*, Glasgow, UK, 25–27 Sep. 1995, LNCS 983, pages 21–32. Springer-Verlag, 1995.

734. A.K. Wright and R. Cartwright. A practical soft type system for Scheme. In *ACM Conf. Lisp & Func. Prog.*, pages 250–262, 1994.
735. D.A. Wright. A new technique for strictness analysis. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. Int. J. Conf. TAPSOFT '91*, Brighton, UK, Volume 2 (ADC/CCPSD), LNCS 494, pages 236–258. Springer-Verlag, 1991.
736. D.A. Wright and C.A. Baker-Finch. Usage analysis with natural reduction types. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Proc. 3rd Int. Work. WSA '93*, Padova, IT, LNCS 724, pages 254–266. Springer-Verlag, 22–24 Sep. 1993.
737. Kwangkeun Yi. Compile-time detection of uncaught exception in Standard ML programs. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, BE, 20–22 Sep. 1994, LNCS 864, pages 238–254. Springer-Verlag, 1994.
738. Kwangkeun Yi. An abstract interpretation for estimating uncaught exceptions in standard ML programs. *Sci. Comput. Programming*, 31(1):147–173, May 1998.
739. Kwangkeun Yi and Sukyoung Ryu. Towards a cost-effective estimation of uncaught exceptions in SML programs. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 98–113. Springer-Verlag, 1997.
740. E. Zaffanella, R. Bagnara, and P.M. Hill. Widening sharing. In G. Nadathur, editor, *Proc. Int. Conf. PPDP '99*, Paris, FR, 29 Sep. – 1 Oct. 1999, LNCS 1702, pages 414–431. Springer-Verlag, 1999.
741. E. Zaffanella, R. Giacobazzi, and G. Levi. Abstracting synchronization in concurrent constraint programming. In M.V. Hermenegildo and J. Penjam, editors, *Proc. 6th Int. Symp. PLILP '94*, Madrid, ES, 14–16 Sep. 1994, LNCS 844, pages 57–72. Springer-Verlag, 1994.
742. E. Zaffanella, P.M. Hill, and R. Bagnara. Decompositing non-redundant sharing by complementation. In A. Cortesi and G. Filé, editors, *Proc. 6th Int. Symp. SAS '99*, Venice, IT, 22–24 Sep. 1999, LNCS 1694, pages 69–84. Springer-Verlag, 1999.
743. F. Zartmann. Denotational abstract interpretation of functional logic programs. In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, FR, 8–10 Sep. 1997, LNCS 1302, pages 141–159. Springer-Verlag, 1997.

Despite the author's exhaustiveness efforts, the bibliography is still incomplete and missing relevant references are welcomed.