History of Abstract Interpretation

HFM 2019

Roberto Giacobazzi
Francesco Ranzato

Acknowledgements

These slides are much indebted and inspired from
Patrick Cousot's talk

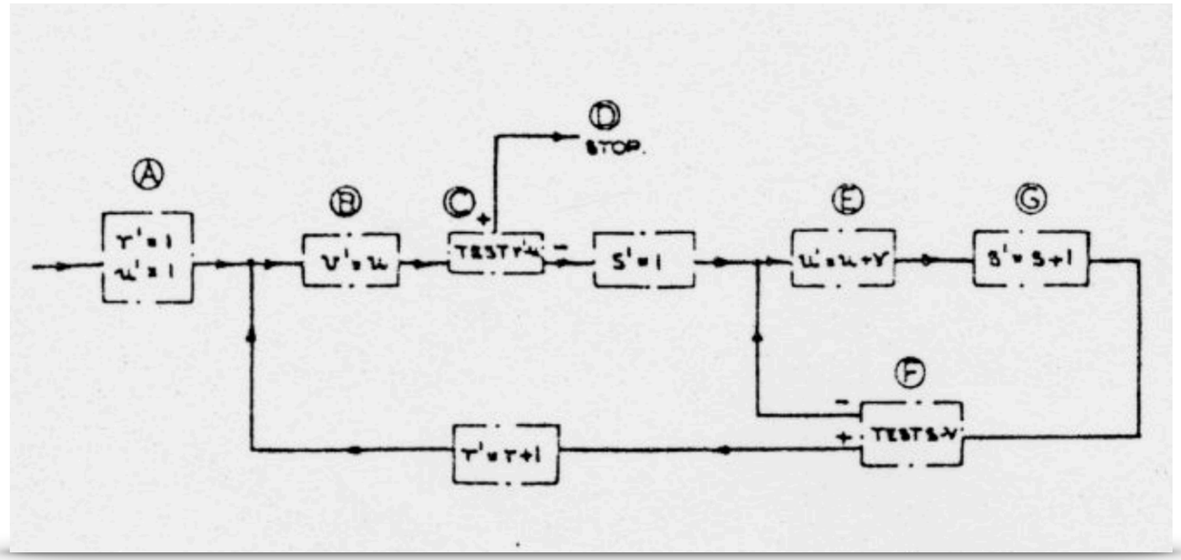
Abstract Interpretation: 40 years back + some years ahead

at the
Next 40 years of Abstract Interpretation (N40AI)
Workshop held in Paris on January 21st 2017
part of POPL2017 week

The cultural context

Historical Context in the 1940 — 1970

Flow diagrams



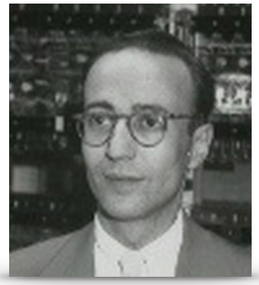
PLANNING AND CODING OF PROBLEMS
FOR AN
ELECTRONIC COMPUTING INSTRUMENT

BY

Herman H. Goldstine John von Neumann

Report on the Mathematical and Logical aspects of an
Electronic Computing Instrument

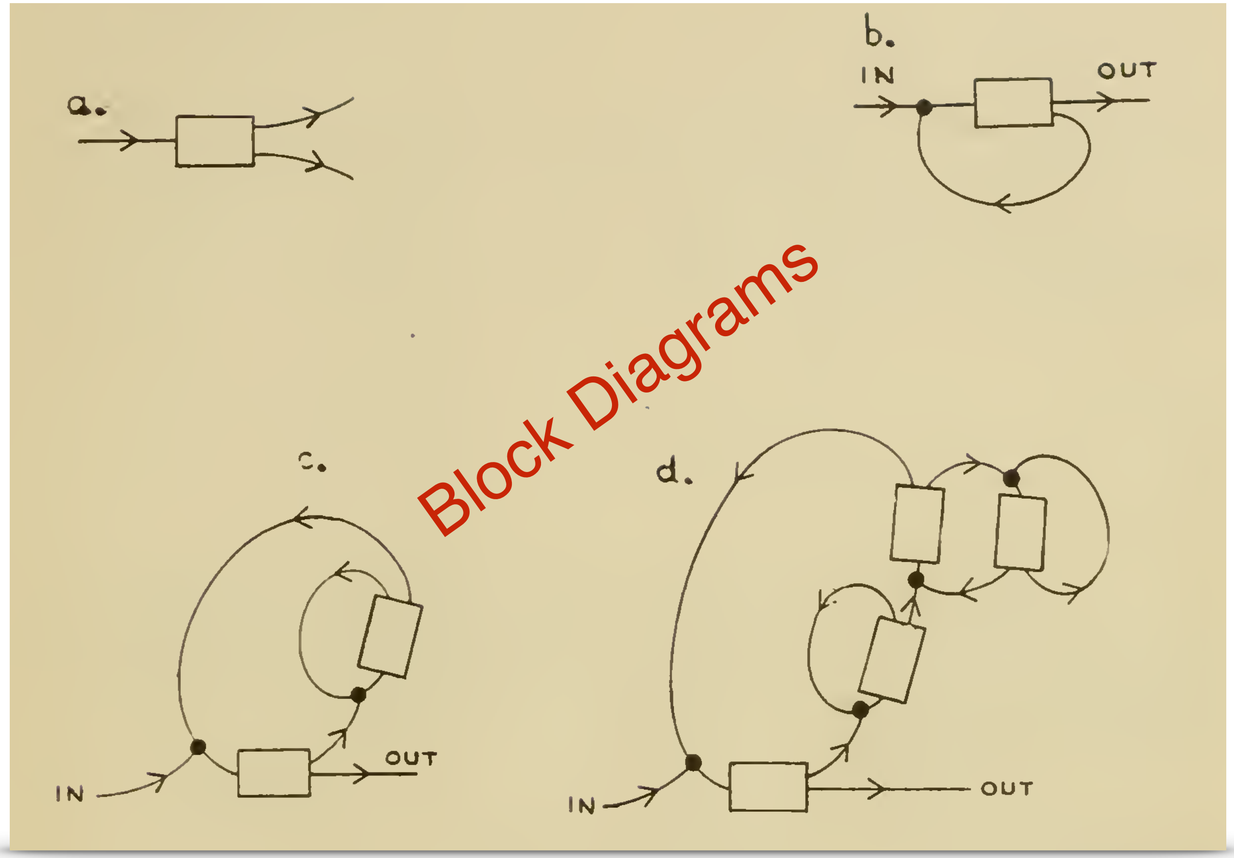
Part II, Volume I - 3

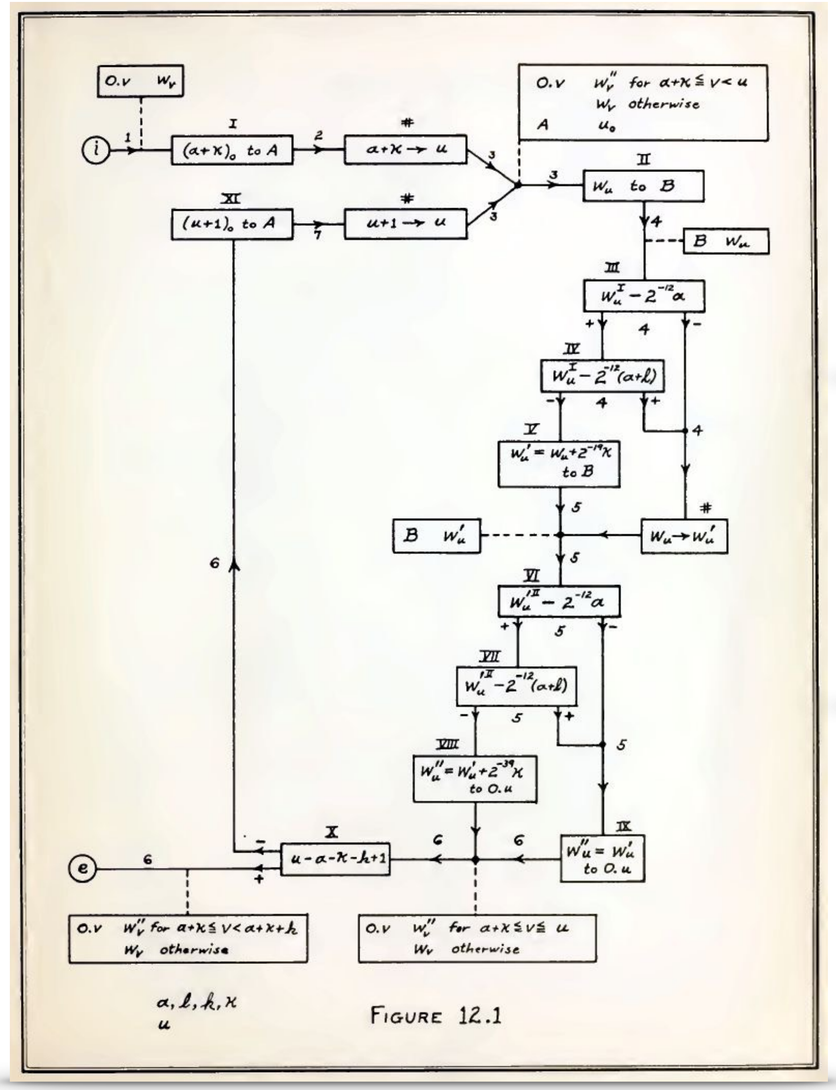


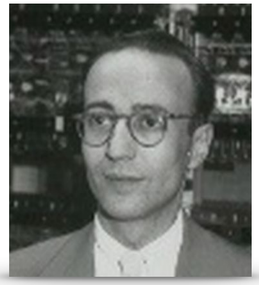
Turing, A. M. 1949

H.H. Goldstine and J. von Neumann. 1947

Historical Context in the 1940 — 1970







H.H. Goldstine and J. von Neumann. 1947

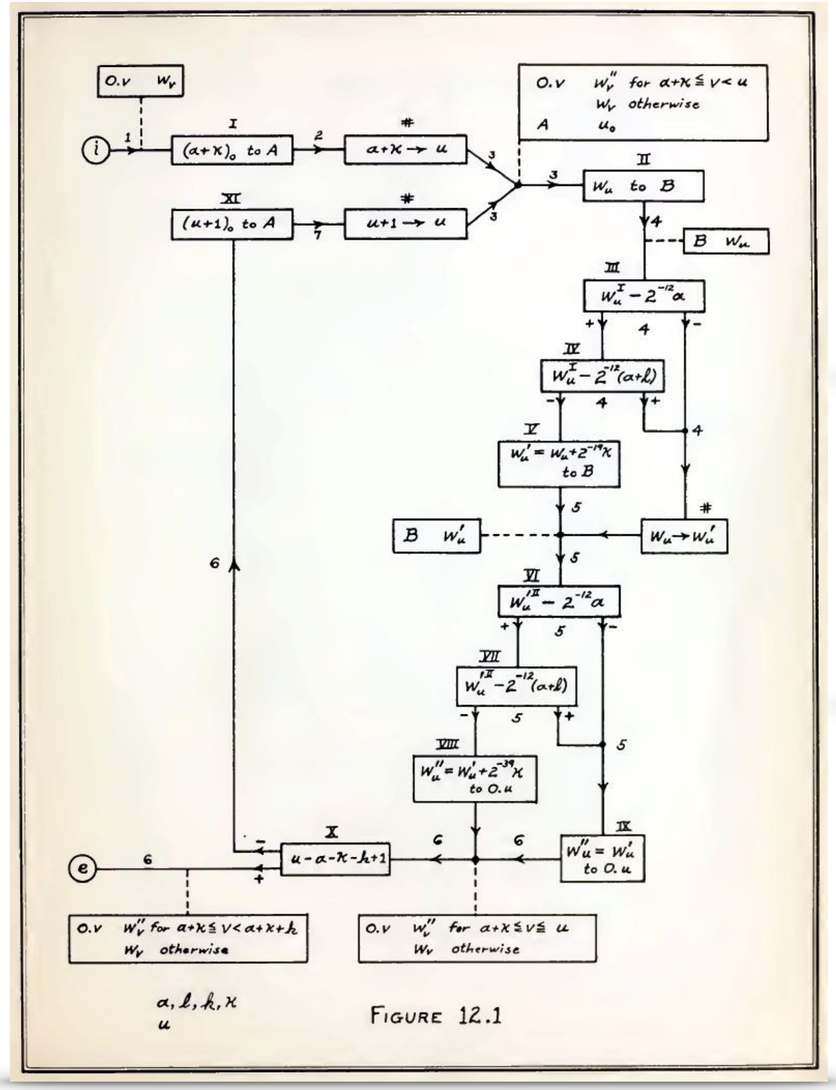
Historical Context in the 1940 — 1970

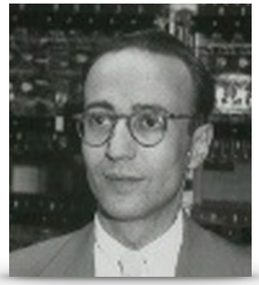
The static coding of the boxes I-XI follows:

C.1		$2^{-19}a$			
C.2		$2^{-19}\kappa$			
I,1		C.1		Ac	$2^{-19}a$
2		C.2	h	Ac	$2^{-19}(a+\kappa)$
3		s.1	Sp^{1}	s.1	$2^{-39}(a+\kappa)$
4		s.1	h	Ac	$(a+\kappa)_0$
5		A	S	A	$(a+\kappa)_0$
	(to II,1)				
II,1		A		Ac	u_0
2		II,3	Sp	II,3	u
3		-			
[		u	]	Ac	w_u
4		B	S	B	w_u
	(to III,1)				
C.3		2^{-32}			
III,1		C.3	R	R	2^{-32}
2		B	x	R	$\{w_u(9), w_u(9), w_u(10), \ldots, w_u(40)\} = w_u(9) + w_u^{I} + \varepsilon_u$
3			A	c	$w_u(9) + w_u^{I} + \varepsilon_u$
4		III.1,1	Cc		
C.4		-1			
III,5		C.4	h	Ac	$w_u^{1} + \varepsilon_u$
	(to III.1,1)				
C.5		2^{-7}			
III.1,1		C.5	R	R	2^{-7}
2		s.1	S	s.1	$w_u^{I} + \varepsilon_u$
3		s.1	x	Ac	$2^{-7}(w_u^{I} + \varepsilon_u)$
4		C.1	h-	Ac	$2^{-7}(w_u^{I} + \varepsilon_u) - 2^{-19}a$
5		IV,1	Cc		
	(to VI,1)				
C.6		$2^{-19}\chi$			
IV,1		C.6	h-	Ac	$2^{-7}(w_u^{I} + \varepsilon_u) - 2^{-19}(a+\chi)$
2		VI,1	Cc		
	(to V,1)				
V,1		C.2		Ac	$2^{-19}\kappa$
2		B	h	Ac	$w_u + 2^{-19}\kappa = w_u'$
3		B	S	B	w_u'
	(to VI,1)				
C.7		2^{-12}			

Assertion boxes

See Mark's presentation



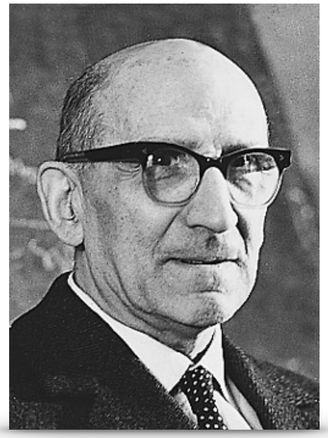


H.H. Goldstine and J. von Neumann. 1947

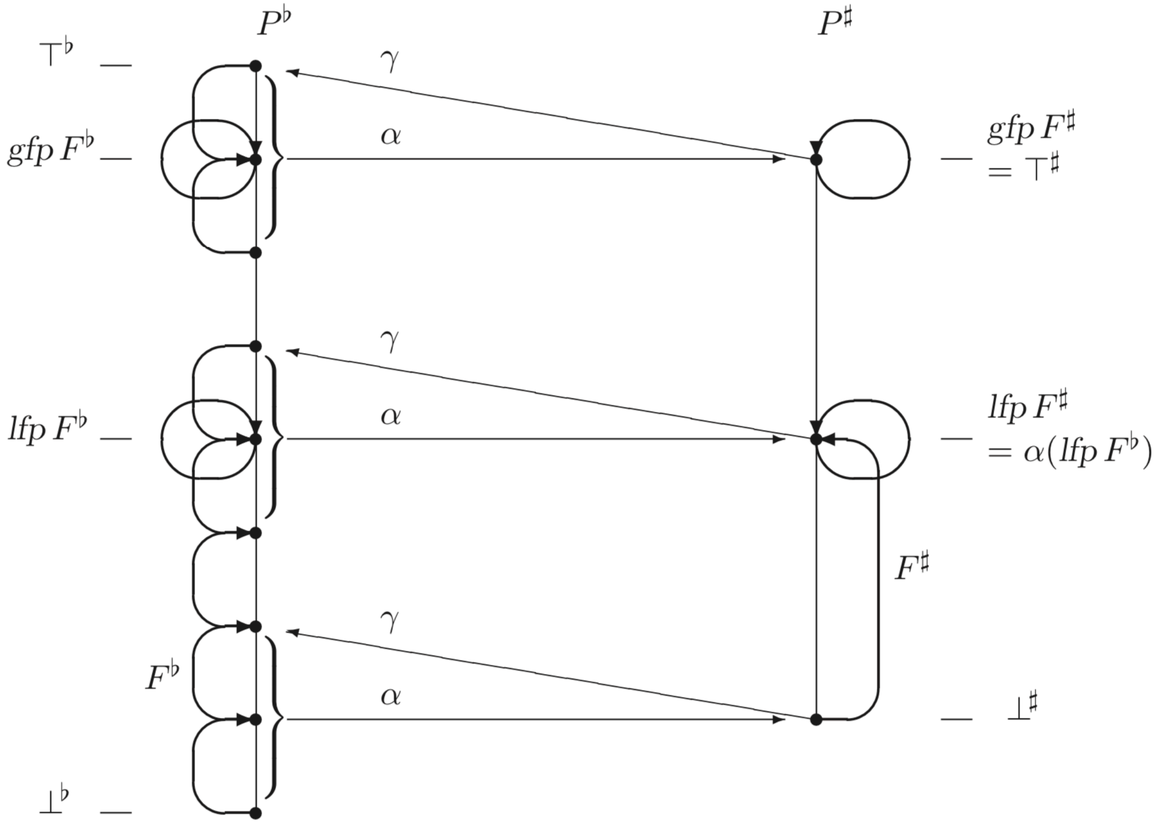
Historical Context in the 1940 — 1970

B. Knaster: *Un théorème sur les fonctions d'ensembles.*

M. K. communique les résultats suivants, obtenus en commun par M. Tarski et lui.



1928



A LATTICE-THEORETICAL FIXPOINT THEOREM AND ITS APPLICATIONS

ALFRED TARSKI

1955

Historical Context in the 1940 — 1970

ANNALS OF MATHEMATICS
Vol. 43, No. 2, April, 1942

THE CLOSURE OPERATORS OF A LATTICE

BY MORGAN WARD

(Received January 29, 1940)

Historical Context in the 1940 — 1970

PORTUGALIAE MATHEMATICA
Vol. 19 — Fasc. 2 — 1960

SOME RESULTS ON CLOSURE OPERATORS OF PARTIALLY ORDERED SETS*

PORTUGALIAE MATHEMATICA
Vol. 21 — Fas. 3 — 1962

A CHARACTERIZATION OF THE CLOSURE OPERATORS BY MEANS OF ONE AXIOM*

BY JOSÉ MORGADO
Instituto de Física e Matemática, Universidade do Recife, Brasil

PORTUGALIAE MATHEMATICA
Vol. 23 — Fasc. 1 — 1964

NOTE ON THE DISTRIBUTIVE CLOSURE OPERATORS OF A COMPLETE LATTICE (*)

e do Recife, Brasil

C1: $x \leq \varphi(x)$, *for every* $x \in P$;

C2: *If* $x \leq y$, *then* $\varphi(x) \leq \varphi(y)$;

C3: $\varphi(\varphi(x)) = \varphi(x)$, *for every* $x \in P$.

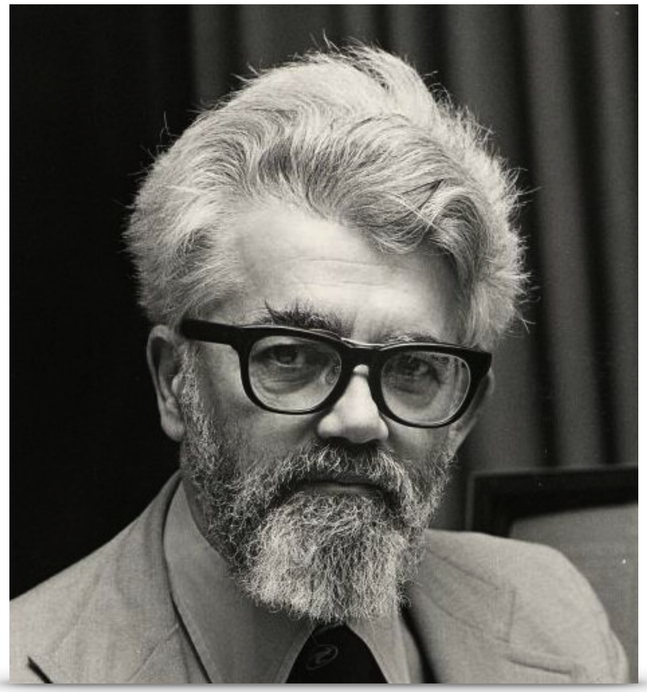
FACTORIZATION OF THE LATTICE OF CLOSURE OPERATORS OF A COMPLETE LATTICE*

BY JOSÉ MORGADO
Instituto de Física e Matemática, Universidade do Recife, Brasil

Historical Context in the 1940 — 1970

A BASIS FOR A MATHEMATICAL THEORY OF COMPUTATION[1])

JOHN McCARTHY



1963

Bridge between computability and programming!

Historical Context in the 1970s

Peter Naur 1966.
Proof of algorithms by general **snapshots**

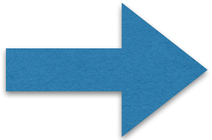
Greatest number, with snapshots

comment *General Snapshot* 1: $1 \leqq N$;
$r := 1$;
comment *General Snapshot* 2: $1 \leqq N$, $r = 1$;
for $i := 2$ **step** 1 **until** N **do**
  begin comment *General Snapshot* 3: $2 \leqq i \leqq N$, $1 \leqq r \leqq i-1$,
  $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[i-1]$;
  if $A[i] > A[r]$ **then** $r := i$;
  comment *General Snapshot* 4: $2 \leqq i \leqq N$, $1 \leqq r \leqq i$, $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[i]$;
  end;
comment *General Snapshot* 5: $1 \leqq r \leqq N$, $A[r]$ *is the greatest among the elements* $A[1], A[2], \ldots, A[N]$;
$R := A[r]$;
comment *General Snapshot* 6: R *is the greatest value of any element,* $A[1], A[2], \ldots, A[N]$;

The stepping stone of the approach is what is called General Snapshots, i.e. expressions of static conditions existing whenever the execution of the algorithm reaches particular points. General Snapshots are further shown to be useful for constructing algorithms.

Historical Context in the 1970s

Peter Naur 1966.
Proof of algorithms by general **snapshots**



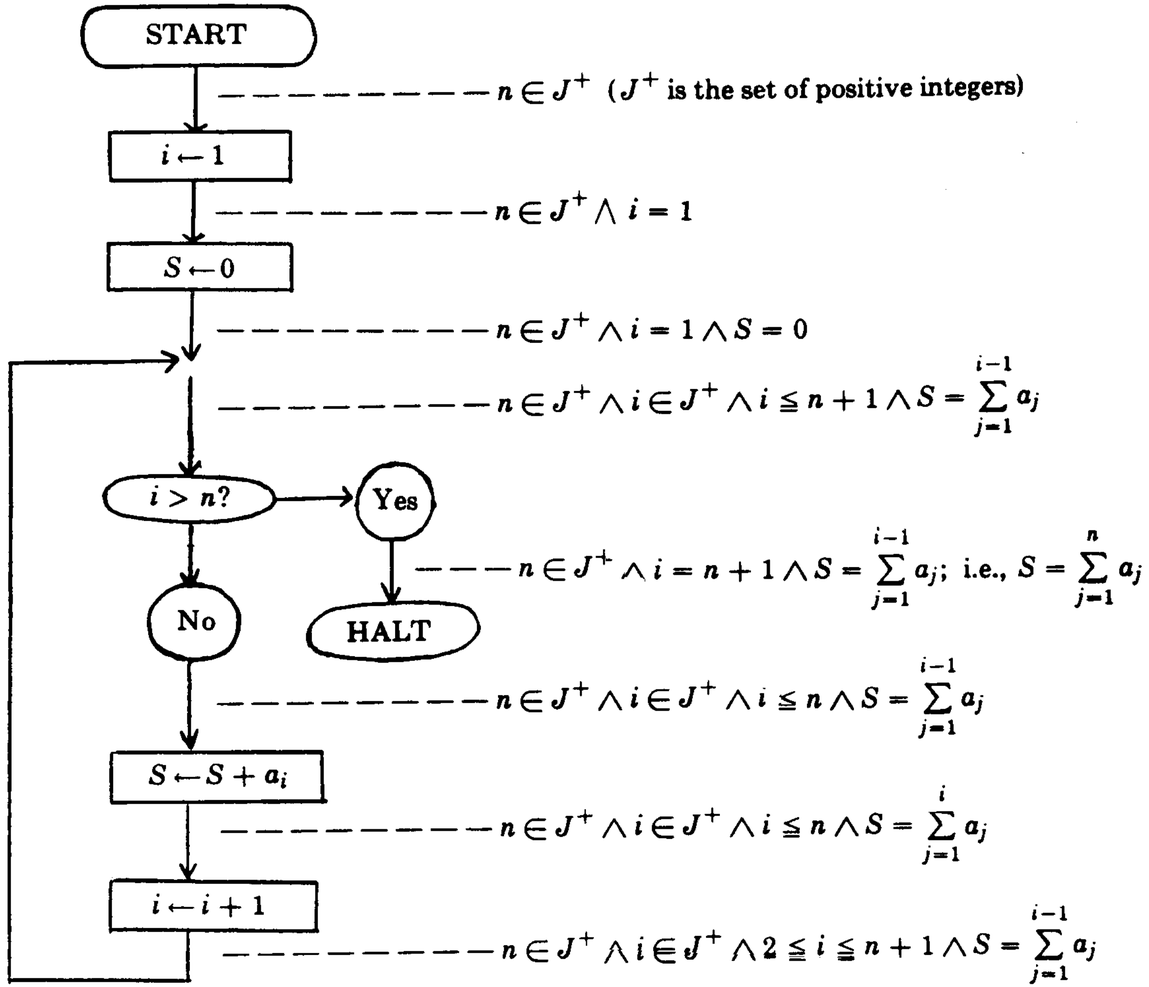
Snapshots.

Our proof problem is one of relating a static description of a result to a dynamic description of a way to obtain the result. Basically there are two ways of bringing the two descriptions closer together, either we may try to make the static description more dynamic, with a hope of getting to the given algorithm, or we may try to make the dynamic description more static. Of these the second is clearly preferable because we have far more experience in manipulating static descriptions, through practise in dealing with mathematical formulae. Therefore, if only we can derive some static description from the dynamic one, there is good hope that we may manipulate it so as to show that it is identical with the given static description.

Related to State Vectors in McCarthy's work on ALGOL

Historical Context in the 1970s

Robert Floyd 1967.
Assigning meanings to programs.



1. J. McCarthy, "A basis for a mathematical theory of computation" in *Computer programming and formal systems*, North-Holland, Amsterdam, 1963, pp. 33-70.

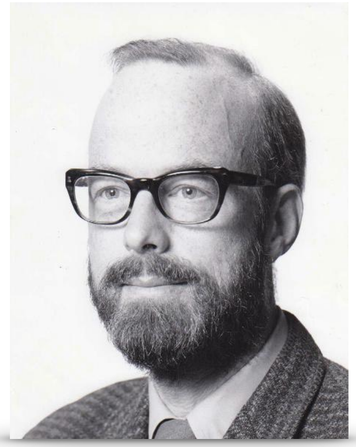
2. _____, *Towards a mathematical science of computation*, Proc. IFIP Congr. 62, North-Holland, Amsterdam, 1962, pp. 21-28.

Historical Context in the 1970s

J.W. DeBakker 1968
Axiomatics of simple assignment statements

R. Burstall 1968
Proving Properties of programs by structural induction

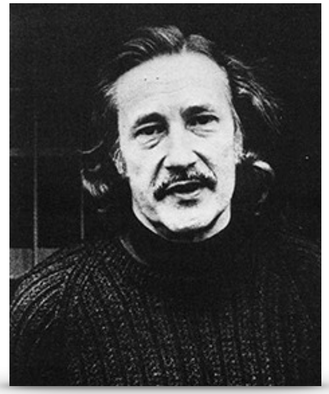
C.A.R Hoare 1969.
An axiomatic basis for computer programming

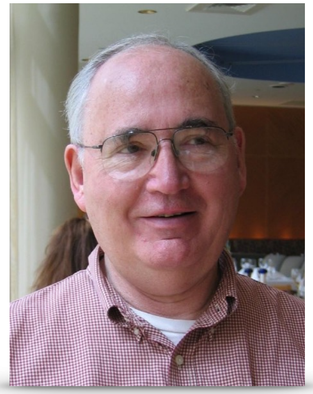


$$P\,\{Q\}\,R.$$

Historical Context in the 1970s

C. Strachey & D. Scott. 1971
A mathematical theory of computation.





$$[\![\cdot]\!] : AExp \longrightarrow (State \longrightarrow Val)$$

$$[\![\cdot]\!] : Comm \longrightarrow (State \longrightarrow State)$$

Denotational semantics was less relevant to the origins of abstract interpretation than expected

Historical Context in the 1970s

Programming Languages

T.A. Standish Editor

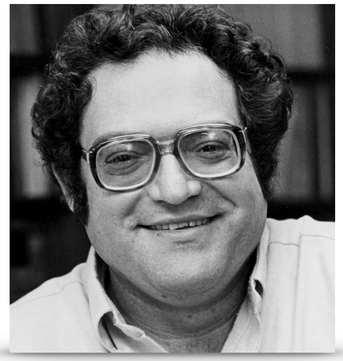
Inductive Methods for Proving Properties of Programs

Zohar Manna, Stephen Ness, Jean Vuillemin
Stanford University

There are two main purposes in this paper: first, clarification and extension of known results about computation of recursive programs, with emphasis on the difference between the theoretical and practical approaches; second, presentation and examination of various known methods for proving properties of recursive programs. Discussed in detail are two powerful inductive methods, computational induction and structural induction, including examples of their applications.

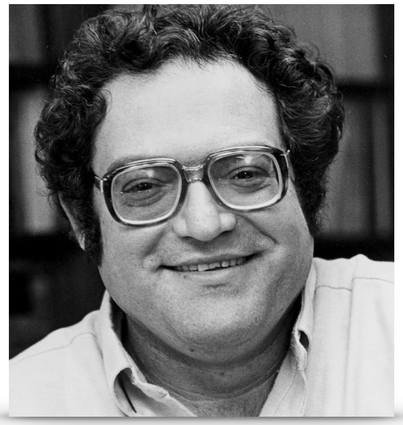
A program may consist in general of a system of recursive definitions of the form

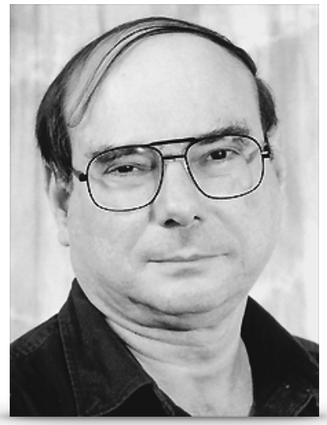
$$\begin{cases} F_1(x) \Leftarrow \tau_1[F_1, \ldots, F_n](x) \\ F_2(x) \Leftarrow \tau_2[F_1, \ldots, F_n](x) \\ \vdots \\ F_n(x) \Leftarrow \tau_n[F_1, \ldots, F_n](x), \end{cases}$$



Historical Context in the 1970s

Z. Manna & A. Pnueli 1974.
Axiomatic approach to total correctness of programs





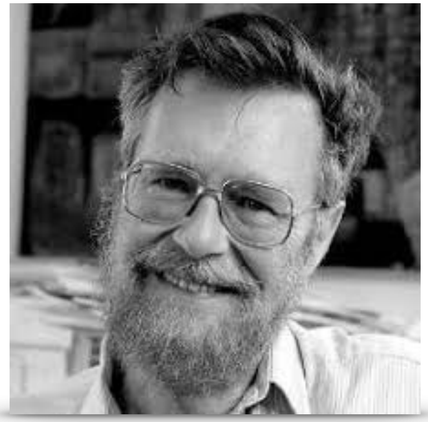
(g) **While Rule**

$$\frac{\begin{array}{l} \langle\, p(\bar{x}) \wedge t(\bar{x}) \mid B \mid q(\bar{x},\bar{x}') \wedge [\neg t(\bar{x}') \vee u(\bar{x}) \succ u(\bar{x}')] \,\rangle \\ q(\bar{x},\bar{x}') \wedge t(\bar{x}') \supset p(\bar{x}') \\ q(\bar{x},\bar{x}') \wedge q(\bar{x}',\bar{x}'') \supset q(\bar{x},\bar{x}'') \\ p(\bar{x}) \wedge \neg t(\bar{x}) \supset q(\bar{x},\bar{x}) \end{array}}{\langle\, p(\bar{x}) \mid \underline{\text{while}}\ t(\bar{x})\ \underline{\text{do}}\ B \mid q(\bar{x},\bar{x}') \wedge \neg t(\bar{x}') \,\rangle}$$

where $(w,\prec)$ is a well-founded set and $u: X \to W$.

Historical Context in the 1970s

E. Dijkstra 1975.
Guarded Commands, Nondeterminacy and Formal Derivation of Programs



Prove correctness and synthesise correct code

$$wp(\text{“}S1\ ;\ S2\text{”},\ R)\ =\ wp(S1,\ wp(S2,R))$$

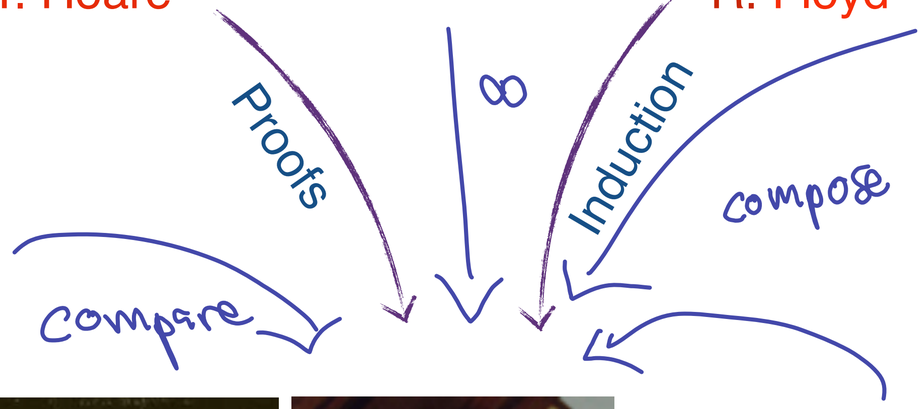
T. Hoare

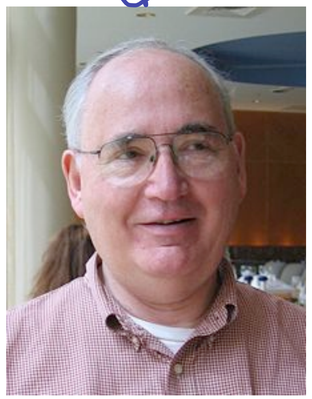
A. Turing

R. Floyd

Alfred Tarski (1902-1983)

E. Galois





D. Scott

The years 1972—1977

….. credits to Tom Ball

Early years (1972-73): Formal semantics

Patrick Cousot worked on the operational semantics of programming languages and the derivation of implementations from the formal definition.

Patrick Cousot. *Définition interprétative et implantation de languages de programmation*. Thèse de Docteur Ingénieur en Informatique, Université Joseph Fourier, Grenoble, France, 14 Décembre 1974 (submitted in 1973 but defended after finishing military service with J.D. Ichbiah at CII).

Static analysis of the formal definition and transformation to get the implementation by “pre-evaluation” (similar to the more recent “partial evaluation”)

Before starting (1972-73): formal syntax

Radhia Rezig worked on precedence parsing (R.W. Floyd, N. Wirth and H. Weber, etc.) for Algol 68

Pre-processing (by static analysis and transformation) of the grammar before building the *bottom-up* parser

Patrick Cousot worked on context-free grammar parsing (J. Earley and F. De Remer)

Pre-processing (by static analysis and transformation) of the grammar before building the *top-down* parser

---

- Radhia Rezig. *Application de la méthode de précédence totale à l'analyse d'Algol 68,* Master thesis, Université Joseph Fourier, Grenoble, France, September 1972.
- Patrick Cousot. *Un analyseur syntaxique pour grammaires hors contexte ascendant sélectif et général*. In Congrès AFCET 72, Brochure 1, pages 106-130, Grenoble, France, 6-9 November 1972.

1972

Intervals →

Assertions →

Static analysis →

pas le niveau de "compréhension" des programmes. Les langages actuels ne sont pas faits pour l'optimisation. Entre autres, il y a certains faits sur un programme qui sont connus du programmeur et qui ne sont pas explicites dans le programme. On pourrait y remédier en incluant des assertions, tout comme on insère des déclarations de type pour les variables.

Exemple :

(1) - pour i de 0 à 10 faire a[i] := i ; fin ;

(2) - pour i de 11 à 10000 faire a[i] := 0 ; fin ;

(3) - a[(a[j] + 1) x a [j + 1]] := j ;

(4) - si a[j x j + 2 x j + 1] ≠ a[j] aller à étiquette ;

Pour un tel programme, il est important de savoir que $1 \leq j < 99$ (à charge éventuellement au système de le déduire à partir d'autres assertions), parce qu'on peut alors remplacer (4) par (4') :

(4') si j < 10 aller à étiquette ;

Cette insertion d'assertions peut donc servir de guide à une analyse automatique des programmes essentielle pour l'optimisation (mais également pour la mise au point, la documentation automatique, la décompilation, l'adaptation à un changement d'environnement d'exécution...).
Dans tous les exemples que nous avons pris, (équivalence de définitions de données, équivalence de définition d'opérateurs) nous avons conduit cette analyse sémantique à la main.

La possibilité de son automation, nous semble conditionner les progrès dans le domaine de l'optimisation de l'implantation automatisée d'un langage étant donnée sa définition, aussi bien que dans celui de l'optimisation des programmes [41].

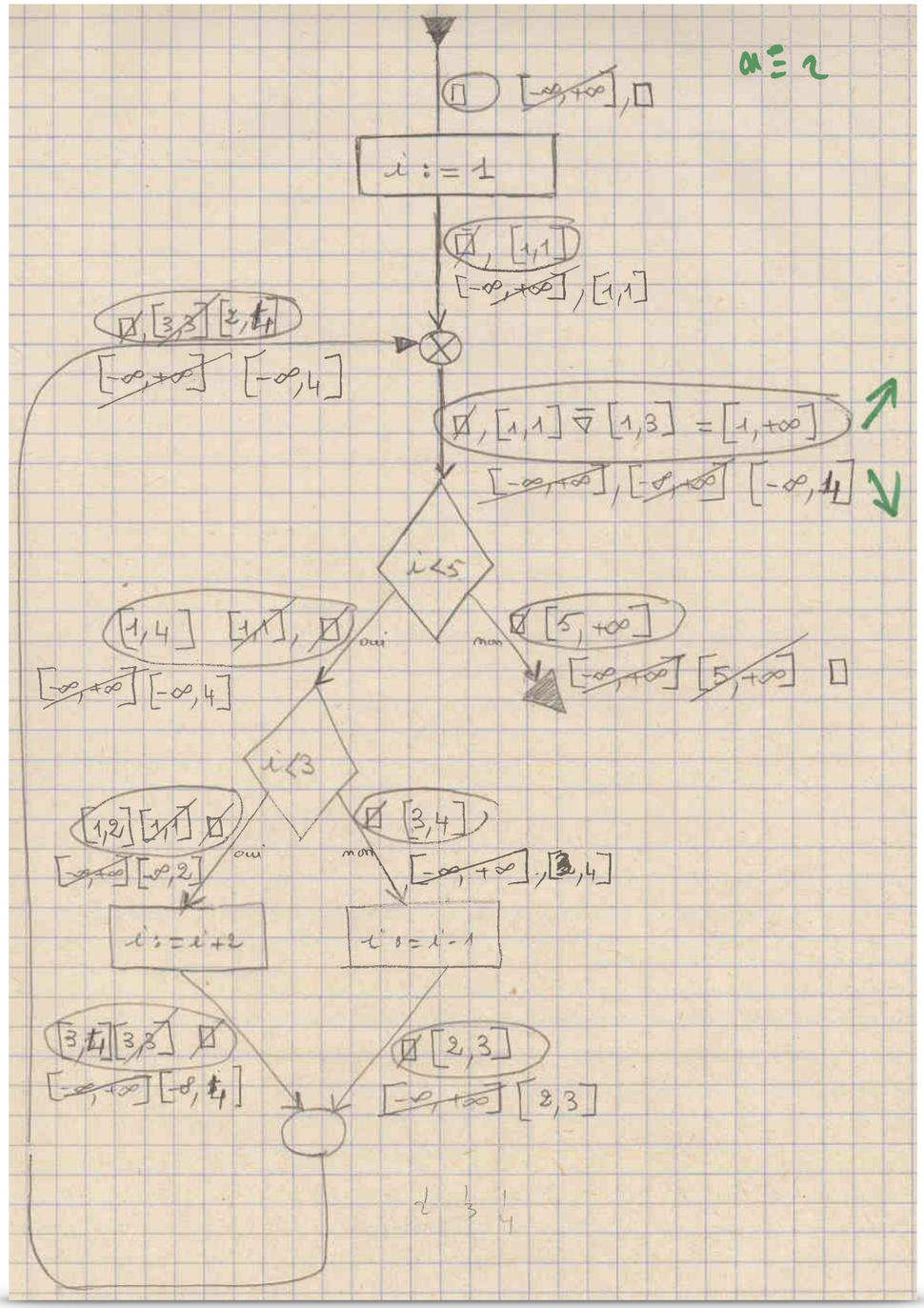
1974: The origins

Radhia Rezig shows her interval analysis ideas to Patrick Cousot

- ➡ Patrick very critical on going backwards from [-∞, +∞] and claims that going forward would be much better
- ➡ Patrick also very skeptical on forward termination for loops

Radhia comes back with the idea of **extrapolating** bounds to ±∞ for the forward analysis

The discover of **widening = induction in the abstract** and that the idea is very general!!



Notes of Radhia Rezig on **forward** iteration from $\square = \bot$ (forward least fix-point) versus **backward** iteration from $[-\infty, +\infty]$ (backward greatest fix-point)

The IRIA-SESORI contract (1975-76)

VERIFICATION STATIQUE DE LA COHERENCE

DYNAMIQUE DES PROGRAMMES

-:-:-:-:-:-:-:-:-:-:-:-:-

(1) Patrick COUSOT
(2) Radhia COUSOT

(1) - Attaché de Recherche au CNRS
(2) - Chercheur sous contrat IRIA-SESORI N° 75-035

23 Septembre 1975

VERIFICATION STATIQUE DE LA COHERENCE
DYNAMIQUE DES PROGRAMMES
-:-:-:-:-:-:-:-:-:-:-:-:-

1°) - PRESENTATION DU PROBLEME -

La notion de type ou mode dans les langages de programmation, permet générale ment une vérification statique (à la compilation), de la cohérence dynamique (à l'exécution) des programmes. Toute valeur a un mode unique, qui caractéris les actions qui peuvent être exécutées sur ou avec elle. En ALGOL 68 par exemple, les déclarations :

```
struct (int jour, string mois, int an) t = (11, "juillet", 1971) ;
struct (long bits c) u ;
```

permettent au compilateur de détecter que les écritures

```
mois of u
```

ou

```
c of t
```

sont erronées.

Pour certains types toutefois, certaines opérations sur des objets de ce type ne sont pas définies pour certaines valeurs ayant ce type. Par exemple, en PASCAL [1] on peut déclarer :

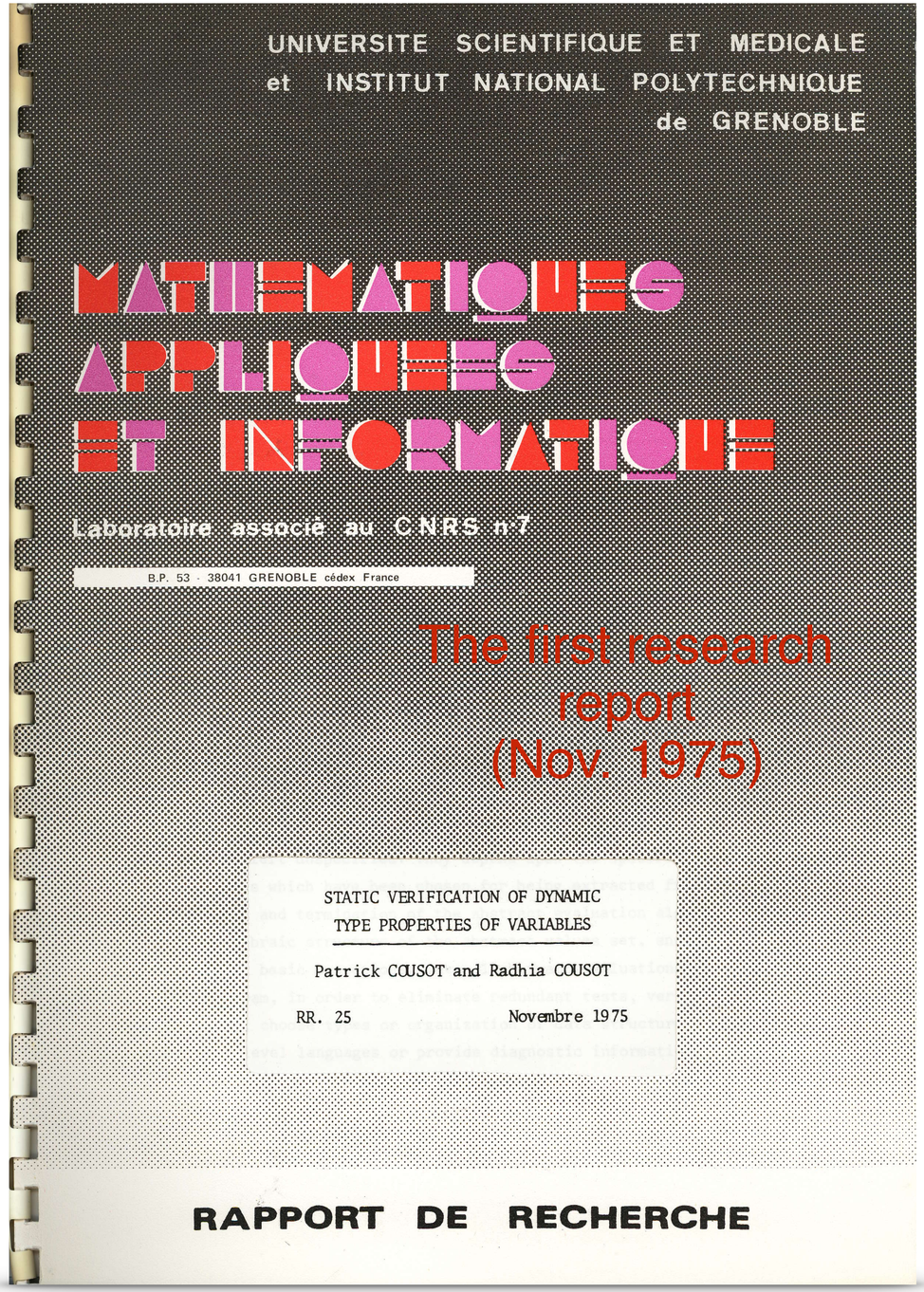
```
type personne = record
                    nom : alfa ;
                    père, mère : ↑ personne ;
                end ;
var x, y, z : ↑ personne ;
...
```

mais l'opération x ↑ • nom n'est définie que si x ≠ nil. Les techniques de compilation actuelles ne permettent pas de déterminer que x n'est pas nil quand on accède à un champ de l'enregistrement repéré par x. De ce fait, certains compilateurs* insèrent des tests dans le code généré pour détecter ce genre d'erreur à l'exécution du programme.

* La plupart des compilateurs utilisent le mécanisme de protection mémoire (à un coût pratiquement nul), mais cette technique n'est pas utilisable pour des langages d'implémentation de système, comme LIS.

The first abstract interpreter with widening (as of 23 Sep. 1975)


```
[1] procédure interprétation abstraite (graphe = (A x (N_a, N_t, N_js, N_jb, N_s, {n_e})))
[2] début
[3]   pour chaque arc de A faire contexte local (arc) := Φ refaire ;
[4]   chemins à exécuter := {arc initial (graphe)} ; jonctions := Ø ;
[5]   tantque (chemins à exécuter ≠ Ø) faire
[6]   |  tantque (chemins à exécuter ≠ Ø) faire  $sélectionner un arc à
      |  |                                                traverser$
[7]   |  |  arc := choix (chemins à exécuter) ;
[8]   |  |  chemins à exécuter := chemins à exécuter -{arc} ;
[9]   |  |  noeud traité := extrémité-finale (arc) ;
[10]  |  |  cas
[11]  |  |  | noeud traité ∈ N_a →
[12]  |  |  |    calcul contexte sortie (arc sortie(noeud traité),
      |  |  |    ∇_a(noeud traité, contexte local(arc))) ;
[13]  |  |  | noeud traité ∈ N_t →
[14]  |  |  |    (V, F) := ∇_t(noeud traité, contexte local(arc)) ;
[15]  |  |  |    calcul contexte sortie (arc sortie vrai (noeud traité), V) ;
[16]  |  |  |    calcul contexte sortie (arc sortie faux (noeud traité), F) ;
[17]  |  |  | noeud traité ∈ (N_js ∪ N_jb) → jonctions ∪ := {noeud traité} ;
[18]  |  |  | noeud traité ∈ N_s → ;
[19]  |  |  fincas ;
[20]  |  refaire ;
[21]  |     pour chaque noeud de jonctions faire
      |     |
[22]  |     |  contexte sortie := ⊔̄ contexte local (prédécesseur)
      |     |                     prédécesseur  arcs d'entrée (noeud)
[23]  |     |  si ¬(contexte sortie ⊑̄ contexte local (arc sortie(noeud)) alors
[24]  |     |  |  cas
[25]  |     |  |  | noeud ∈ N_js →
[26]  |     |  |  |    calcul contexte sortie (arc sortie (noeud),
[27]  |     |  |  |                          contexte sortie) ;
[28]  |     |  |  | noeud ∈ N_jb →
[29]  |     |  |  |    calcul contexte sortie (arc sortie(noeud),
[30]  |     |  |  |      contexte local (arc sortie(noeud)) ∇̄ contexte sortie) ;
[31]  |     |  |  fincas ;
[32]  |     |  finsi ;
[33]  |     refaire ;
[34]  |     jonction := Ø ;
[35]  refaire ;
[36]  procédure calcul contexte sortie (arc, contexte) ;
[37]  début
[38]  |  si ¬ (contexte ⊑̄ contexte local (arc)) alors
[39]  |  |  contexte local (arc) := contexte ;
[40]  |  |  chemins à exécuter ∪ := {arc} ;
[41]  |  finsi ;
[42]  fin ;
[43] fin
```



The first publication (ISOP II, Apr. 1976)

programmation

Proceedings
of the 2nd international symposium on Programming
edited by B. Robinet
Paris
April, 13-15 1976

Actes du 2e colloque international sur la programmation
direction B. Robinet
Paris
13-15 avril 1976

phase recherche **DUNOD informatique**

STATIC DETERMINATION OF DYNAMIC
PROPERTIES OF PROGRAMS

Patrick COUSOT* and Radhia COUSOT**

Université Scientifique et Médicale de Grenoble

1 - INTRODUCTION -

In high level languages, compile time type verifications are usualy incomplete, and dynamic coherence checks must be inserted in object code. For example, in PASCAL one must dynamically verify that the values assigned to subrange type variables, or index expressions lie between two bounds, or that pointers are not nil, ... We present here a general algorithm allowing most of these certifications to be done at compile time. The static analysis of programs we do consists of an abstract evaluation of these programs, similar to those used by NAUR for verifying the type of expressions in ALGOL 60 [6] , by SINTZOFF for verifying that a module corresponds to its logical specification [9] , by KILDALL for global program optimization [5] , by WEGBREIT for extracting properties of programs , [9] , by KARR for finding affine relationships among variables of a program [4] , by SCHWARTZ for automatic data structure choice in SETL [8] ,...
The essential idea is that, when doing abstract evaluation of a program, "abstract" values are associated with variables instead of the "concrete" values used while actually executing. The basic operations of the language are interpreted accordingly and the abstract interpretation then consists in a transitive closure mechanism. One may consider abstract values belonging to no finite sets, but the properties of the transitive closure algorithm are chosen such that the abstract interpretation stabilizes after finitely many steps,

---

* Attaché de Recherche au CNRS, Laboratoire Associé N°7.
** This work was supported by IRIA-SESORI under grant 75-035.

The breakthrough!

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

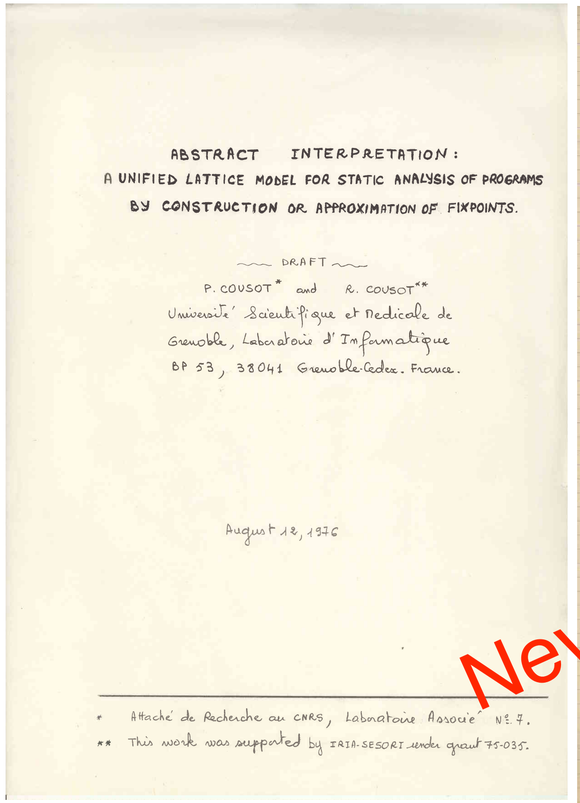
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

POPL 1977
Los Angeles, January 17-19, 1977

The POPL 1977 submission

For 4th POPL'77, Patrick and Radhia submitted on August 12, 1976 hard copies of a two-hands written manuscript of **100 pages**.



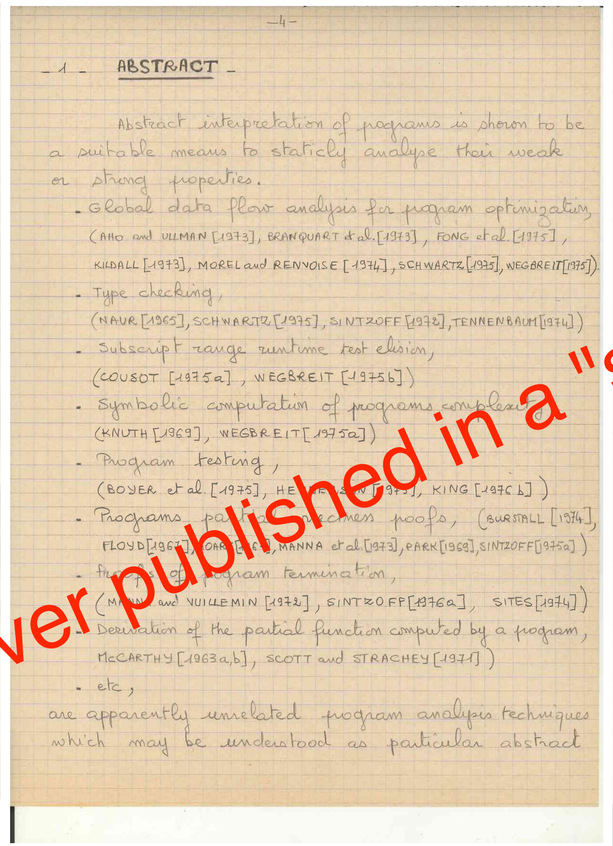
ABSTRACT INTERPRETATION :
A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS
BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS.

~~ DRAFT ~~

P. COUSOT* and R. COUSOT**
Université Scientifique et Medicale de
Grenoble, Laboratoire d'Informatique
BP 53, 38041 Grenoble-Cedex. France.

August 12, 1976

* Attaché de Recherche au CNRS, Laboratoire Associé N°. 7.
** This work was supported by IRIA-SESORI under grant 75-035.



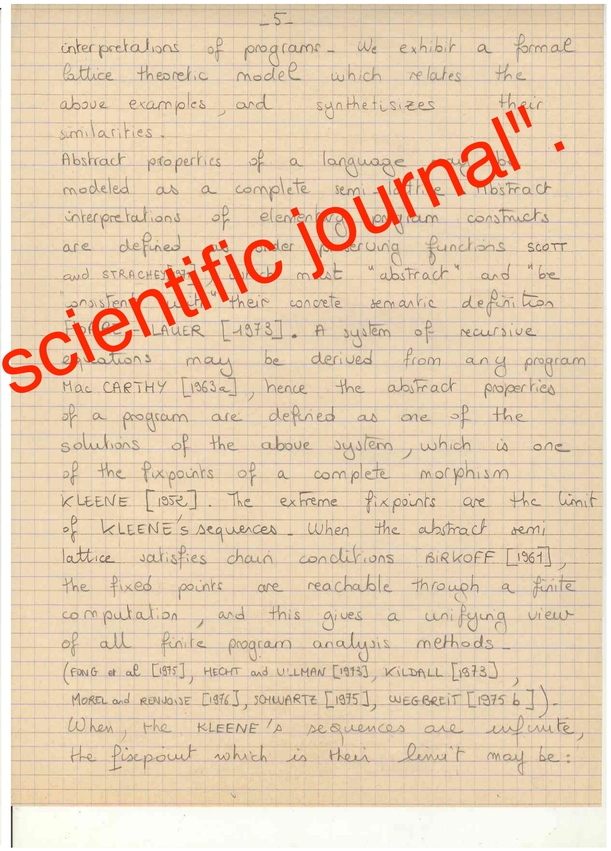
-4-

-1- ABSTRACT -

Abstract interpretation of programs is shown to be a suitable means to staticly analyse their weak or strong properties.

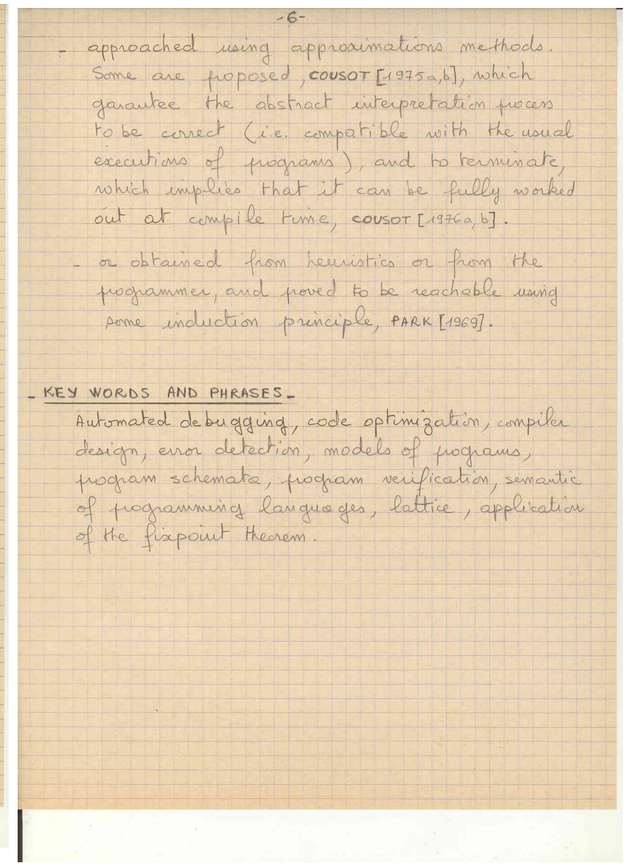
- Global data flow analysis for program optimization, (AHO and ULLMAN [1973], BRANQUART et al. [1975], FONG et al. [1975], KILDALL [1973], MOREL and RENVOISE [1974], SCHWARTZ [1975], WEGBREIT [1975])
- Type checking, (NAUR [1965], SCHWARTZ [1975], SINTZOFF [1972], TENNENBAUM [1974])
- Subscript range runtime test elision, (COUSOT [1975a], WEGBREIT [1975b])
- Symbolic computation of programs complexity (KNUTH [1969], WEGBREIT [1975a])
- Program testing, (BOYER et al. [1975], HE[illegible] [1975], KING [1975b])
- Programs partial correctness proofs, (BURSTALL [1974], FLOYD [1967], [illegible], MANNA et al. [1973], PARK [1969], SINTZOFF [1975a])
- [illegible] of program termination, (MANNA and VUILLEMIN [1972], SINTZOFF [1976a], SITES [1974])
- Derivation of the partial function computed by a program, McCARTHY [1963a,b], SCOTT and STRACHEY [1971])
- etc,

are apparently unrelated program analysis techniques which may be understood as particular abstract



-5-

interpretations of programs. We exhibit a formal lattice theoretic model which relates the above examples, and synthetisizes their similarities.
Abstract properties of a language [illegible] modeled as a complete semi-[illegible] the abstract interpretations of elementary program constructs are defined [illegible] order preserving functions SCOTT and STRACHEY [1971] must "abstract" and "be consistent" with their concrete semantic definition [illegible] LAUER [1973]. A system of recursive equations may be derived from any program Mac CARTHY [1963a], hence the abstract properties of a program are defined as one of the solutions of the above system, which is one of the fixpoints of a complete morphism KLEENE [1952]. The extreme fixpoints are the limit of KLEENE's sequences. When the abstract semi lattice satisfies chain conditions BIRKOFF [1961], the fixed points are reachable through a finite computation, and this gives a unifying view of all finite program analysis methods. (FONG et al. [1975], HECHT and ULLMAN [1973], KILDALL [1973], MOREL and RENVOISE [1976], SCHWARTZ [1975], WEGBREIT [1975 b]). When the KLEENE's sequences are infinite, the fixpoint which is their limit may be:



-6-

- approached using approximations methods. Some are proposed, COUSOT [1975a,b], which garantee the abstract interpretation process to be correct (i.e. compatible with the usual executions of programs), and to terminate, which implies that it can be fully worked out at compile time, COUSOT [1976a,b].
- or obtained from heuristics or from the programmer, and proved to be reachable using some induction principle, PARK [1969].

- KEY WORDS AND PHRASES -
Automated debugging, code optimization, compiler design, error detection, models of programs, program schemata, program verification, semantic of programming languages, lattice, application of the fixpoint theorem.

Never published in a "scientific journal".

_0_ INTRODUCTION

A program denotes (complicated) computations in some universe of objects. The idea of abstract interpretation of programs, is to use that denotation to describe (simple) computations in another universe of abstract objects with the hope that the results of abstract interpretation will give us some informations on the actual computations.

An intuitive example of abstract interpretation, is the rule of signs which we borrow from SINTZOFF [1972]. The Text :

```
- 1515 * 17
```

denotes the multiplication of integers, but it can be understood to denote computations on the universe $\{(+), (-), (\pm)\}$ where the semantics of operators +, *, ... are defined by the rule of signs. Our above example leads to the following reductions

```
   - 1515  *  17
=>   -(+)  *  (+)
=>    (-)  *  (+)
=>    (-)
```

which state that the result of -1515 * 17

rule of signs which we borrow from SINTZOFF [1972]

The text :

$$-\ 15\ 15 \quad * \quad 17$$

denotes the multiplication of integers, but it can be understood to denote computations on the universe $\{(+), (-), (\pm)\}$ where the semantics of operators $+, *, \ldots$ are defined by the rule of signs. Our above example leads to the following reductions

$$\begin{aligned} & -\ 15\ 15 \quad * \quad 17 \\ \Rightarrow \quad & -\ (+) \quad * \quad (+) \\ \Rightarrow \quad & (-) \quad * \quad (+) \\ \Rightarrow \quad & (-) \end{aligned}$$

which state that the result of $-1515 * 17$

- 1 - ABSTRACT -

Abstract interpretation of programs is shown to be a suitable means to staticly analyse their weak or strong properties.

- Global data flow analysis for program optimization, (AHO and ULLMAN [1973], BRANQUART et al. [1973], FONG et al. [1975], KILDALL [1973], MOREL and RENVOISE [1974], SCHWARTZ [1975], WEGBREIT [1975]),
- Type checking, (NAUR [1965], SCHWARTZ [1975], SINTZOFF [1972], TENNENBAUM [1974])
- Subscript range runtime test elision, (COUSOT [1975a], WEGBREIT [1975b])
- Symbolic computation of programs complexity, (KNUTH [1969], WEGBREIT [1975a])
- Program testing, (BOYER et al. [1975], HENDERSON [1975], KING [1976b])
- Programs partial correctness proofs, (BURSTALL [1974], FLOYD [1967], HOARE [1967], MANNA et al. [1973], PARK [1969], SINTZOFF [1975a])
- Proofs of program termination, (MANNA and VUILLEMIN [1972], SINTZOFF [1976a], SITES [1974])
- Derivation of the partial function computed by a program, McCARTHY [1963a,b], SCOTT and STRACHEY [1971])
- etc,

are apparently unrelated program analysis techniques which may be understood as particular abstract

- KEY WORDS AND PHRASES -

Automated debugging, code optimization, compiler design, error detection, models of programs, program schemata, program verification, semantic of programming languages, lattice, application of the fixpoint theorem.

Example

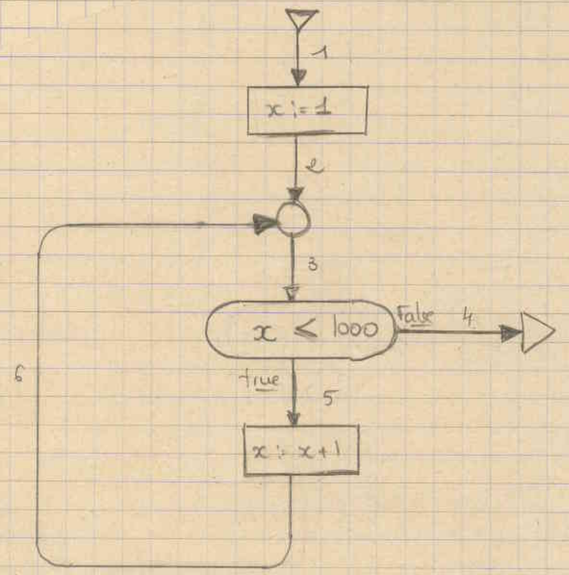


fig. 2.1.a

Example :

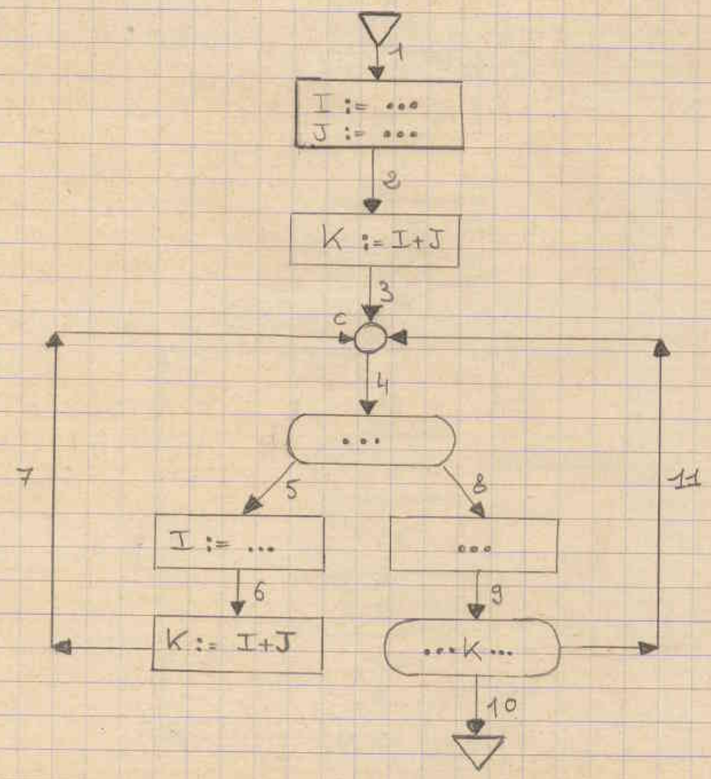
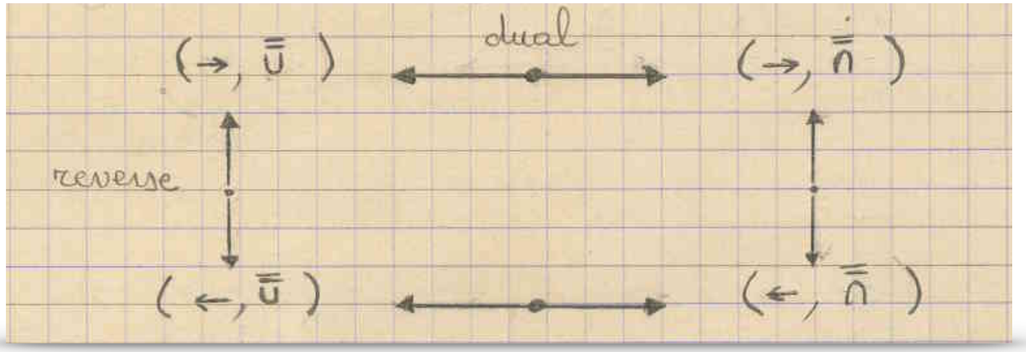


fig 2.4.1.b

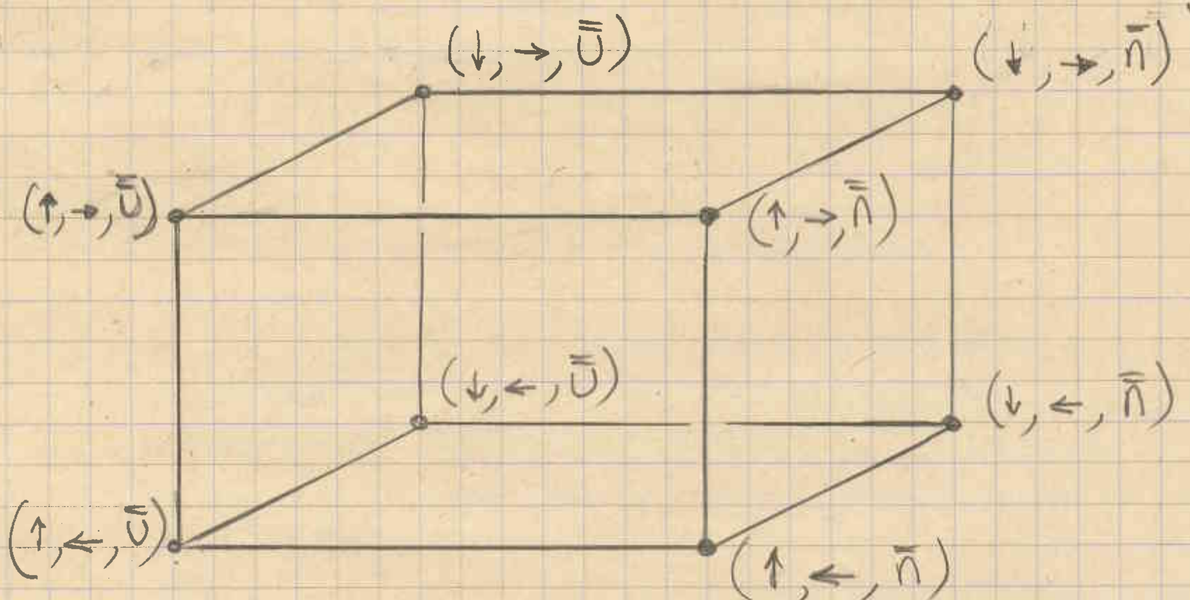
3.3.2 - Extreme fixed points existence -

The definition of a global program property as some of the multiple solutions to a system of equations is somewhat too inaccurate - Fortunately the fixed points form a complete semi lattice relative to the ordering $\overline{\leq}_R$ - (TARSKI [1955]) - There is a greatest fixpoint -

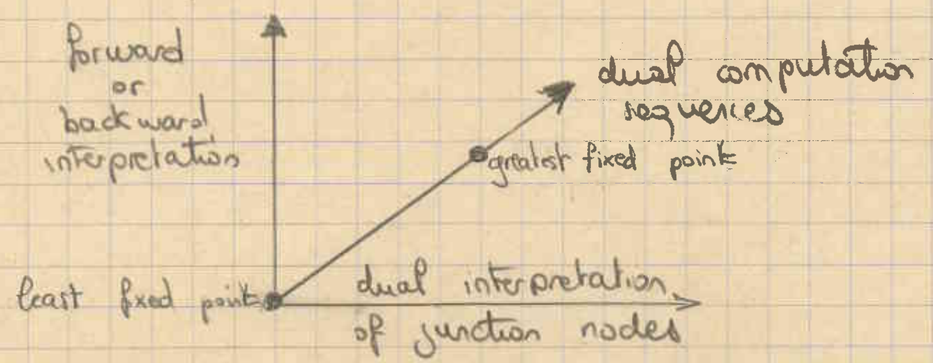


Summary

It is shown that a system of equations on the space of partial functions (SCOTT and STRACHEY [1971]) may be associated to any program, which fixpoint is the "meaning" of the program. (MANNA et al. [1973]).



These alternatives are related by :



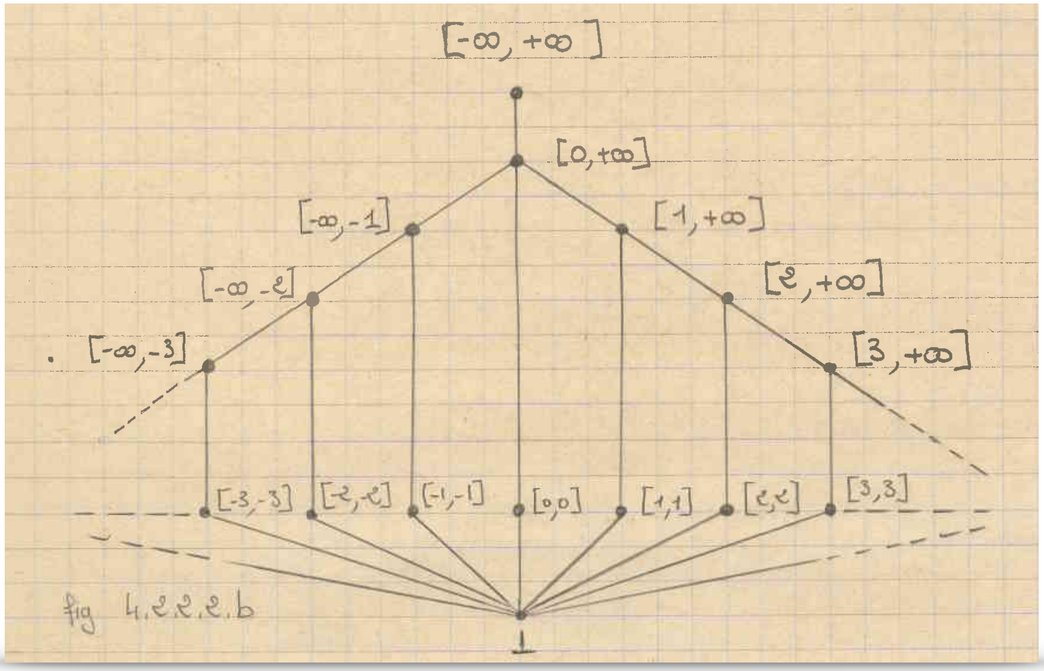


fig 4.2.2.2.b

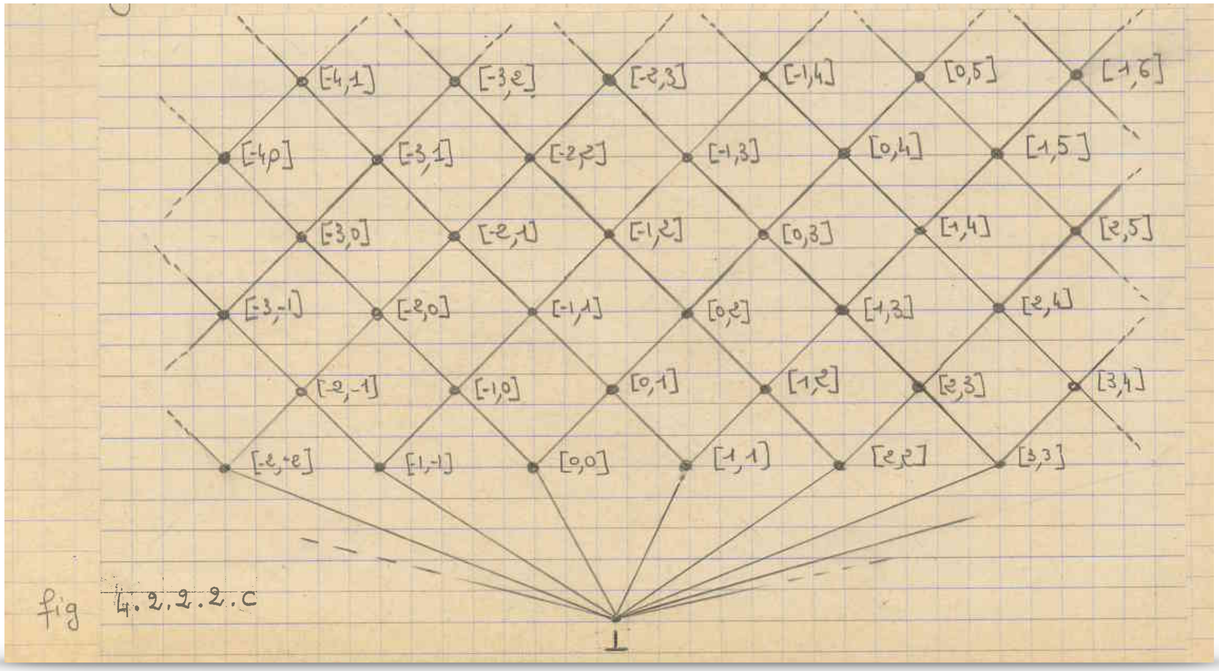
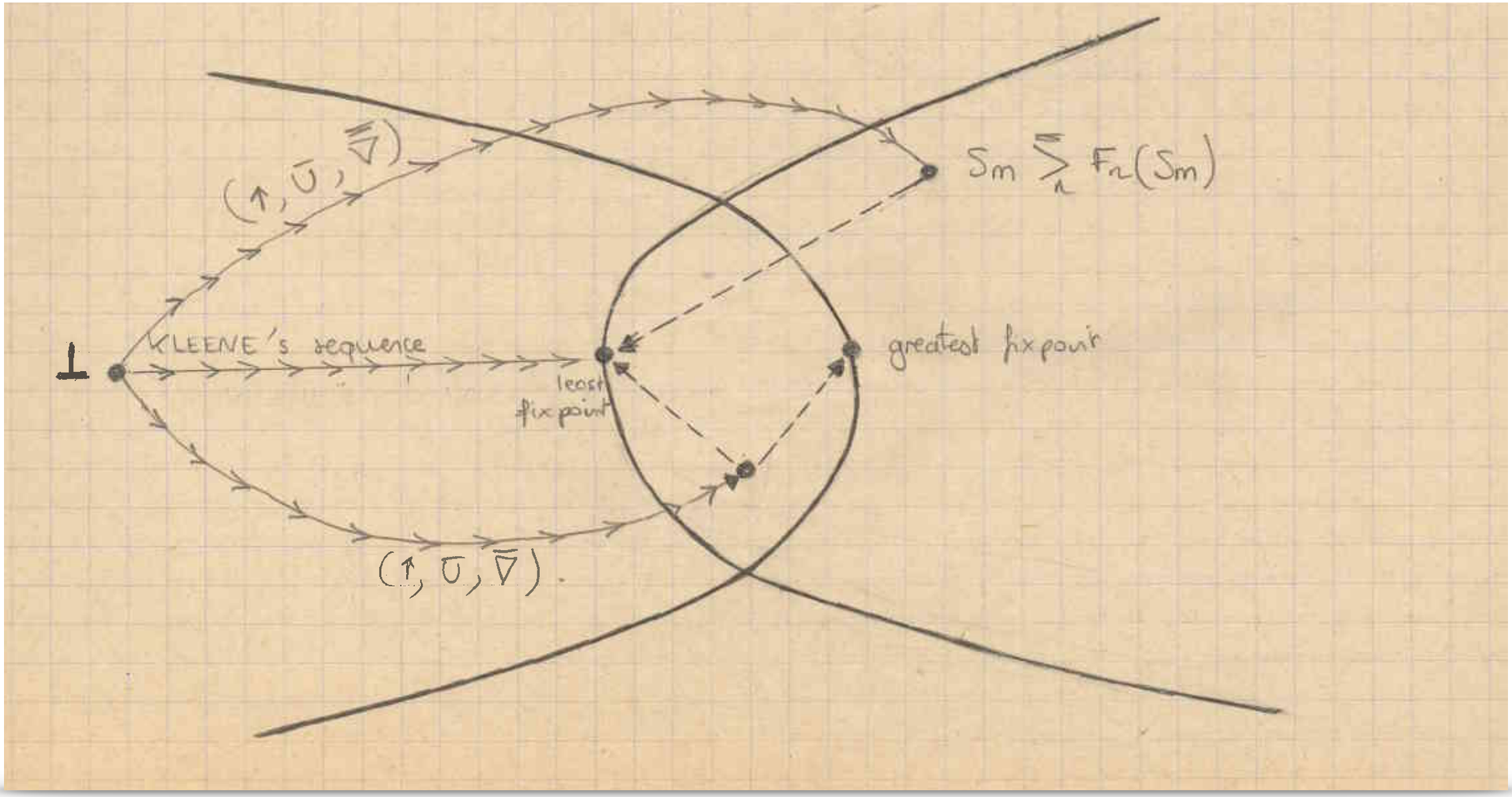


fig 4.2.2.2.c

— In order to built approximation sequences, we define the widening $\bar{\nabla}$ such that :

- $[,] \bar{\nabla} \delta = \delta \bar{\nabla} [,] = \delta$
- $[i,j] \bar{\nabla} [k,\ell] =$
  $[$ if $k < i$ then $-\infty$ else i fi ,
  if $\ell > j$ then $+\infty$ else j fi $]$


$(\uparrow, \cup, \overline{\overline{\nabla}})$
$Sm \overline{\overline{\geq}} F_n(Sm)$
$\perp$
KLEENE's sequence
least fixpoint
greatest fixpoint
$(\uparrow, \cup, \overline{\nabla})$

- 5 - CONCLUDING REMARKS -

We did not covered all language features of the present days programming languages, nor tried to enumerate the one thousand and one possible applications of that work. However we think that this work might influence compiler design and implementation techniques, (especially code optimization), semantics of programming languages which should provide the way to define abstract models of programs (especially for program specification and verification techniques), programming language design since abstract interpretation of programs should be at the service of the programmer (like present data type which should be enlarged to give new error detection facilities).

Everything was there!

3.1.3 L'approche du point fixe à l'étude du comportement d'un système dynamique discret

i.e. pre

DEFINITION 3.1.3.0.1

$wp \in (((S\times S)\to B)\to((S\to B)\to(S\to B)))$

$= \lambda\theta.\{\lambda\beta.[\lambda e_1.(\exists e_2 \in S : \theta(e_1,e_2) \text{ et } \beta(e_2))]\}$

i.e. post transformer

DEFINITION 3.1.3.0.2

$sp \in (((S\times S)\to B)\to((S\to B)\to(S\to B)))$

$= \lambda\theta.\{\lambda\beta.[\lambda e_2.(\exists e_1 \in S : \beta(e_1) \text{ et } \theta(e_1,e_2))]\}$

Partant du fait que $\tau^* = eq \text{ ou } \tau^*\circ\tau = eq \text{ ou } \tau\circ\tau^*$, nous obtiendrons $wp(\tau^*)$ et $sp(\tau^*)$ comme points fixes d'une équation.

THEOREME 3.1.3.0.3

(a) - $((S\times S)\to B)(\Rightarrow, \lambda(e_1,e_2).faux, \lambda(e_1,e_2).vrai, OU, ET, non)$ est un treillis booléen complet,

(b) - Soient $a, b \in ((S\times S)\to B)$ alors $\lambda\alpha.[a \text{ ou } b\circ\alpha]$ et $\lambda\alpha.[a \text{ ou } \alpha\circ b]$ sont des morphismes complets pour la disjonction,

(c) - Soient $\tau \in ((S\times S)\to B)$ et eq la relation d'égalité alors $\tau^* = lfp(\lambda\alpha.[eq \text{ ou } \alpha\circ\tau]) = lfp(\lambda\alpha.[eq \text{ ou } \tau\circ\alpha])$.

fixpoint reflexive transitive closure

abstract transformer

concrete transformer

Preuve: Posons $h=\lambda\theta.[wp(\theta)(\beta)]$, $f=\lambda\alpha.[a \text{ ou } b\circ\alpha]$ et $g = \lambda\alpha.[wp(a)(\beta) \text{ ou } wp(b)(\alpha)]$ et montrons que $h\circ f = g\circ h$.

• • •

THEOREME 3.1.3.0.6.

Quels que soient $a, b \in ((S\times S)\to B)$ et $\beta \in (S\to B)$ nous avons:

- $wp(lfp(\lambda\alpha.[a \text{ ou } b\circ\alpha]))(\beta)$
  $= lfp(\lambda\alpha.[wp(a)(\beta) \text{ ou } wp(b)(\alpha)])$
  $= \underset{n\in\omega}{OU}\, wp(b^n)(wp(a)(\beta))$
- $sp(lfp(\lambda\alpha.[a \text{ ou } \alpha\circ b]))(\beta)$
  $= lfp(\lambda\alpha.[sp(a)(\beta) \text{ ou } sp(b)(\alpha)])$
  $= \underset{n\in\omega}{OU}\, sp(b^n)(sp(a)(\beta))$

fixpoint
backward reachability
forward reachability

iterative fixpoint computation

Fixpoint abstraction under commutativity with abstraction h

POPL1977

This implies that A-Cont is in fact a complete lattice, but we need only one of the two join and meet operations. The set of context vectors is defined by $\widehat{\text{A-Cont}} = \text{Arcs}^0 \to \text{A-Cont}$. Whatever (Cv', Cv") ∈ $\widehat{\text{A-Cont}}^2$ may be, we define :

$$Cv' \;\tilde{\cup}\; Cv" = \lambda r \,.\, Cv'(r) \cup Cv"(r)$$

$$Cv' \;\tilde{\leq}\; Cv" = \{\forall r \in \text{Arcs}^0,\ Cv'(r) \leq Cv"(r)\}$$

$\tilde{\top} = \lambda r \,.\, \top$ and $\tilde{\bot} = \lambda r \,.\, \bot$

$\langle \widehat{\text{A-Cont}}, \tilde{\cup}, \tilde{\cap}, \tilde{\leq}, \tilde{\top}, \tilde{\bot} \rangle$ can be shown to be a complete lattice. The function :

$$\underline{\text{Int}} : \text{Arcs}^0 \times \widehat{\text{A-Cont}} \to \text{A-Cont}$$

defines the interpretation of basic instructions. If {C(q) | q ∈ a-pred(n)} is the set of input contexts of node n, then the output context on exit arc r of n (r ∈ a-succ(n)) is equal to Int(r, C). Int is supposed to be order-preserving :

$$\forall a \in \text{Arcs},\ \forall (Cv', Cv") \in \widehat{\text{A-Cont}}^2,$$
$$\{Cv' \;\tilde{\leq}\; Cv"\} \Longrightarrow \{\underline{\text{Int}}(a, Cv') \leq \underline{\text{Int}}(a, Cv")\}$$

The local interpretation of elementary program constructs which is defined by Int is used to associate a system of equations with the program. We define

$$\widehat{\underline{\text{Int}}} : \widehat{\text{A-Cont}} \to \widehat{\text{A-Cont}} \mid \widehat{\underline{\text{Int}}}(Cv) = \lambda r \,.\, \underline{\text{Int}}(r, Cv)$$

It is easy to show that $\widehat{\text{Int}}$ is order-preserving. Hence it has fixpoints, Tarski[55]. Therefore the context vector resulting from the abstract interpretation I of program P, which defines the global properties of P, may be chosen to be one of the extreme solutions to the system of equations $Cv = \widehat{\text{Int}}(Cv)$.

5.2 Typology of Abstract Interpretations

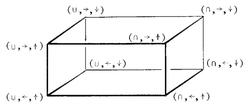
The restriction that "A-Cont" must be a complete semi-lattice is not drastic since Mac Neille[37] showed that any partly ordered set S can be embedded in a complete lattice so that inclusion is preserved, together with all greatest lower bounds and lowest upper bounds existing in S. Hence in practice the set of abstract contexts will be a lattice, which can be considered as a join (∪) semi-lattice or a meet (∩) semi-lattice, thus giving rise to two dual abstract interpretations.

It is a pure coincidence that in most examples (see 5.3.2) the ∩ or ∪ operator represents the effect of path converging. The real need for this operator is to define completeness which ensures $\widehat{\text{Int}}$ to have extreme fixpoints (see 8.4).

The result of an abstract interpretation was defined as a solution to forward (÷) equations : the output contexts on exit arcs of node n are defined as a function of the input contexts on entry arcs of node n. One can as well consider a system of backward (÷) equations : a context may be related to its successors. Both systems (÷, ÷) may also be combined.

Finally we usually consider a maximal (↑) or minimal (↓) solution to the system of equations, (by agreement, maximal and minimal are related to the ordering ≤ defined by (x ≤ y) <⟹> (x ∪ y = y) <⟹> (x ∩ y = x)). However known examples such as Manna and Shamir[75] show that the suitable solution may be somewhere between the extreme ones.

These choices give rise to the following types of abstract interpretations :



Examples :

Kildall[73] uses (∩,→,↑). Wegbreit[75] uses (∪,→,↓). Tenenbaum[74] uses both (∪,→,↓) and (∩,←,↑).

5.3 Examples

5.3.1 Static Semantics of Programs

The static semantics of programs we defined in section 4 is an abstract interpretation :

$$I_{SS} = \langle \text{Contexts}, \cup, \subseteq, \text{Env}, \emptyset, \underline{\text{n-context}} \rangle$$

where Contexts, ∪, ⊆, Env, ∅, n-context, Context-Vectors, ∪̃, ⊆̃, P-Cont respectively correspond to A-Cont, ∪, ⊆, T, ⊥, Int, $\widehat{\text{A-Cont}}$, ∪̃, ⊆̃, $\widehat{\text{Int}}$.

5.3.2 Data Flow Analysis

Data flow analysis problems (see references in Ullman[75]) may be formalized as abstract interpretations of programs.

"Available expressions" give a classical example. An expression is available on arc r, if whenever control reaches r, the value of the expression has been previously computed, and since the last computation of the expression, no argument of the expression has had its value changed.

Let Expr_P be the set of expressions occuring in a program P. Abstract contexts will be sets of available expressions, represented by boolean vectors :

$$\text{B-vect} : \text{Expr}_P \to \{\underline{\text{true}}, \underline{\text{false}}\}$$

B-vect is clearly a complete boolean lattice. The interpretation of basic nodes is defined by :

```
avail(r, Bv)
  let n be origin(r) within
    case n in
      Entries ⟹> λe . false
      Assignments ∪ Tests ∪ Junctions ⟹>
        λe.(generated(n)(e) or ((  and   Bv(p)(e))
                                 p∈a-pred(n)
                               and transparent(n)(e)))
    esac
```

(Nothing is available on entry arcs. An expression e is available on arc r (exit of node n) if either the expression e is generated by n or for all predecessors p of n, e is available on p and n does not modify arguments of e).

The available expressions are determined by the maximal solution (for ordering λe . false ⊂̃ λe . true) of the system of equations :

$$\underline{Bv} = \widehat{\underline{\text{avail}}}(\underline{Bv})$$

FDPC1977

Formal Descriptions of Programming Concepts, E.J. Neuhold (ed.)
North-Holland Publishing Company, (1978)

STATIC DETERMINATION OF DYNAMIC PROPERTIES OF RECURSIVE PROCEDURES

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP.53
38041 Grenoble cedex, France

1. INTRODUCTION

We present a general technique for determining properties of recursive procedures. For example, a mechanized analysis of the procedure reverse can show that whenever L is a non-empty linked linear list then reverse(L) is a non-empty linked linear list which shares no elements with L. This information about reverse approximates the fact that reverse(L) is a reversed copy of L.

In section 2, we introduce ⊔-topological lattices that is complete lattices endowed with a ⊔-topology. The continuity of functions is characterized in this topology and fixed point theorems are recalled in this context.

The semantics of recursive procedures is defined by a predicate transformer associated with the procedure. This predicate transformer is the least fixed point of a system of functional equations (§3.2) associated with the procedure by a syntactic algorithm (§3.1).

In section 4, we study the mechanized discovery of approximate properties of recursive procedures. The notion of approximation of a semantic property is introduced by means of a closure operator on the ⊔-topological lattice of predicates. Several characterizations of closure operations are given which can be used in practice to define the approximate properties of interest (§4.1.1). The lattice of closure operators induces a hierarchy of program analyses according to their fineness. Combinations of different analyses of programs are studied (§4.1.2). A closure operator defined on the semantic ⊔-topological space induces a relative

---

* Attaché de Recherche au C.N.R.S., Laboratoire Associé n° 7.

** This work was supported by I.R.I.A.- S.E.S.O.R.I. under grant 76-160.

POPL1979

THEOREM 6.4.0.2

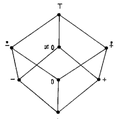
(1) - Let I be a principal ideal and J be a dual ideal of a complete lattice L(E,⊥,⊤,⊔,⊓). If InJ is nonvoid then InJ is a complete and convex sub-join-semilattice of L.
(2) - Every complete and convex sub-join-semilattice C of L can be expressed in this form with I = {x∈L : x≤ (⊔C)} and {x∈L : (∃y∈C : y≤ x)} ≤ J.

THEOREM 6.4.0.3

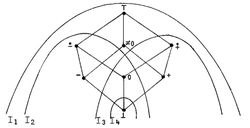
Let $\{I_i \in \Delta\}$ be a family of principal ideals of the complete lattice L(E,⊥,⊤,⊔,⊓) containing L. Then $\lambda x.\sqcup\{n I_i : i \in \Delta \wedge x \in I_i\}$ is an upper closure operator on L.

Example 6.4.0.4

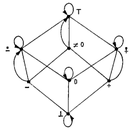
The following lattice can be used for static analysis of the signs of values of numerical variables :



(where ⊥, -, +, ±, ≠0, ∓, ⊤ respectively stand for λx.*false*, λx.x<0, λx.x>0, λx.x≤0, λx.x≠0, λx.x≥0, λx.*true*). A further approximation can be defined by the following family of principal ideals :



which induces an upper closure operator ρ :



and the space of approximate assertions (used in example 5.2.0.5)

End of Example.

7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to A and Y the specification of a program analysis framework also includes the choice of an approximate predicate transformer $\tau\in(L\to(A\to A))$ (or a monoid of maps on A plus a rule for associating maps to program statements (e.g. Rosen[78])). We now show that in fact this is not indispensable since there exists a best correct choice of τ which is induced by $\overline{A}$ and the formal semantics of the considered programming language.

7.1 A Reasonable Definition of Correct Approximate Predicate Transformers

At paragraph 3, given (V,A,τ) the minimal assertion which is invariant at point i of a program π with entry specification $\phi\in\overline{A}$ was defined as :

$$P_i = \bigvee_{p \in path(i)} \overline{\tau}(p)(\phi)$$

Therefore the minimal approximate invariant assertion is the least upper approximation of P_i in $\overline{A}$ that is :

$$\rho(P_i) = \rho\Big(\bigvee_{p \in path(i)} \overline{\tau}(p)(\phi)\Big)$$

Even when path(i) is a finite set of finite paths the evaluation of $\overline{\tau}(p)(\phi)$ is hardly machine-implementable since for each path $p = \alpha_1,\ldots,\alpha_m$ the computation sequence $X_0 = \phi,\ X_1 = \tau(C(\alpha_1))(X_0),\ \ldots,\ X_m = \tau(C(\alpha_m))(X_{m-1})$ does not necessarily only involve elements of $\overline{A}$ and $(\overline{A}\to\overline{A})$. Therefore using $\phi\in\overline{A}$ and $\tau\in(L\to(\overline{A}\to\overline{A}))$ a machine representable sequence $\overline{X}_0=\phi,\ \overline{X}_1=\overline{\tau}(C(\alpha_1))(\overline{X}_0),\ \ldots,\ \overline{X}_m=\overline{\tau}(C(\alpha_m))(\overline{X}_{m-1})$ is used instead of $X_0,\ldots,X_m$ which leads to the expression :

$$Q_i = \rho\Big(\bigvee_{p \in path(i)} \overline{\tau}(p)(\phi)\Big)$$

The choice of $\overline{\tau}$ and $\overline{\phi}$ is correct if and only if Q_i is an upper approximation of P_i in $\overline{A}$ that is if and only if :

$$\Big(\bigvee_{p \in path(i)} \tilde{\tau}(p)(\phi)\Big) \Longrightarrow \rho\Big(\bigvee_{p \in path(i)} \overline{\tau}(p)(\overline{\phi})\Big)$$

In particular for the entry point we must have $\phi \Longrightarrow \rho(\overline{\phi})=\overline{\phi}$ so that we can state the following :

DEFINITION 7.1.0.1

(1) - An approximate predicate transformer $\overline{\tau}\in(L\to(\overline{A}\to\overline{A}))$ is said to be a *correct upper approximation* of $\tau\in(L\to(A\to A))$ in $\overline{A}\wr\rho(A)$ if and only if for all $\phi\in A$, $\overline{\phi}\in\overline{A}$ such that $\phi \Longrightarrow \overline{\phi}$ and program π we have : $MOP_\pi(\tau,\phi) \Longrightarrow MOP_\pi(\overline{\tau},\overline{\phi})$
(2) - Similarly if $A\triangleright\!\!<\alpha,\gamma>A$, $\tau\in(L\to(A\to A))$ is said to be a *correct upper approximation* of $\tau\in(L\to(A\to A))$ in $A\triangleright\!\!<\alpha(A)$ if and only if $\forall\phi$, $\forall\overline{\phi}$: $\phi \Longrightarrow \gamma(\overline{\phi})$, $\forall\pi$, $\alpha(MOP_\pi(\tau,\phi)) \leq MOP_\pi(\overline{\tau},\overline{\phi})$, (i.e. $MOP_\pi(\tau,\phi) \Longrightarrow \gamma(MOP_\pi(\overline{\tau},\overline{\phi}))$)

This global correctness condition for $\overline{\tau}$ is very difficult to check since for any program π and any program point i all paths p∈path(i) must be considered. However it is possible to use instead the following equivalent local condition which can be checked for every type of statements :

On this page: dual, conjugate and inversion: **lfp/gfp** wp/sp (i.e. pre/post) wp/sp)

Topology, higher-order fixpoints, operational/ summary/... analysis

Galois connections, **closure operators**, Moore families, ideals,...

Key Ideas!

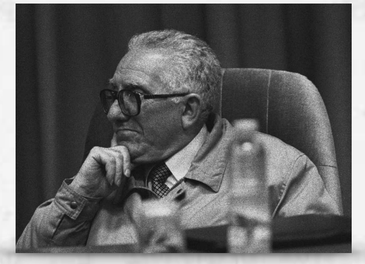
Program analysis is approximating program semantics

Formalise the notion of approximation in Semantics

Domain approximation

Fix-point approximation

PORTUGALIAE MATHEMATICA
Vol. 38 Fasc. 1-2 — 1979



A CONSTRUCTIVE CHARACTERIZATION OF THE LATTICES OF ALL RETRACTIONS, PRECLOSURE, QUASI-CLOSURE AND CLOSURE OPERATORS ON A COMPLETE LATTICE

BY

PATRICK COUSOT AND RADHIA COUSOT *

Université de Metz
Faculté des Sciences
Ile du Saulcy
5700 Metz — França

PACIFIC JOURNAL OF MATHEMATICS
Vol. 82, No. 1, 1979

CONSTRUCTIVE VERSIONS OF TARSKI'S FIXED POINT THEOREMS

PATRICK COUSOT AND RADHIA COUSOT

Let F be a monotone operator on the complete lattice L into itself. Tarski's lattice theoretical fixed point theorem states that the set of fixed points of F is a nonempty complete lattice for the ordering of L. We give a constructive proof of this theorem showing that the set of fixed points of F is the image of L by a lower and an upper preclosure operator. These preclosure operators are the composition of lower and upper closure operators which are defined by means of limits of stationary transfinite iteration sequences for F. In the same way we give a constructive characterization of the set of common fixed points of a family of commuting operators. Finally we examine some consequences of additional semi-continuity hypotheses.

The community

POPL1977 POPL1979 POPL1992 POPL1997 POPL2000

POPL2002 POPL2004 POPL2011 POPL2012 POPL2014 POPL2017

POPL2019

POPL subject areas

Compilers Formal languages and automata theory Formal software verification Lambda calculus Language features Language types Logic Program reasoning Program semantics Program verification Semantics and reasoning Software development process management Software verification Type structures Verification

Bibliometrics: publication history

Publication years	1973-2019
Publication count	1,991
Citation Count	54,165
Available for download	1,838
Downloads (6 Weeks)	4,329
Downloads (12 Months)	35,907
Downloads (cumulative)	800,815
Average downloads per article	435.70
Average citations per article	27.20

Top 10 Downloaded Articles (past 6 weeks) / Top 10 Most Cited Articles

Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints - 1977
Patrick Cousot, Radhia Cousot
Cited 2128 times

Proof-carrying code - 1997
George C. Necula
Cited 678 times

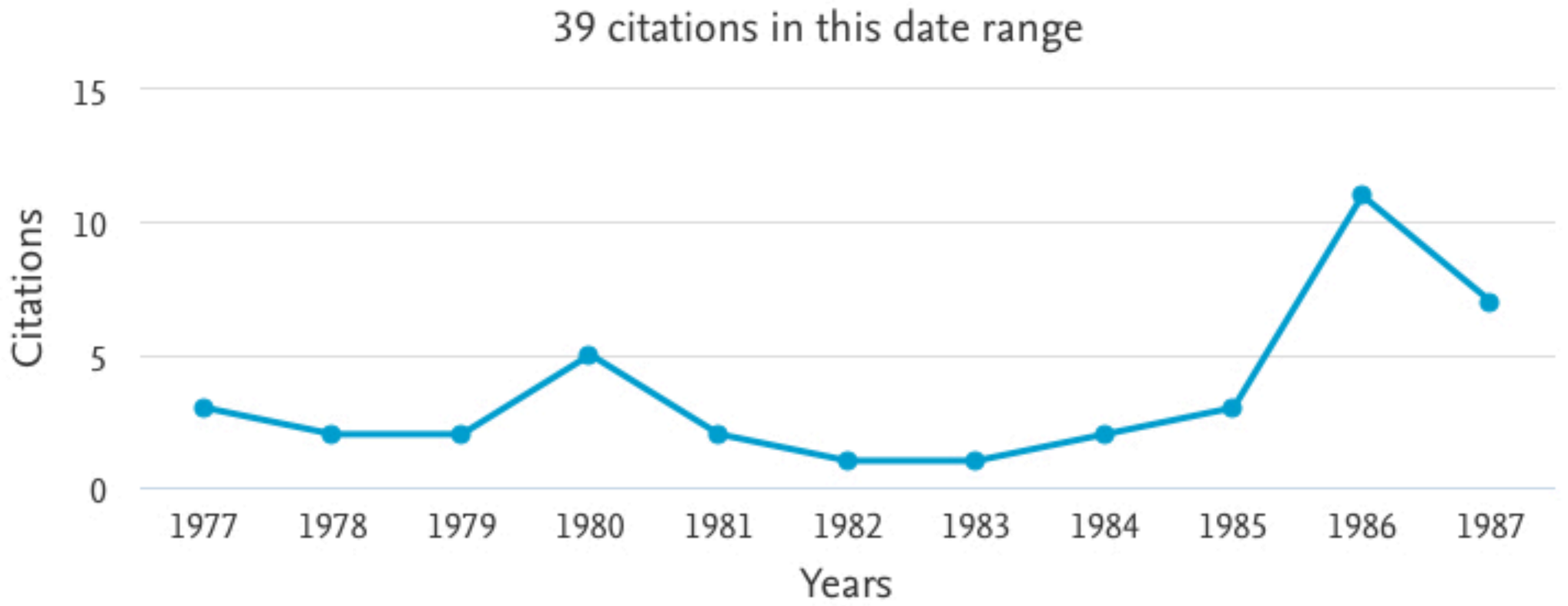
Systematic design of program analysis frameworks - 1979
Patrick Cousot, Radhia Cousot
Cited 639 times

Automatic discovery of linear restraints among variables of a program - 1978
Patrick Cousot, Nicolas Halbwachs
Cited 590 times

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France



ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

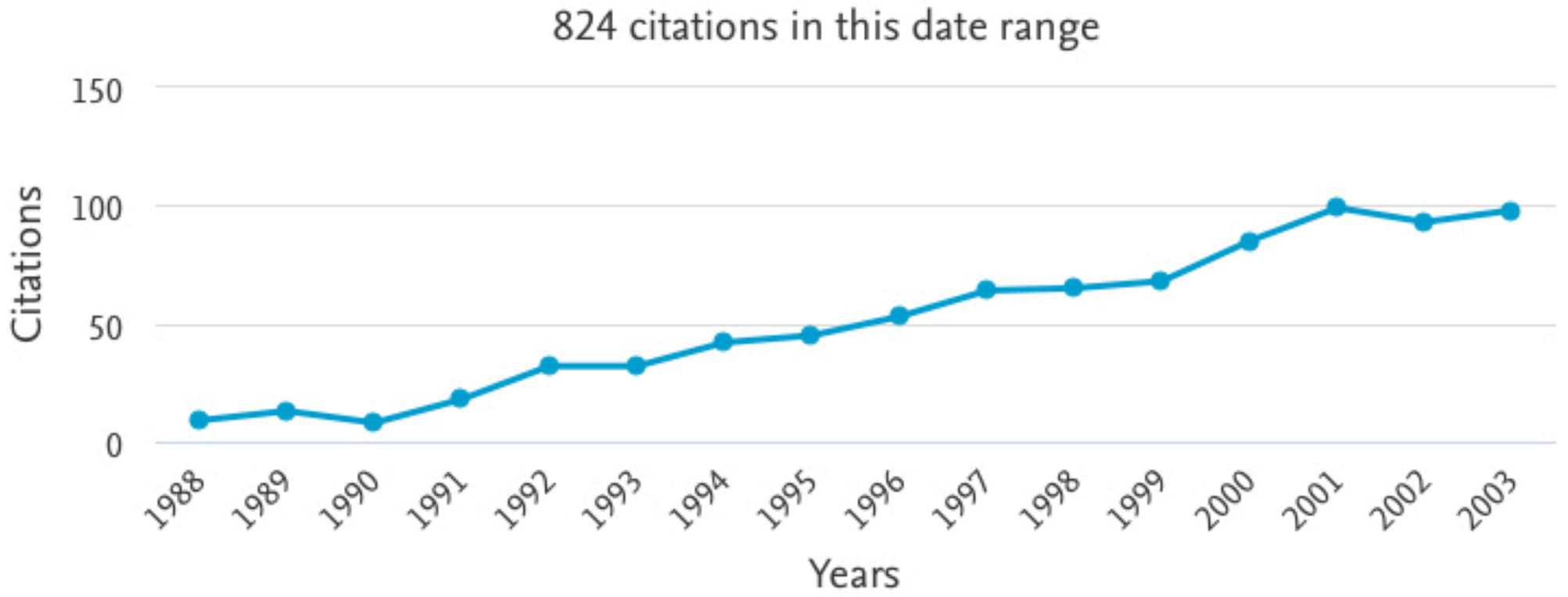
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

Scopus Metrics

3429 Citations

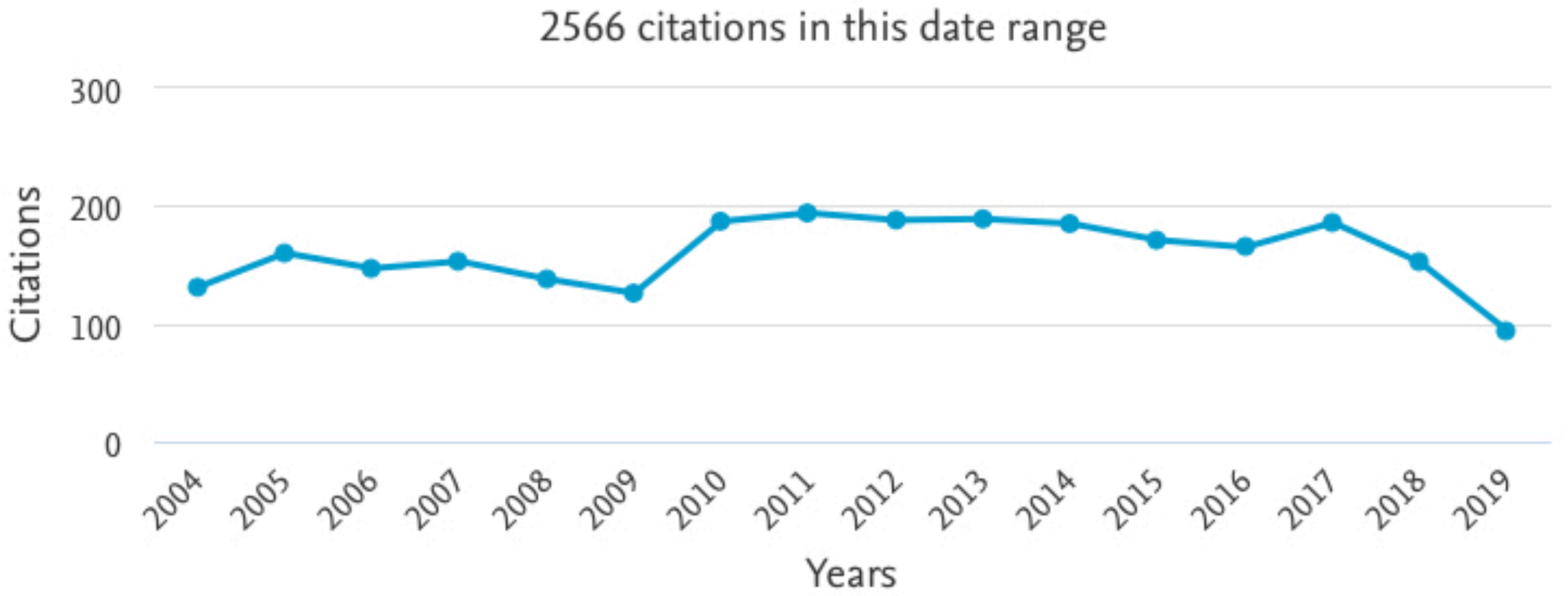
Total number of times this document has been cited in Scopus.



ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

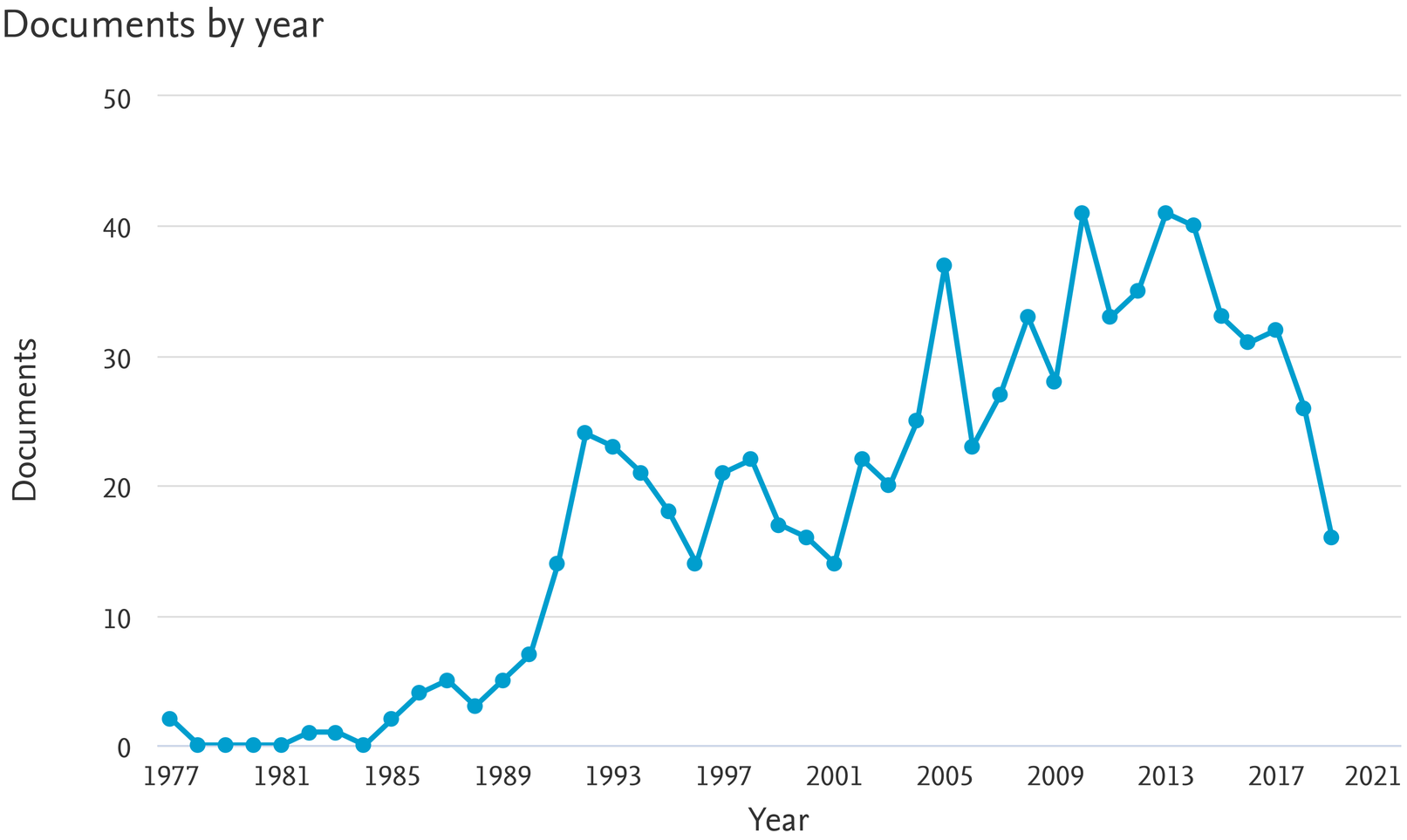


Articles whose title contains "Abstract Interpretation" OR "abstract domain"

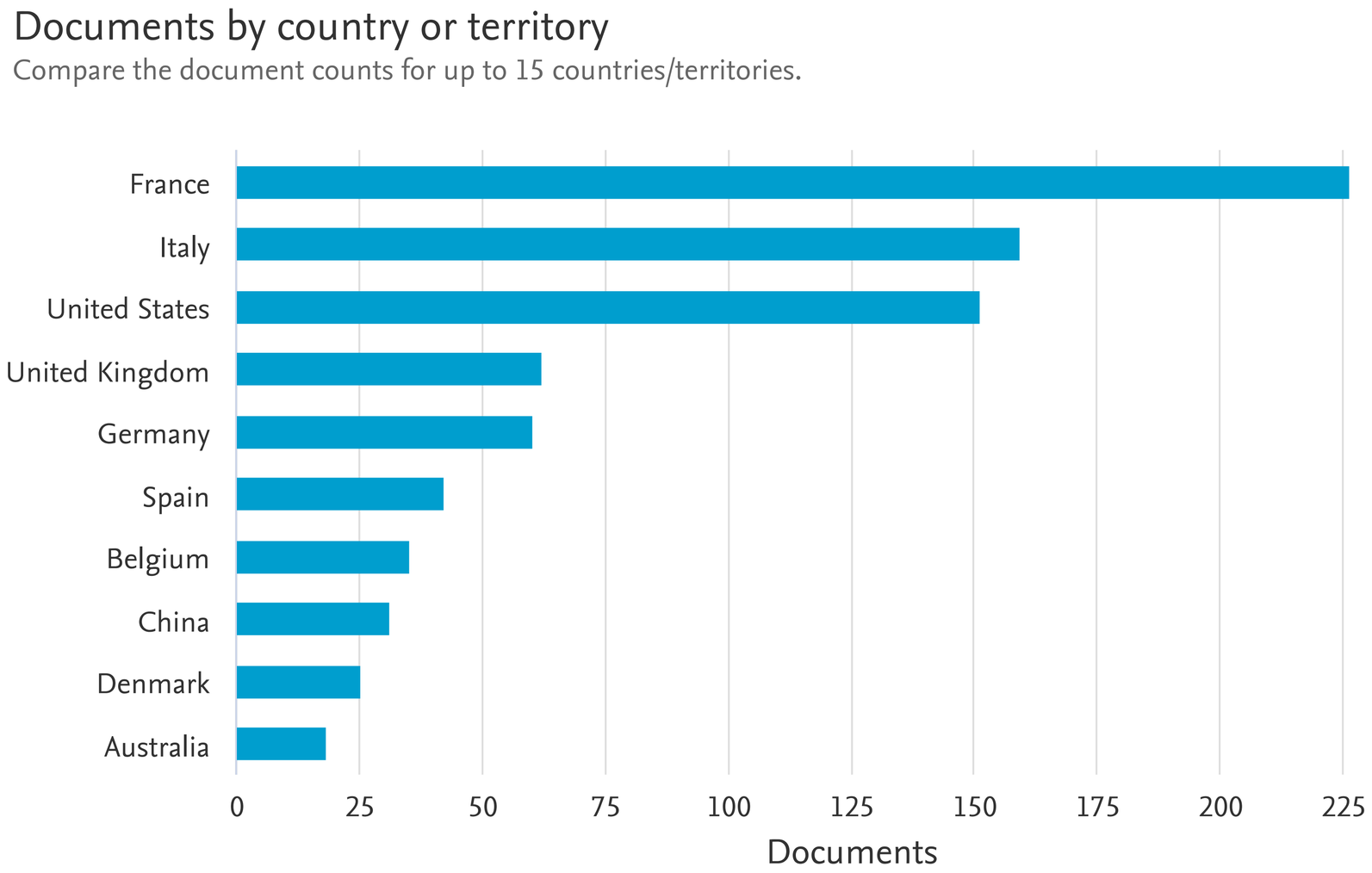
SCOPUS: 777 entries
1977-2000 = 220 entries
2001-2019 = 557 entries

DBLP: 884 entries
1977-2000 = 292
2001-2019 = 592

Source: Scopus



Articles whose title contains "Abstract Interpretation" OR "abstract domain"

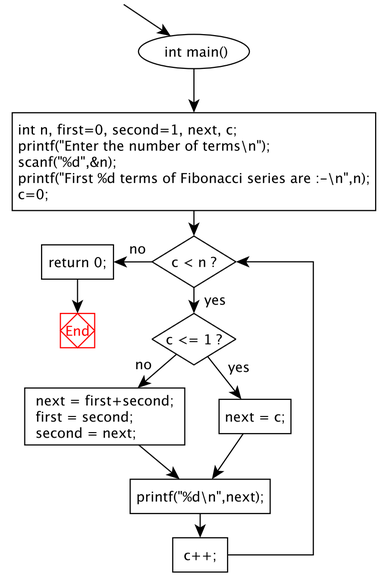


Year Range:
1977 — 2019

Source: Scopus

The PL perspective

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation

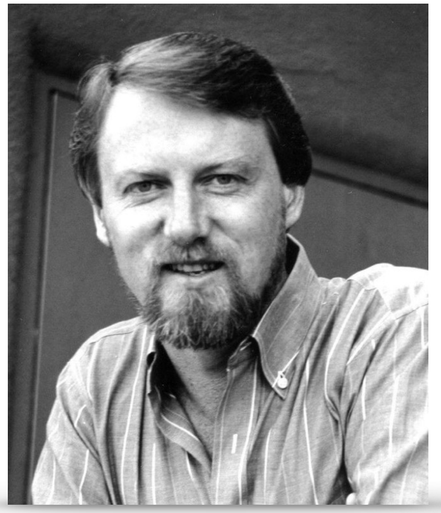


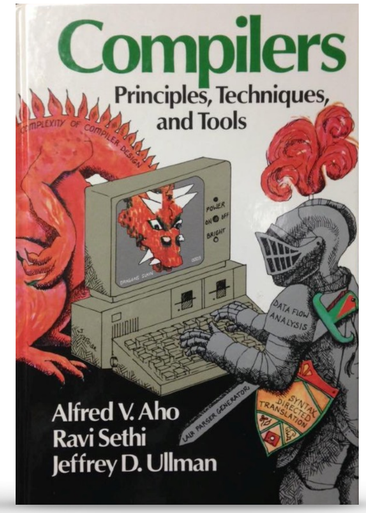
Imperative programming?

Lack of a clean/simple fixed point semantics

Dominance of compile-time data-flow static analysis (a la Kildall)

- compile-time optimisation
- compiler oriented





The 80s and 90s: The main difficulties in the spread of Abstract Interpretation

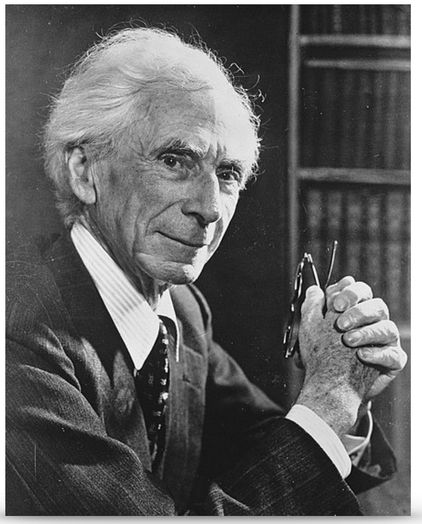
```
let square x = x * x

let rec fac n =
  if n = 0
  then 1
  else n * fac (n-1)

let rec fac2 = function
  | 0 -> 1
  | n -> n * fac (n-1)
```

Functional programming ?

Mathematical Logic as based on the Theory of Types.

BY BERTRAND RUSSELL.

Thus a predicative function of an individual is a first-order function; and for higher types of arguments, predicative functions take the place that first-order functions take in respect of individuals. We assume then, that every function is equivalent, for all its values, to some predicative function of the same argument. This assumption seems to be the essence of the usual assumption of classes [modern sets] . . . we will call this assumption the axiom of classes, or the **axiom of reducibility**.[14]



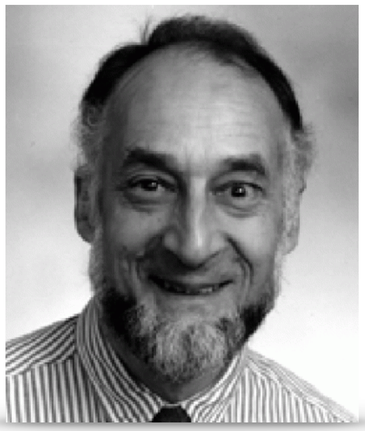
```
$ ocamlc -i trivial.ml
val square : int -> int
val fac : int -> int
val fac2 : int -> int
```

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation

```
let square x = x * x

let rec fac n =
  if n = 0
  then 1
  else n * fac (n-1)

let rec fac2 = function
  | 0 -> 1
  | n -> n * fac (n-1)
```

Functional programming?

Dominance of **type-based verification** (a la Milner)



Binding-time analysis (a little open window for AI)

Strictness was essentially the only true example!

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation

$$(\nu x)\ (\ \overline{x}\langle z\rangle.\ 0 \\ \mid x(y).\ \overline{y}\langle x\rangle.\ x(y).\ 0\ ) \\ \mid z(v).\ \overline{v}\langle v\rangle.0$$

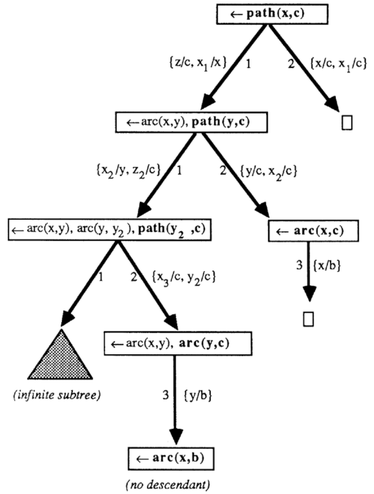
Concurrent programming?

Debate on the (right) model and the calculus

Dominance of **types** coming from the FP tradition and Milner's impact

Synchronisation considered too hard for static analysis applications

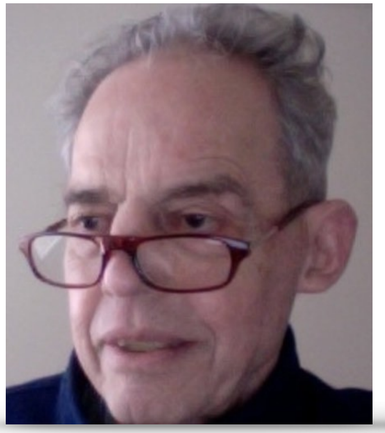
The 80s and 90s: The main difficulties in the spread of Abstract Interpretation

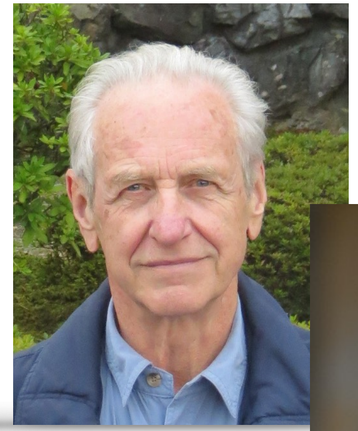


Logic programming?

Clean and simple fix-point semantics

Simple synchronisation mechanism

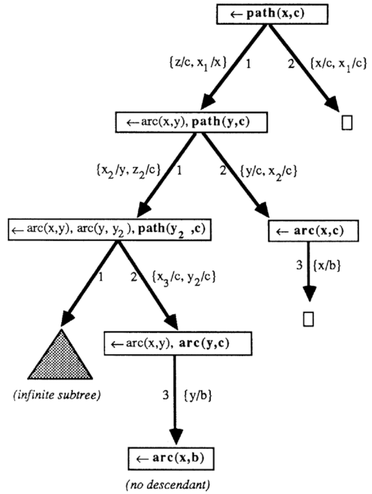




Van Emden & Kowalski: The Semantics of Predicate Logic as a Programming Language, *J. ACM* 23, 4, 1976, pp. 733-742.

$$T_p(X) = \{A \mid A \leftarrow A_1, \ldots, A_n \in P \ \wedge \ \{A_1, \ldots, A_n\} \subseteq X\}$$

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation



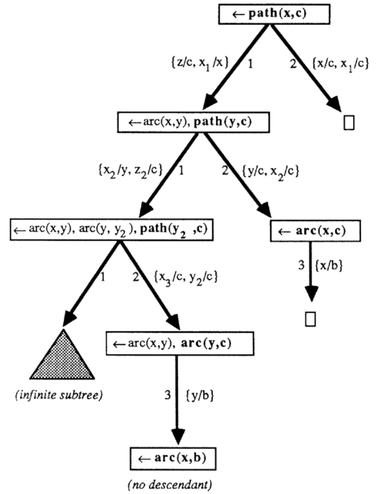
Logic programming?

Clean and simple fix-point semantics

Simple synchronisation mechanism

No type system!!

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation



Logic programming?

Clean and simple fix-point semantics

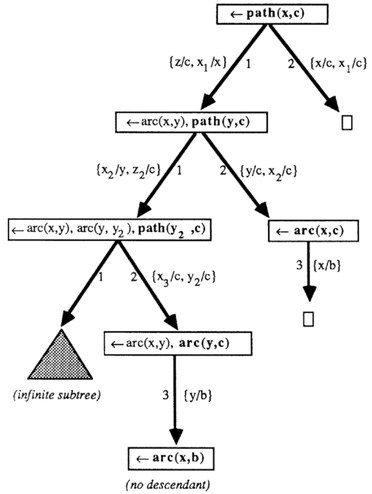
Simple synchronisation mechanism

Messy control!

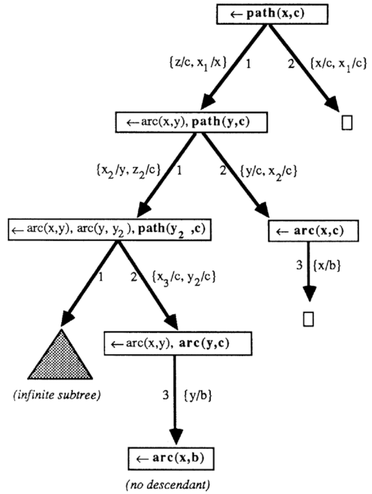
…heavy problems in optimisation => analysis

- AND/OR Parallelism (a true implementation **CIAO**)
- support to correct execution and debugging

The 80s and 90s: The main difficulties in the spread of Abstract Interpretation



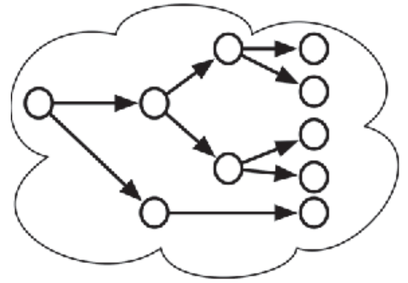
Logic programming?



Abstract Interpretation in LP

80s & 90s

Abstract Interpretation vs Model Checking
1992 —



Model Checking and Abstraction*

Edmund M. Clarke

Orna Grumberg

David E. Long

Predicate abstraction

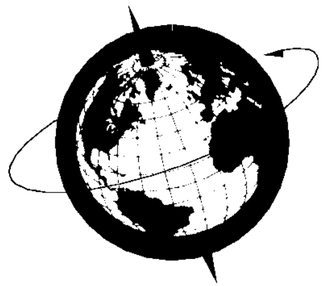
SLAM

The 2nd breakthrough!

The Industrialisation of Abstract Interpretation

Very first industrial implementation:

The interval analysis was implemented in the AdaWorld compiler for IBM PC 80286 by J.D. Ichbiah and his Alsys SA corporation team in 1980–87.



Ada

The International Language for Software Engineering

The 90s

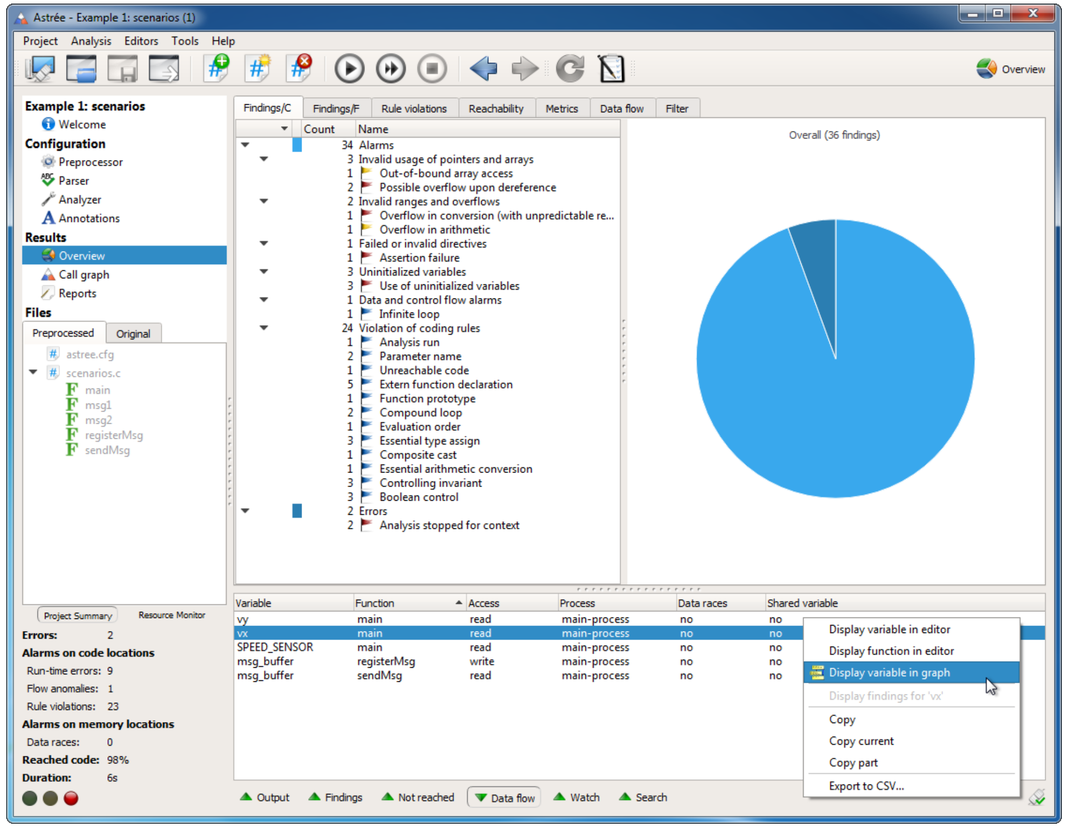
PolySpace TECHNOLOGIES

The Industrialisation of Abstract Interpretation

The 2000s!

The Astrée case and the BIG change!

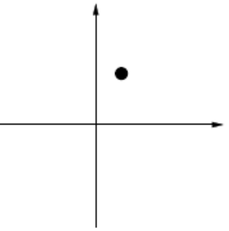
- Array index out of bounds
- Integer division by 0
- Invalid pointer dereferences
- Arithmetic overflows & wrap-arounds
- Floating point overflows and invalid operations
  - IEEE floating values `Inf` & `NaN`
- User-defined assertions, unreachable code
- uninitialised variables
- Elimination of false alarms by local refinement



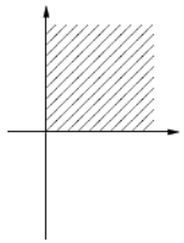
Industrialisation

The evolution of Abstract Interpretation

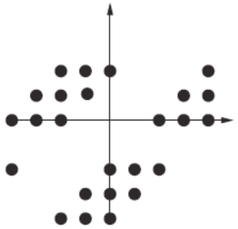
On the Abstract Domains
Specific Abstractions



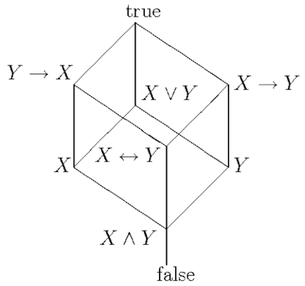
Constants
$X_i = c_i$
[Kildall73]



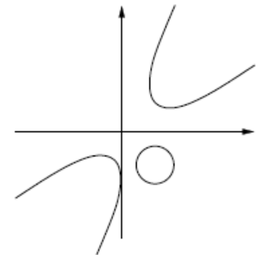
Signs
$X_i \geq 0 \wedge X_i \geq 0$
[Cousot Cousot 76]



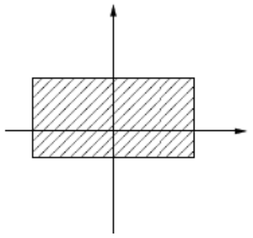
Trapezoidal congruences
[Masdupuy 1993



Groundness & dependency
X -> Y
[Marriott Sondergaard 94]

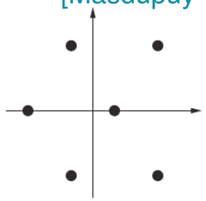


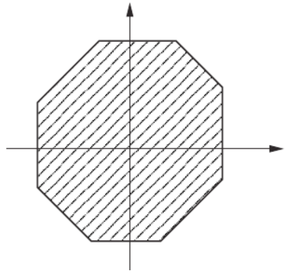
Varieties
$P_i(X) = 0$; polynomial form
[Carbonell & Kapur 04]



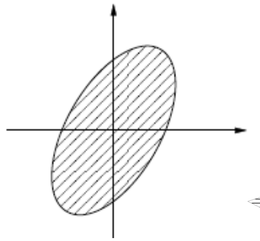
Intervals
$X_i \in [a_i; b_i]$
[Cousot Cousot 76]

Strictness
[Mycroft 91]

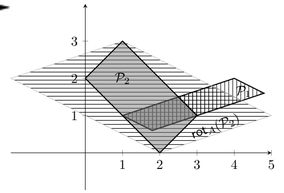




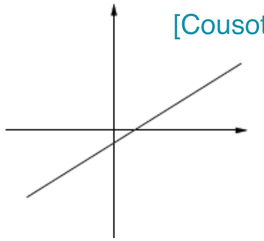
Octagons
$\mp X \mp Y \leq d$
[Mine01]



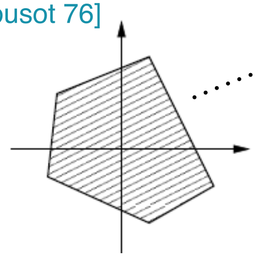
Ellipsoids
$aX^2+bY^2+cXY \leq d$
[Feret 2004]



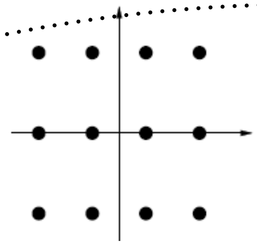
Parallelotopes
[Amato Scozzari 2012]



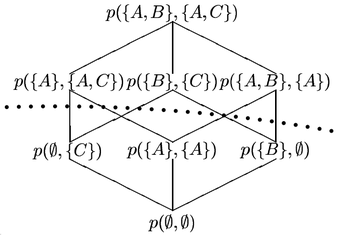
Linear Equalities
$\sum_i a_{i,j} X_i = d_j$
[Karr76]



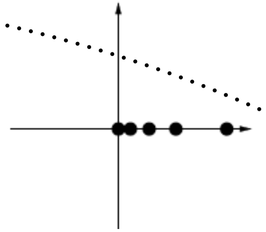
Polyhedra
$\sum_i a_{i,j} X_i \leq d_j$
[Cousot Halbwachs 78]



Simple Congruences
$X_i \equiv a_i\ [b_i]$
[Granger89]



Sharing
{{XY}{Z}}
[Jacobs & Langen 92]

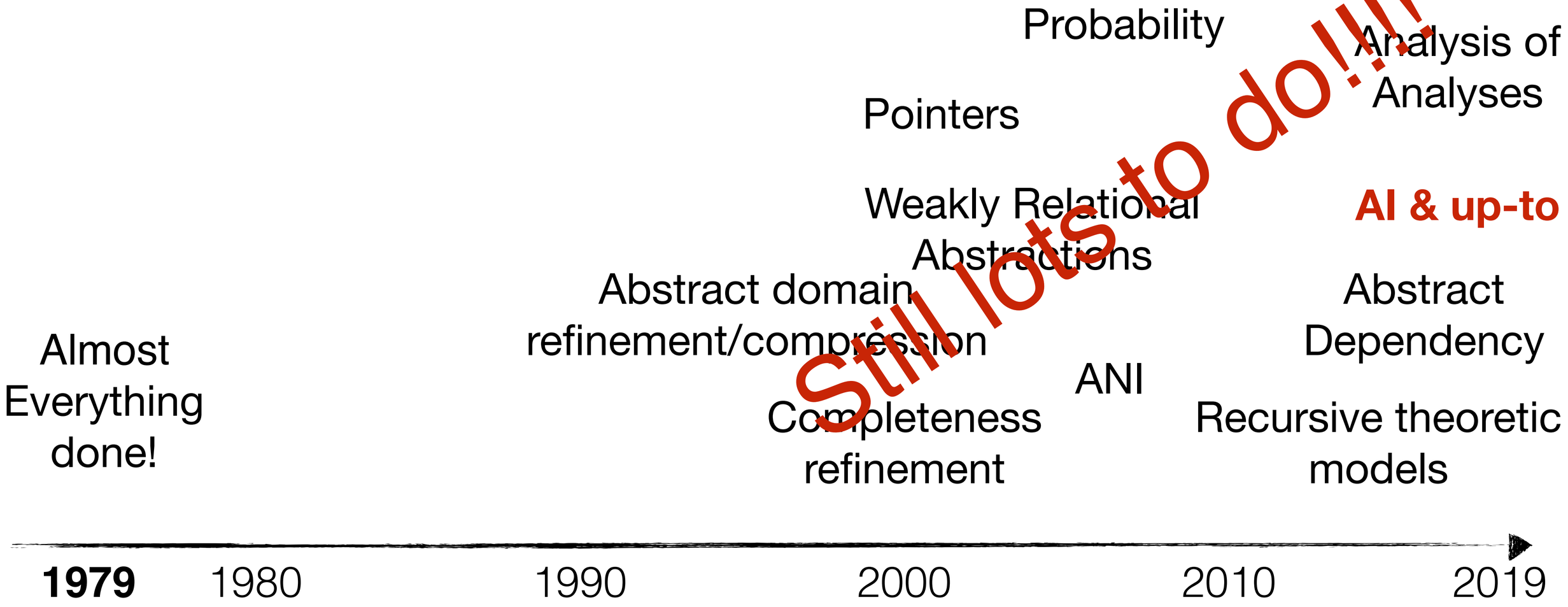


Power Analysis
$X \in a^{f(n)}$ wirh f a non rel. analysis
[Mastroeni01]

PPL

1977 1980 1990 2000 2010 2019

On the Theory of Abstract Interpretation
Independent from the abstraction



Thanks!

History is neither watchmaking nor cabinet construction. It is an endeavour toward better understanding.

—Marc Bloch