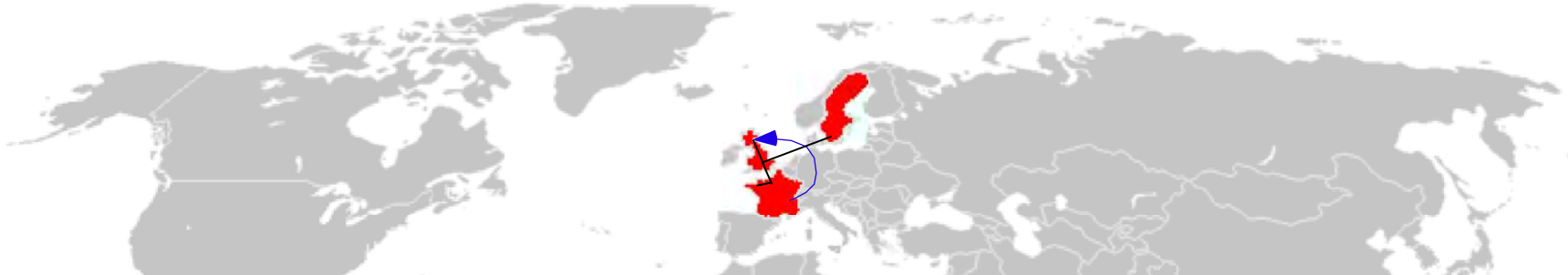
3. Examples of initial diffusions

France to UK to Sweden back to France



- Rod Burstall visits Grenoble and gets interested in abstract interpretation
- Back in Edinburgh, he propagates the idea to Alan Mycroft, who developped *Strictness analysis* which spreads all over the UK (Samson Abramsky, Chris Hankin, ...) and beyond (e.g. David Sands in Sweden) and even back to France (Thomas Jensen in Paris and next Rennes)

From Germany to Israel, the US and India

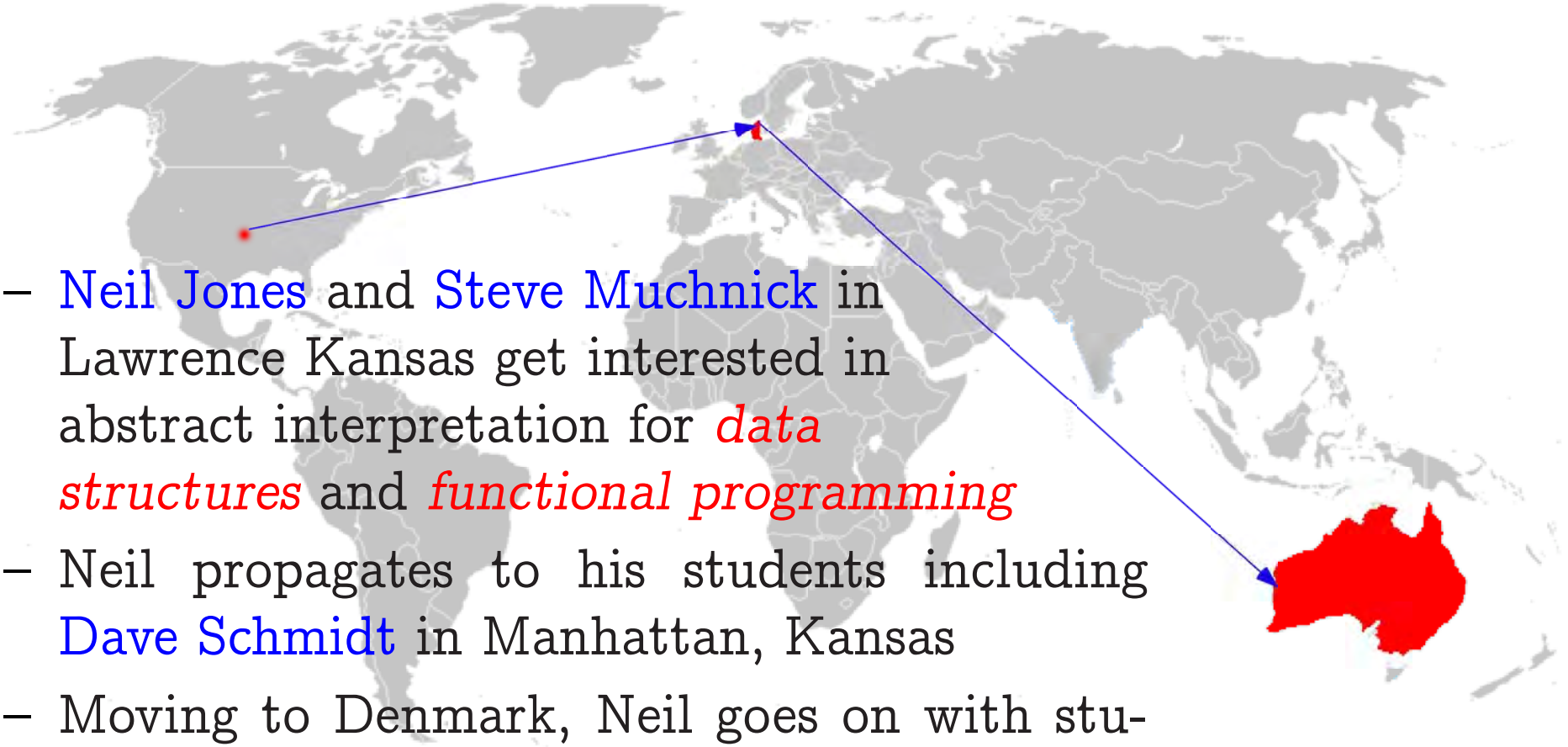
- Reinhard Wilhelm gets interested in abstract interpretation for *compiler optimization*
- Propagates to Mooly Sagiv in Israel who visits IBM Hawthorn (where his students Eran Yahav and Greta Yorsh are recruited)
- Mooly works there with Ganesan Ramalingam who moves to Bangalore, India.

From Italy to the US and Spain

- Giorgio Levi gets interested in abstract interpretation for *logic programming*
- He propagates to his students including Roberto Bagnara, Roberto Giacobazzi, and many others [21]
- Francesca Rossi meets Manual Hermenegildo in Austin Texas who brings back abstract interpretation to Spain

[20] Gianluca Amato, Marco Comini, Maurizio Gabbrielli, Roberta Gori, Maria Chiara Meo, Francesca Rossi, Francesca Scozzari, Fausto Spoto, Giuliana Vitiello, Paolo Volpe, Enea Zaffanella, to cite a few

From Kansas to Denmark and Australia



- Neil Jones and Steve Muchnick in Lawrence Kansas get interested in abstract interpretation for *data structures* and *functional programming*
- Neil propagates to his students including Dave Schmidt in Manhattan, Kansas
- Moving to Denmark, Neil goes on with students including Flemming Nielson and Harald Søndergaard who moves to Melbourne, Australia

From Illinois to Korea and England

- Luddy Harrison in Illinois gets interested in abstract interpretation for *static analysis of parallel programs*
- Luddy propagates to his students including Kwangkeun Yi who moves to KAIST and next to Seoul, Korea.
- Kwang's student Hongseok Yang moves to London.

Example of thematic diffusion

– At POPL 1977 and 1979, we discuss a lot with Edmund Clarke working on program proof methods (who invites us to Harvard [22])

– At the same time, we had also a lot of discussions with Joseph Sifakis, next door to our office in Grenoble, working on the anaysis of Petri nets

– Could not convince them to use infinite abstractions!

[21] where we had the immense pleasure to have Garrett Birkhoff attending our seminar!

Difficulties of Static Analysis

– undecidable $\Longrightarrow \neg$ (automatic + infinite state + termination)!

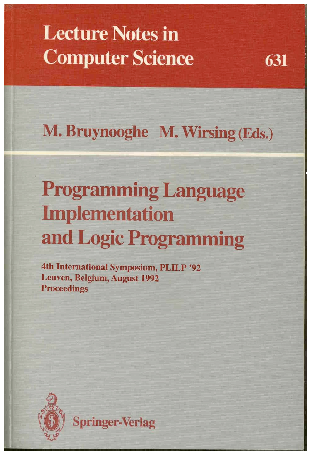
– for a programming (and a specification) language, not for a given model of a given program:

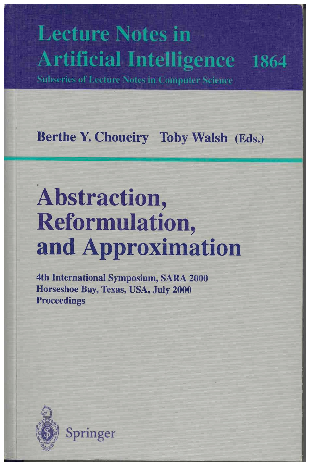
$$\forall P \in \mathbb{P} : \forall S \in \mathbb{S} : \mathfrak{S}_{\mathbb{P}}[\![P]\!] \subseteq \mathcal{S}_{\mathbb{S}}[\![P, S]\!]?$$

or, more simply for an *implicit specification* $\mathfrak{S}[\![P]\!]$:

$$\forall P \in \mathbb{P} : \mathfrak{S}_{\mathbb{P}}[\![P]\!] \subseteq \mathfrak{S}[\![P]\!]?$$

Reminder





- Proving a given program is always possible using a finite abstraction [CC92a, Cou00]
- Discovering (or computing) this abstraction is equivalent to the program proof [Cou00]
- When considering proofs for all programs of a programming language, no finite abstraction will do for all programs [CC92a]

4. Scope of application of abstract interpretation

Applications of Abstract Interpretation

– Static Program Analysis [CC77a], [CH78] [CC79] including

- Abstract Debugging [Cou81], [Bou93a], [Bou99], [Riv05a], etc
- Dataflow Analysis [CC79], [CC00], [Sch98], [Tzo97], etc
- Numerical Properties [CC75], [CC76], [CC77a], [CH78], [Fer04], [Fer05c], [Fer05d], [Gra89], [Gra91b], [Gra91a], [Gra97], [Hal79], [Mas91], [Mas92], [Mas93b], [Mas93a], [Min04a], [Min06b], [Min05], [Min04b], [Min06c], [Min01b], [Min01a], [Min02], etc
- Heap / Shape analysis [CFRS98] [CC77c], [CC77d], [Deu90], [Deu92b], [Deu92a], [Deu93], [Deu94], [Deu95], [HP06], [DRS00], [GH96a], [GH96b], [LAS00], [MRF+02b], [MYRS05], [GH07b], [Mau00a], [Mau00c], [Mau99a], [Mau03], [Mau99b], [Mau00b], [Mau07], [Mau98], [Min06a], [RPHR+07], [SFRW90], [SRW00], [SRW96], [Str88], [Str92], [Ven98a], [Ven96], [Ven99], [WW96], etc

- Predicate Abstraction [ADI02], [ADI03], [BBC+06], [BMMR01], [BMR02], [CHH04], [Cou03], [DD02], [DDP99], [FQ02], [Gra94], [Gra99], [GL93c], [GL93b], [GS96], [GS97b], [HBGT02], [LBC05], [LGS+95], [MRS02], [VPP00], etc
- Probabilistic Analysis [DPHW05], [PHW06], [DPW00], [Mon01c], [Mon01b], [Mon01a], [Mon00], [Mon03a], [Mon03b], [Mon03c], etc
- Set-based & Constraint-Based Analysis [CC95], [FF99], [Hei92a], [Hei92b], [Hei94], [Pod97], [TTD97], etc
- Typing & Type Inference [CC75], [CC77c], [Cou97b], [Mon91], [Mon92b], [Mon92a], [Mon93a], [Mon93b], [Mon94], [Mon95b], [Mon95a], etc
- Worst-case execution time (including cache/pipeline behavior) [ACMW96], [FHW04] [FMWA99a] [FMWA99b], [FHL+01], [FW99], [TSH+03], etc
- . . .

For which programming languages?

– Imperative languages e.g.

Interprocedural Analysis [Bou90], [Bou92c], [Bou93b] [CC77d], [CI98], [GT07], [JGR05], [JM82], [JM86], [JN95], [JR97a], [JR97b], [Rep94] [SRH96], [SFM07], [CIY95], etc

Pointer Analysis [CFRS98], [Cou78], [CC75], [CC76], [CC77c], [CC77d], [DLFR01], [Deu94], [Deu95], [EGH94], [FGLM96], [GH98], [Hin01], [HBCC99], [HP98], [Min06a], [RLS+01], [SBF99], [SIH97], [Ven99], [XRM01], etc

Vectorization and Parallelization [BK89], [IJT91], [Jou86], [Jou87], [Kal93], [KBC+99], [Mid96], [PW94], [TFI86], [Tzo97], [YAI95], etc

. . .

For which programming languages? (Cont'd)

– Functional languages including

Strictness/Comportment Analysis [Abr86], [AH87a], [AH87b], [AJ91], [Ben93], [BHA86a], [BHA86b] [Con94], [CC93], [Deu91], [GH93], [HY86], [Hug86], [JN95], [Mau94], [Mon95b], [Myc80], [Myc81], [MN83], [MN83], [MS95], [Nie87], [Nie87], [Nie82], [Nie86], [Nie88], [Nie88], [Nie89], [Nie96], [NN92], [NN93], [SPR90], [SMR91], [SR95], [Sol94], [SGRNN98], [SGRNN98], [SGRNN98], [Véd95], [Wad87], etc

Binding-Time Analysis [Asa99], [Bul93], [BK95], [Con93], [Dav93], [DHM95], [GR92a], [Hen91], [HS95], [HS95], [HS91], [Jen92], [Jon91], [Jon94], [Lau89], [KN95], [NN88], [NN88], [Pal93], [RG95], [SNN92], [Véd00], [Véd94], [Véd95], etc

Control-Flow/Environment Structure Analysis [DP97], [Mig07], [MS06b], [MS06a], [MS07], [Shi88], [Shi04b], [Shi91], [Shi04a], [TJ94], etc

Escape Analysis [Bla00], [Bla98], [Bla03], [Deu97], [PG92], [TJ92], etc

Exception Analysis [Yi94], [Yi98], [YR97], etc

For which programming languages? (Cont'd)

– (Concurrent/Constraint/Functional) Logic Languages

[BGL92a], [BCFP99], [BF97], [BCGL91], [BGL89], [BCGL92], [BCGL93], [BGL93a], [BGL93b], [Bou92a], [BB94], [BBD94], [Bru87], [Bru91], [BB93], [BCM95], [BJCD87], [BJ92], [BCHP96], [Cas97], [CCG94], [CCH93], [CLM96a], [CLM96b], [CDY94b], [CDS98], [CFMW93], [CFMW97], [CFM94b], [CMB+95], [CFMW92], [CGS89], [CDY90b], [CFM91], [CDG93], [CGdlBBH93], [CMB+93], [CDY94a], [CFM94c], [CC90], [CF91a], [CL94], [CLM95], [CPHB06], [Cor89], [CMRLC93], [MC92], [CFW92], [CLCvH00], [CMS90], [DRW96], [Deb93], [Deb87], [Deb88b], [Deb88a], [Deb89a], [Deb91], [Deb94], [Deb95], [DR94], [DSB88], [EM91], [FG98], [FR93], [GG94], [Gal92], [GBS95], [GCS88b], [GdW94], [GCS88a], [dlBHB+96], [dlBMS95], [Get94], [Gia94], [Gia93], [GDL93], [GDL95], [Han96], [Han98], [Han93], [HL99], [Her97], [HWD92], [HS03], [Hor92], [HKL04], [JB92a], [JB93], [JBD95], [JBE94], [JS98], [JS94], [JS87], [JMM97], [Kåg93], [Kan93], [KKMH89], [KK93], [KMSS96], [KR90], [KS90], [Cha94], [LCMVH91], [LCF97], [CRvH94], [LCVH94], [CvH92], [LCVH92a], [LMBG00], [Lin95], [MRB92], [MU87], [MJMB89], [Mar93], [MGdlBH93], [MS88a], [MS89], [MS90], [MS92], [MSJ94], [Nil88], [Nil90], [Nil91], [PDL92], [PCH+04], [PAH05a], [PAH05b], [PAH06], [PAH07], [PH96b], [VRD90], [Soh93], [SL99], [TTD97], [Ued99], [VHDLCM93], [dlC91], [Wær88], [WHD88], [WW96], etc

including

Abstract Proof & Debugging [BDD+97], [CGL01], [CLMV99], [CLMV97], [CGLV99], [CLM01], [CLV95b], [CLV95a], [CLV94], [HPBLG03b], [HPBLG02], [HPBLG03a], [HGP05], [HPB99], [LV98], [LG93], [Oga99], [PCPH06], [PH07], [PBH98], etc

Control Analysis (Backtracking Optimization/Determinacy/ Failure/Negation/Parallelization/...)

[BdlBH99a], [BdlBH99b], [BLGH04], [BGdlBH94], [GH94], [CD85], [CDD85], [CdlBBH97], [CFM94a], [CMT00], [DLGH97], [DLGHL97], [DLH90], [DW86b], [DW89], [GPA06], [GdlBBH96], [GR90], [GR92b], [GMP91], [JL92], [JB92b], [JB92c], [KHK88], [KK87], [KS92], [Klu88], [LDL91], [LDL92], [LGBH05], [LGBH05], [Mal92], [MS88b], [MG91], [Mog96], [MHMNH01], [Pos94], [PH99a], [PH99c], [PH96a], [PHG99], [Red84], [Ric90], [Sah91a], [Sah91b], [ST84], [Søn86], [SST92], [Ued87], [Win88], [WW88], [Yan92], etc

Groundness & Freeness Analysis [Bag96], [Bag96], [BS99], [BGL89], [BM88], [Cod99], [CD95], [CD93], [CDY90a], [CDY91], [CDY91], [CDFB93], [CM96], [CSS99], [Cor89], [CDFB96], [CFW96], [CFW91], [CFW91], [CF91b], [CF91b], [DJBC93], [ELCRVH92], [ELCRVH93], [HAZCK00], [HS99], [JL89], [CH93], [MSD90], [RBM99], [Sco97], [SC92], etc

Mode Analysis [BJ88], [CU96], [Deb87], [Deb89b], [DW86a], [DW86a], [DW88], [DW88], [RS95], [Lu96], [MU87], [Mar94], [MJMB89], [MT95b], [Mel81], [Mel85], [Mel86], [Mel87], [Red84], [RP96], [Som87], [TL96], [JT97], [Tay89], [Ued87], [VB99], [VRDW87], [VRDW87], [VS99], [War77],† [WHD88], etc

Heap and Sharing Analysis [AS01], [Bag96], [BZGH01], [BHZ97], [BC93], [BCM94], [BDB+96], [BdlB04], [CLB00], [CLB97], [CS98], [CF99], [CF91b], [Fec96], [dlBH92], [HBZ98], [PL93], [LS99], [LK04], [MKSH06], [MSHK07], [Mul93], [MWB90], [MH89], [MH91], [MH92], [NBH05], [NBH07], [Tay89], [VU84], etc

(Non-)Termination, Cost & Size Analysis [CT99], [DLGHL94], [DL90], [DL91], [DL93], [DS92], [DSV99], [Der04], [SD94], [SS04], [SV95], [SVB92], [GT96], [GC01], [GL99], [KHK87], [AK97], [Leu92], [Leu98], [MT95a], [MM92], [MGS96], [NBH06], [NMLGH07] [PM04], [PH07], [Plü97], [Ser03], [SS01a], [SS01b], [SS02], [Soh94], [SSS97], [VG90], [VS99], [VSS99], [VDS91], [VDS93], [VDS92], etc

Type Checking & Inference [BS88], [BGL92b], [BM99], [BJ88], [CP98], [CD94], [CD94], [CL00], [BC91], [DBLC90], [FS91], [FS91], [GZ86], [HK87], [KH85], [Klu87], [LS98], [LS98], [MLGP$^+$07], [Mis84], [MR85], [SG95], [vHCLC94], [VB02], [VB02], [Vol98], [YS87], [YS91], etc

Abstract specialization & Binding-Time Analysis [CGLH05], [PH07], [PH99b], [VB99], [VBL04], etc

Abstraction Carrying Code [APH04b], [APH05b], [AASPH06], [HALGP05], etc

For which programming languages? (Cont'd)

- Dataflow & Synchronous Languages [BJT99], [BJT99], [Fer93], [Hal93], [Hal94], [Hal98], [HM95], [HM95], [Jen95], [Mas98], [MK84], etc
- Quasi-Synchronous Languages [Ber06a], [Ber05], [Ber06b], etc
- Parallel and Distributed Languages [BCT04], [BFSV99a], [BFSV99b], [BW07], [BET03], [CH92b], [CH94], [Cla80], [Col95b], [Col95a], [CC80], [CC84], [CC85b], [FGL96], [GH07a], [GL94], [GL93a], [GL92], [Ste98], [Mer90], [Mer91], etc
- Circuits [KN05], [TM04], etc
- Mobile Code [APH04a], [APH05a], [Fer01b], [Fer02], [Fer00b], [Fer05a], [Fer05b], [Fer05b], [Fer00a], [Fer01a], [HJNRN99], [Huc99], [RNN00], [Ven98a], [Ven97], [Ven98b], etc

– Object-Oriented Languages [ACSE99], [BC05], [Bla99], [Bla00], [Bla03], [DDC95], [RGD05], [GSR03], [JS01], [KPK02] [Log04c], [Log03], [Log04b], [Log04a], [LC05], [LC06], [PFKS05], [SJ03], [SP03], [VHU98], [XRM01], etc

– Hybrid & Real Time Systems [ACH+95], [HPR94], [HPR97], etc

– Cellular Automata [HTYS04], etc

– . . .

Applications of Abstract Interpretation (Cont'd)

- Grammar Analysis and Parsing [CC03], [CC06], etc
- Hierarchies of Semantics and Proof Methods [BDP98], [CC92b], [Cou02], [CC07], [Cou08], [Gia96], [GM03b], etc
- Temporal Abstractions and (Abstract) Model Checking [CC00], [CC99], [GR02] [GR04a], [GR04b], [GR06], [HR96], [Mas04], [Mas02], [Mas03], [Mas01], [RT05], [RT06b], etc
- [Bi]simulations/Strong Preservation [RT02], [RT04a], [RT04b], [RT04b], [RT06a], [RT06b], [RT06c], [RT07a], [RT07b], [RT07c], etc
- Security analysis of cryptographic protocols [Mon99], [Bla01], etc

Applications of Abstract Interpretation (Cont'd)

– Language-based security [BBF02], [GM05], [GM04], [PBN07], [PL07], [SS99], [Sch00], etc

– Program Transformation [CC02b], including
  - code optimization, partial evaluation, etc [CC02b]
  - Code certification [Riv05b], [Riv04b], [Riv04a], [Riv03], etc
  - Software Watermarking [CC04]
  - Code obfuscation [PG05a], [PG05b]
  - Semantics-based obfuscated malware detection [PCJD07]

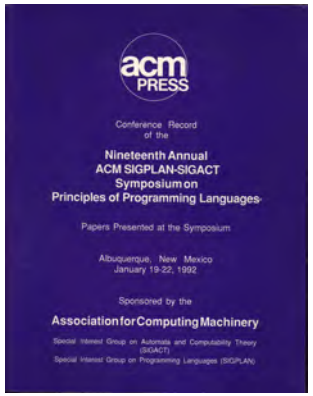
Applications of Abstract Interpretation (Cont'd)

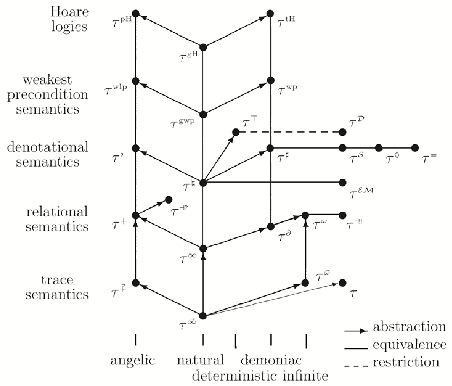
- Databases [AGM93a], [AGM93c], [AGM94], [AGM92], [AGM93d] [AGM93b] [BPC01], [BP99], [BS97], [Tom97], etc
- Image processing [Ser94], etc
- Computational biology [Dan07], [DFFK07], [DFF+07], [FS06], etc
- Quantum computing [JP06], [Per06], etc

All these techniques involve sound approximations that can be formalized by abstract interpretation

5. Widening the theory

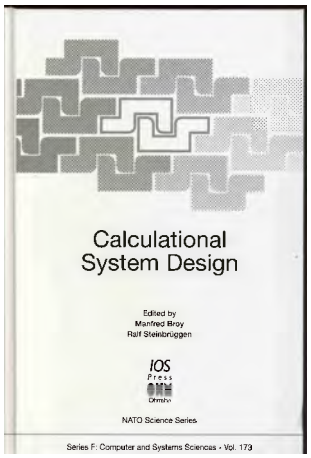
Hierarchies of semantics



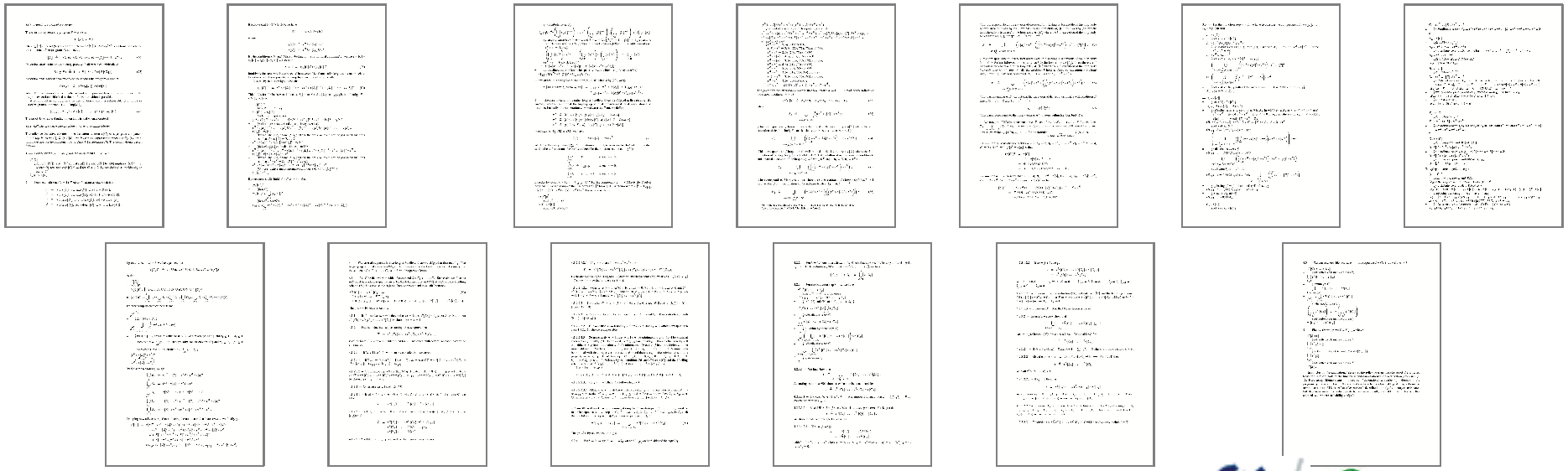


- The literature on the semantics of programing languages is copious and abundant (often on toy examples)
- No semantics is universal and hierarchies at various levels of abstraction are needed [CC92b, CC07, Cou08]
- In practice, programming languages have no rigorous semantics so implementations are disparate
- Formal methods would need some more rigor from design to verified implementation

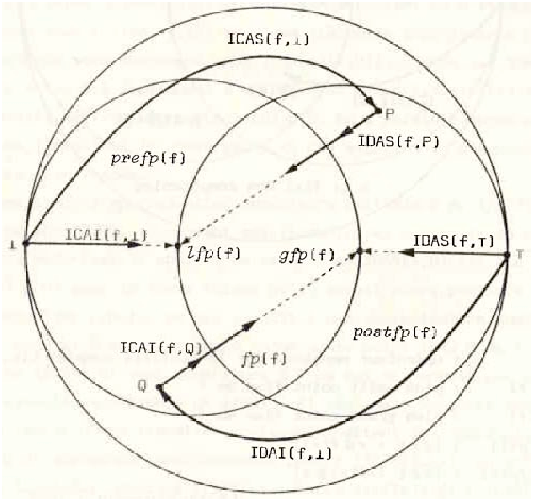
Calculational design of abstract interpreters

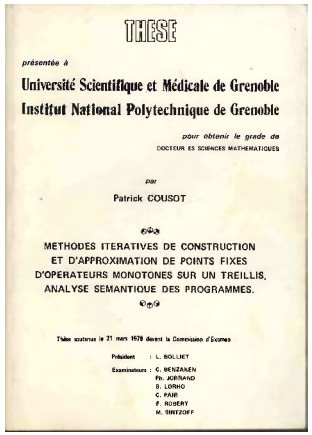


- The theory of abstract interpretation underlies the calculational design of static analyzers [Cou99]
- Besides academic examples [CJPR05, Mon98, Pic05], is this applicable in practice? at least for the verification phase?



Under-approximation




THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

par

Patrick COUSOT

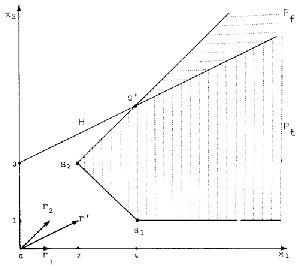
METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE POINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

- Most abstractions involve over-approximations ($\forall$)
- Under-approximation is dual ($\exists$), so theoretically solved, and is indispensable (e.g. for liveness)
- We are lacking non-trivial under-approximations [23]
- This is because we are missing suitable dual widenings [24]

[22] e.g. testing for a trace semantics

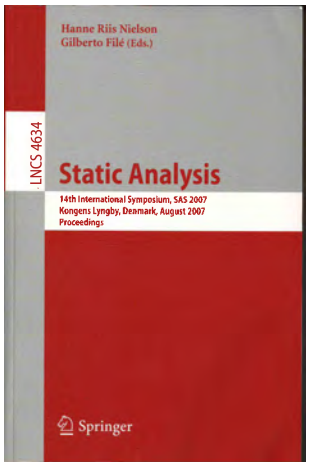
[23] but in the trivial case of finite systems

Codings and algorithms for abstract properties



- The use of universal representations of properties in program verification (like BDD's or terms in deductive methods) is a great convenience
- One can then use a generalist abstraction [Cou97a] e.g. TVLA [DRS00], [LAS00], [MRF+02a], [RBR+05]
- The only possible refinements are through the semantics (not the abstraction)
- Mathematicians do not use logic for their reasonings, since this representation is not adapted to problem solving (e.g. geometry)
- Similarly, to effectively scale up, static analyzers must use carefully designed computer representations of abstract properties and transformers

Soundness, Completeness, Refinement



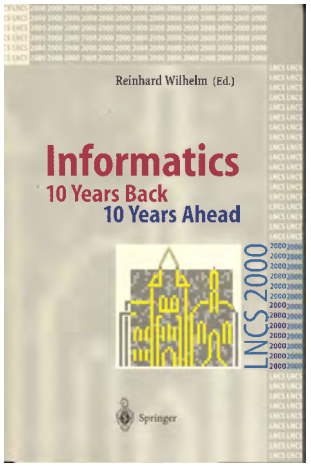
- Refining an incomplete abstraction is absolutely necessary to get rid of false alarms
- Theoretically, refinement [CGR07] is a solved completeness problem [25]
- Automation does not scale up or even terminate for infinite systems

– How can we automate the discovery of an efficient computer representation of refined abstract properties and algorithms for refined abstract transformers?

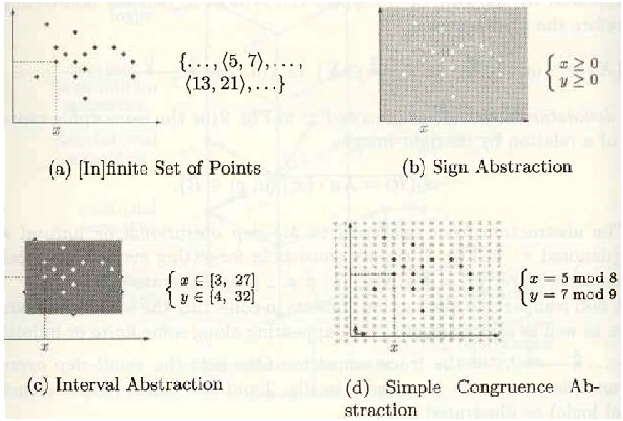
[24] [BPR02], [Bou92b], [Cou00], [GRS05], [GQ01], [GR04b], [GR06], [GR97a], [GRS00], [HT98], [JHR99], [MR05], [Myc93], [Ran01], [RT02], [RT04b], [RT04b], [RT06c], [RT07b], [Sch06]

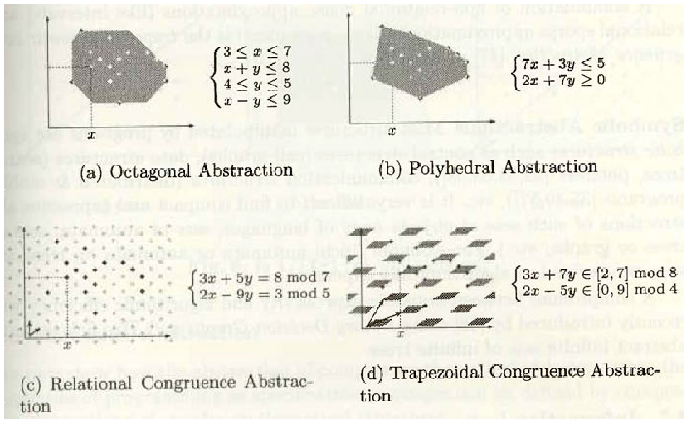
Widening the applications: shareable resources

Numerical Abstract Domains



- Numerical static analyzes have known a continuous well-structured development thanks to the concept of abstract domain
- These abstract domains are implemented in shared and reusable libraries (such as APRON [JM], [Min], PARMA [BHZ06], [BRZH02], etc)





Symbolic Abstract Domains

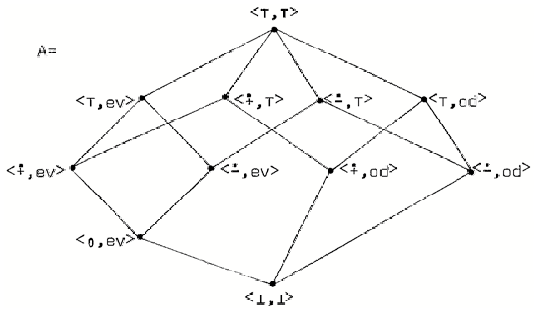
– The notion of symbolic abstract domain is developping for non-numerical (pointer, heap, graphs, shape analysis) analyzes [Mau99b], [Mau00a], [Mau00b], [Mau03], [Mau07]

– Libraries are needed for results to be easily shared, compared and reused

Abstract fixpoint iterators

– No such idea either exists for program control structures (procedures, modules, etc), and composable abstract iterators would be wellcomed

Combination of abstractions



- The design of abstractions by combination of abstract domains and reduction [CC79, Gra92] has been well studied [26]
- The refinement of abstract interpreters with new abstract domains, in particular using the reduced product [CC79], is the key to false alarm solving
- Practical solutions have been implemented [CCF+06]
- It remains to be incorporate reduction in shared libraries of abstract domains

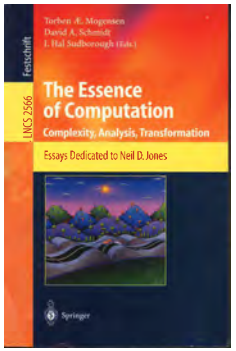
[25] [FR94], [FR96], [FGR96], [FR99], [GR96], [GM03a], [GR95], [GPR96], [GR97c], [GR99], [GS97a], [GRS98b], [Gia98], [GR98b], [GR98a], [GR98c], [GS98], [GR97b], [GRS98a], [LCVH92b], [CH92a], [CH95], [Mar96], [Mar99], [MS93], etc

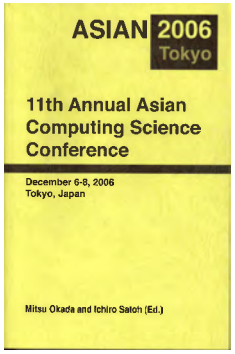
Broadening the range of practical applications

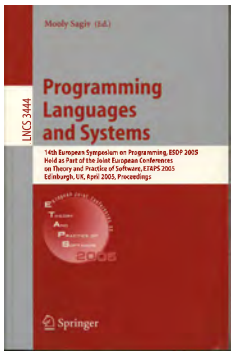
Abstraction, approximation and false alarms

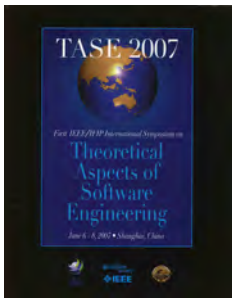
- Brute-force exhaustive exploration of the behavior of complex systems does not scale up
- Sound (and complete) abstractions and/or approximations are potential keys to understanding, specification, design and verification of complex systems
- By undecidability, abstraction yields false alarms
- False alarms in automatic verification can be reduced thanks to domain specific abstractions
- Instead of refining a weak invariant for a program, we refine the abstraction for a family of programs

Astrée [BCC+02, BCC+03, CCF+05, CCF+06, CCF+07]









- Started in Nov. 2001
- Success due to its well-focused scope (proof of absence of runtime errors in synchronous control/command software) ⇒ no false alarm [DS07]
- Mandatory in the development of future flight control systems?


P. Cousot


R. Cousot


J. Feret


L. Mauborgne

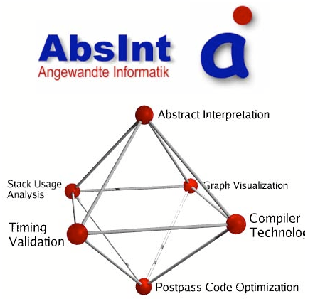

A. Miné


X. Rival

B. Blanchet, Nov. 2001 — Nov. 2003, D. Monniaux, Nov. 2001 — Aug. 2007.

Industrialization

Difficulties of industrialization



- Program verification is difficult, without immediate profitability, hence not an imperative priority
- Research concentrates on a few fashionable short-term subjects and methods [27]
- There very few open source or directly industrializable academic prototypes [28]

- The software industry lacks the competence [29]
- There are too few engineers in the program verification field [30]
- Nevertheless partly successful [DS07, LMR+98, RSD00, Sou04]

---

[26] hoping for rapid solutions that do not last e.g. by lack of scalability

[27] which development is hardly compatible with the short term individual evaluation of academic researchers

[28] but within a few research centers with marginal influence on the companies practices

[29] CS curricula emphasize massive empirical software production more than software quality

Examples of challenges

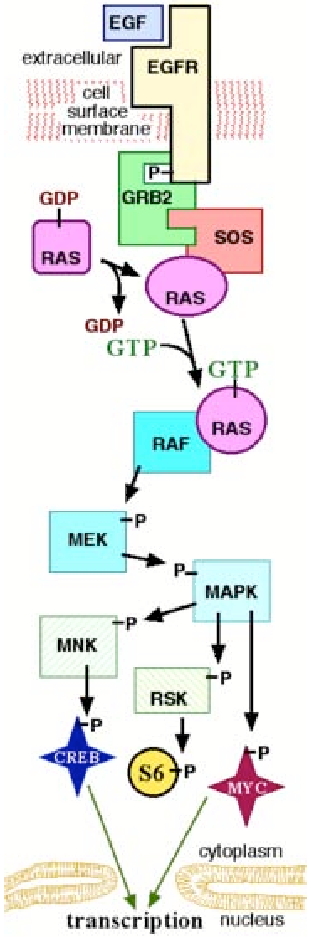
Applications of static analysis

Various kinds of applications:

1. Unsound: easy but ultimately unconvincing (e.g. testing)
2. Sound, with acceptance threshold: below a certain threshold, false alarms are accepted (e.g. compiler optimization, WCET, security checks, etc.)
3. Sound, without acceptance threshold: false alarms are unacceptable (e.g. program verification)

Which one should be privileged?

Biological systems

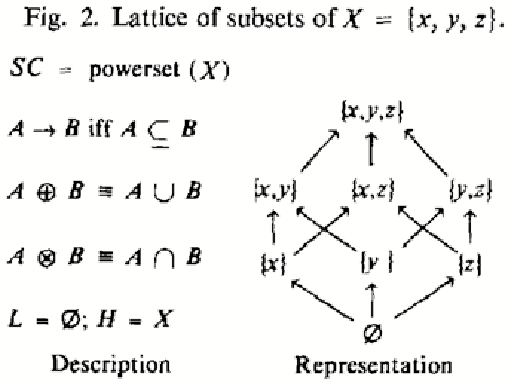


– Biological systems can be described by transition systems [Cou78], hence subject to static analysis in order to predict their evolution [DFFK07, DFF+07]

– More generally, raise the interesting question of abstraction of qualitative models by quantitative models (e.g. abstraction of behaviors by ODEs, performance analysis, etc)

> Le concept de système dynamique discret est évidemment très général. Il s'applique aussi bien aux systèmes informatiques qu'économiques ou biologiques, à condition que le modèle du système étudié soit à évolution discrète dans le temps. En particulier, les systèmes dynamiques discrets sont des modèles des programmes aussi bien séquentiels que parallèles.

Security systems



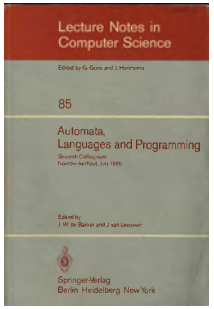
- Verification of security protocols [Bla01, Bla05]
- language-based security [GM04]
- Static analysis is the only way to ensure safe and precise Access-Control Policies [31] [Den76, DD77, PBN07]
- "You are sitting on a gold mine" [32]

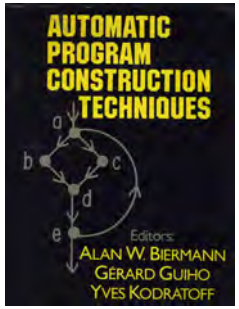
[30] testing is unsafe because of covert channels (channels of information not intended for normal communication, but still revealing protected information), the branch not taken problem, etc.

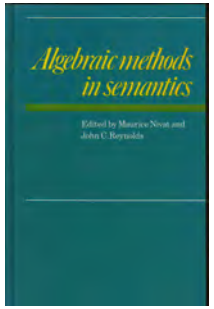
[31] Fred B. Schneider, *Language-Based Security: What's Needed and Why*, Invited Talk, SAS '2001, Paris

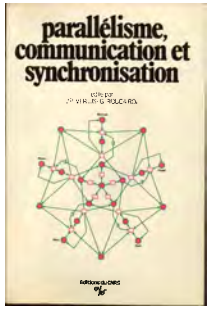
Parallelism

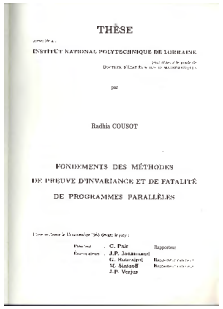
– In principle, the proof [CC85a, CC85b] and analysis [CC80, CC84] of parallel programs is theoretically solved:
  - Analysis of concurrent sequential processes cooperating by synchronous messages [CC80, Mer91]
  - Analysis of concurrent processes sharing global variables [CC84]
– Because of (fair, scheduled) interleaving, it is extremely difficult to scale up!



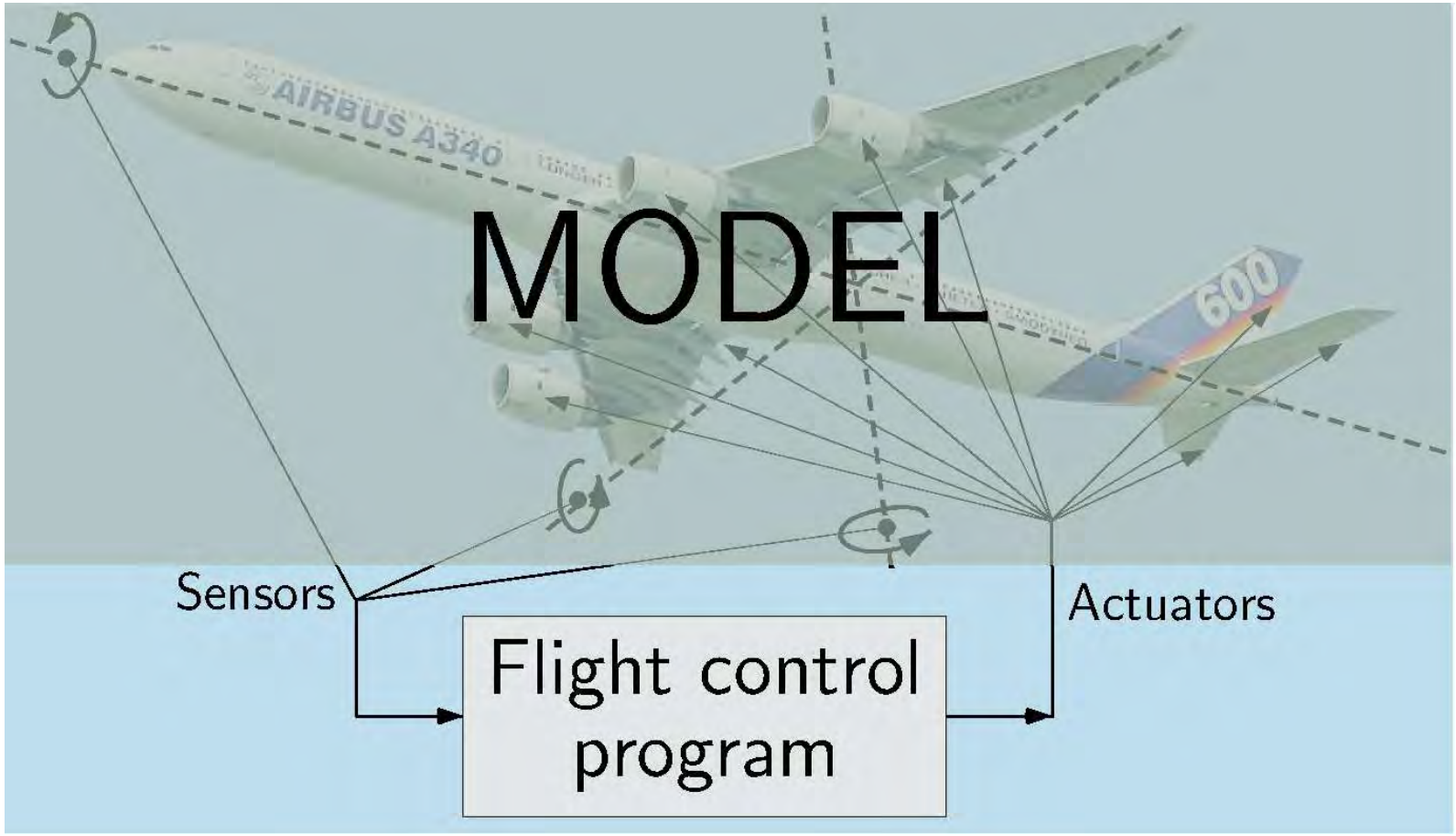








System analysis & verification



Abstractions: program + system

6. Conclusion

Programming

– The evolution of programming languages and programming assistance systems has greatly helped to considerably speed up the development and scale up the size of conceivable programs
– Software quality remains much far beyond, essentially because it is anti-economical
– . . . until the next catastrophy, evolution of the standards, revolution of the customers, or new laws holding computer scientists accountable for bugs

Formal methods

- Formal methods might then become profitable at every stage of program design
- The winners, if any, will definitely have to scale up, at a reasonable cost
- Up to now, research has mainly concentrated on easy avenues with short-term rewards
- Small groups cannot make it, large groups fail to share common interests
- There is still a long long way to go

Abstract interpretation

– Beyond programming, abstraction is the only way to apprehend complex systems
– Therefore, the scope of application of abstract interpretation ideas is large
– Over 30 years, abstract interpretation theory, practice and achievements have grown despite trends and evanescent applications
– Hopefully, abstract interpretation will continue to be useful in the future

THE END

Many thanks to all of you
who contributed to abstract interpretation!