« Proving the Absence of Run-Time Errors in Safety-Critical Avionics Code »

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

Embedded Systems Week, International Conference on Embedded Sofware — Salzburg, Austria
September 30th, 2007

Abstract

– Abstract interpretation [1], [2] is a formal method for software verification with significant industrial applications, in particular in avionics [3], [4], [5], [6].

References

[1] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, FR, 21 Mar. 1978.

[2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[3] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE Static Analyzer. http://www.astree.ens.fr/.

[4] É. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: a simple abstract interpreter. In D. Le Métayer, editor, *Proc. 11th ESOP '2002*, Grenoble, FR, LNCS 2305, pages 209–212. Springer, 8–12 Apr. 2002.

[5] F. Randimbivololona, J. Souyris, and A. Deutsch. Improving avionics software verification cost-effectiveness: Abstract interpretation based technology contribution. In *Proceedings DASIA 2000 – DAta Systems In Aerospace*, Montreal, CA. ESA Publications, 22–26 May 2000.

[6] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. Abstract interpretation-based timing validation of hard real-time avionics software. In *Proc. Int. Conf. DSN 2003*, San Francisco, CA, US, pages 625–634. IEEE Comp. Soc. Press, 22–25 June 2003.

Abstract (Cont'd)

– We explain program correctness proofs by static analysis and the design of a static analyzer by abstract interpretation of a program semantics.

– This is illustrated with the ASTRÉE abstract-interpretation-based static analyzer which, after six years of academic development, is progressing towards industrial acceptance for validating software intensive applications such as safety-critical avionics code [7], [8].

References

[7] D. Delmas and J. Souyris. ASTRÉE*: from Research to Industry*. Proc. 14th Int. Symp. SAS '07, G. Filé and H. Riis-Nielson (eds), 22–24 Aug. 2007, Kongens Lyngby, DK, LNCS 4634, pp. 437–451, Springer.

[8] J. Souyris. Industrial experience of abstract interpretation-based static analyzers. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 393–400. Kluwer Acad. Pub., 2004.

Contents

1. The Endless “Software Failure” Problem

Software is hidden everywhere








Origin of accidents (metro)

– Paris métro line 12 accident [1]: the driver was going too fast

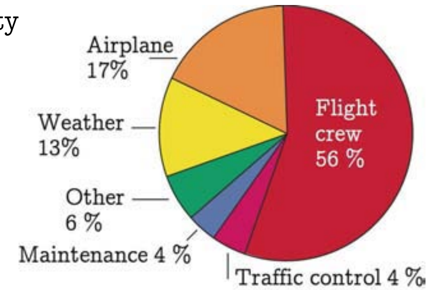
– Roma metro line A accident [2]: the driver went on at a red signal

1 On August 30th, 2000, at the Notre-Dame-de-Lorette métro station in Paris, a car flipped over on its side and slid to a stop just a few feet from a train stopped on the opposite platform (24 injured).

2 On October 17th, 2006, a speeding subway train rammed into another train halted at the Vittorio Emanuele station in central Rome (1 dead, 60 injured). The driver might have misunderstood the control centre authorizing the train to proceed to the "next station" (Manzoni, closed to the public) while the driver would have understood it to mean the "next working station" (Vittorio Emanuele, after Manzoni), *La Repubblica*, Oct. 20th, 2006.

Origin of accidents (aviation)

Worldwide analysis of the primary cause of major commercial-jet accidents betwwen 1995 and 2004 as determined by the investigating authority



Reference

[9] D. Michaels & A. Pasztor. *Incidents Prompt New Scrutiny Of Airplane Software Giltches* citing a Boeing source. Wall Street Journal, Vol. CCXLVII, No 125, 30 mai 2006.

3 includes only accidents with known causes.

Software replaces human operators

– Computer control is the cheapest and safest solution to avoid such accidents

– New high-speed métro line 14 (Météor): fully automated, no operators

– Modern commercial airplanes: massive automation of control/commands, piloting, communications, collision avoidance, etc

As computer hardware capacity grows exponentially since 1975...

ENIAC
5,000 flops[4]

IBM's Blue Gene/P
4 petaflops[5]

4 Floating point operations per second
5 petaflop = 10^{15} flops = one-quadrillion operations per second

Software size grows...

Text editor
1,700,000 lines of C[6]

Operating system
35,000,000 lines of C[7]

6 3 months for full-time reading of the code
7 5 years for full-time reading of the code

... and so does the number of bugs

Text editor
1,700,000 lines of C[6]
1,700 bugs (estimation)

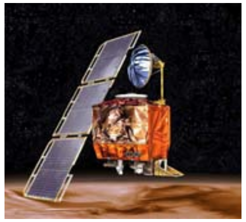
Operating system
35,000,000 lines of C[7]
30,000 known bugs

6 3 months for full-time reading of the code
7 5 years for full-time reading of the code

All Computer Scientists Have Experienced Bugs

Ariane 5.01

Patriot



Mars orbiter

Mars Global Surveyor

Example 1: Overflow

Computers are finite

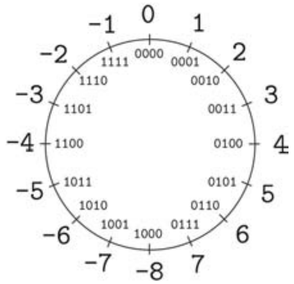
- Scientists reason on continuous, infinite mathematical structures (e.g. $\mathbb{R}$)
- Computers can only handle discrete, finite structures

Overflows

- Numbers are encoded onto a limited number of bits (*binary digits*)
- Some operations may overflow (e.g. integers: 32 bits $\times$ 32 bits $=$ 64 bits)
- Using different number sizes (32, 64, ... bits) can also be the source of overflows

Modular integer arithmetics...

- Todays, computers avoid integer overflows thanks to modular arithmetic
- Example: integer 2's complement encoding on 8 bits



Modular arithmetics is not very intuitive (cont'd)

In C:

```
% cat -n modulo-c.c
     1 #include <stdio.h>
     2 int main () {
     3 int x,y;
     4 x = -2147483647 / -1;
     5 y = ((-x) -1) / -1;
     6 printf("x = %i, y = %i\n",x,y);
     7 }
     8
% gcc modulo-c.c
% ./a.out
x = 2147483647, y = -2147483648
```


Static Analysis with ASTRÉE

```
% cat -n modulo.c
     1 int main () {
     2 int x,y;
     3 x = -2147483647 / -1;
     4 y = ((-x) -1) / -1;
     5 __ASTREE_log_vars((x,y));
     6 }
     7
% astree -exec-fn main -unroll 0 modulo.c\
 |& egrep -A 1 "(<integers)|(WARN)"
modulo.c:4.4-18::[call#main@1:]: WARN: signed int arithmetic range
  {2147483648} not included in [-2147483648, 2147483647]
  <integers (intv+cong+bitfield+set): y in [-2147483648, 2147483647] /\ Top
   x in {2147483647} /\ {2147483647} >
```

ASTRÉE signals the overflow and goes on with an unkown value.

Float Arithmetics does Overflow

In C:

```
% cat -n overflow.c
 1  void main () {
 2  double x,y;
 3  x = 1.0e+256 * 1.0e+256;
 4  y = 1.0e+256 * -1.0e+256;
 5  __ASTREE_log_vars((x,y));
 6  }
% gcc overflow.c
% ./a.out
x = inf, y = -inf
```

```
% astree -exec-fn main
overflow.c |& grep "WARN"
overflow.c:3.4-23::[call#main1:]:
WARN: double arithmetic range
[1.79769e+308, inf] not
included in [-1.79769e+308,
1.79769e+308]
overflow.c:4.4-24::[call#main1:]:
WARN: double arithmetic range
[-inf, -1.79769e+308] not
included in [-1.79769e+308,
1.79769e+308]
```

The Ariane 5.01 maiden flight

- June 4th, 1996 was the maiden flight of Ariane 5

The estimated cost of an overflow

- **500 000 000 $;**
- Including indirect costs (delays, lost markets, etc):
  2 000 000 000 $;
- The financial results of Arianespace were **negative** in 2000, for the first time since 20 years.
- At the origin of the development of PolySpace Verifier, (now part of The MathWorks) [9]

[9] The cost of development of the PolySpace Verifier was 10.000.000 $ (Daniel Pilaud ipse dixit).

The Ariane 5.01 maiden flight failure

- June 4th, 1996 was the maiden flight of Ariane 5
- The launcher was destroyed after 40 seconds of flight because of a software overflow [8]

[8] A 16 bit piece of code of Ariane 4 had been reused within the new 32 bit code for Ariane 5. This caused an uncaught overflow, making the launcher uncontrolable.

PolySpace Verifier

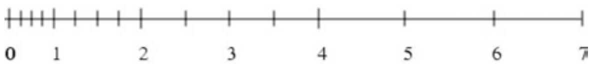
Example 2: Rounding

Mapping many to few

– Reals are mapped to floats (floating-point arithmetic)

$$\pm d_0.d_1d_2\ldots d_{p-1}\beta^e \ ^{(*)}$$

– For example on 6 bits (with $p = 3$, $\beta = 2$, $e_{\mathrm{m\ n}} = -1$, $e_{\max} = 2$), there are 32 normalized floating-point numbers. The 16 positive numbers are



(*) where
- $d_0 \neq 0$,
- p is the number of significative digits,
- β is the basis (2), and
- e is the exponant ($e_{\mathrm{m\ n}} \leq e \leq e_{\max}$)

Rounding

– Computations returning reals that are not floats, must be rounded

– Most mathematical identities on $\mathbb{R}$ are no longer valid with floats

– Rounding errors may either compensate or accumulate in long computations

– Computations converging in the reals may diverge with floats (and ultimately overflow)

Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

$$(x + a) - (x - a) \neq 2a$$

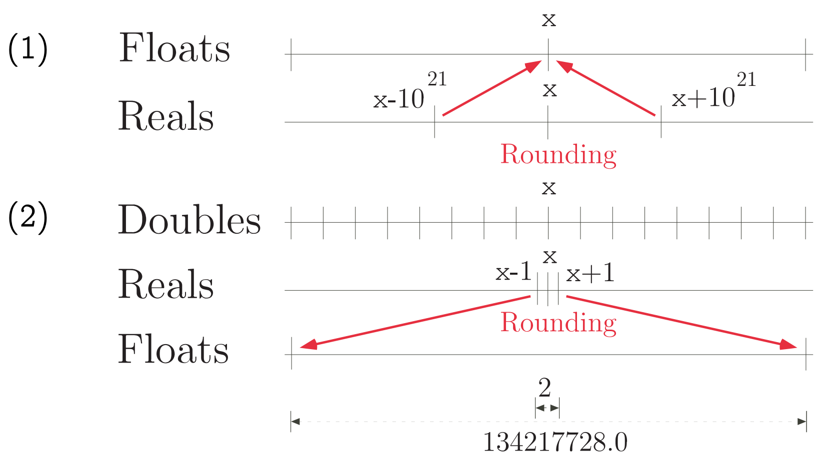
Example of rounding error

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951487.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
0.000000
```

$(x + a) - (x - a) \neq 2a$

Explanation of the huge rounding error



Static analysis with ASTRÉE[10]

```
% cat -n double-error.c
 2  int main () {
 3  double x; float y, z, r;;
 4  /* x = ldexp(1.,50)+ldexp(1.,26); */
 5  x = 1125899973951488.0;
 6  y = x + 1;
 7  z = x - 1;
 8  r = y - z;
 9  __ASTREE_log_vars((r));
10  }
% gcc double-error.c
% ./a.out
134217728.000000
% astree -exec-fn main -print-float-digits 10 double-error.c |& grep "r in
direct = <float-interval: r in [-134217728, 134217728] >
```

[10] ASTRÉE makes a worst-case assumption on the rounding ($+\infty$, $-\infty$, 0, nearest) hence the possibility to get -134217728.

Example of accumulation of small rounding errors

```
% cat -n rounding-c.c
 1  #include <stdio.h>
 2  int main () {
 3   int i; double x; x = 0.0;
 4   for (i=1; i<=1000000000; i++) {
 5    x = x + 1.0/10.0;
 6   }
 7  printf("x = %f\n", x);
 8  }
% gcc rounding-c.c
% ./a.out
x = 99999998.745418
%
```

since $(0.1)_{10} = (0.0001100110011001100\ldots)_2$

Static analysis with ASTRÉE

```
% cat -n rounding.c
     1  int main () {
     2   double x; x = 0.0;
     3   while (1) {
     4    x = x + 1.0/10.0;
     5    __ASTREE_log_vars((x));
     6    __ASTREE_wait_for_clock(());
     7   }
     8  }
% cat rounding.config
 __ASTREE_max_clock((1000000000));
% astree -exec-fn main -config-sem rounding.config -unroll 0 rounding.c\
 |& egrep "(x in)|(\|x\|)|(WARN)" | tail -2
direct = <float-interval: x in [0.1, 200000040.938] >
  |x| <= 1.*((0. + 0.1/(1.-1))*(1.)^clock - 0.1/(1.-1)) + 0.1
      <= 200000040.938
```


The Patriot missile failure

– "On February 25th, 1991, a Patriot missile … failed to track and intercept an incoming Scud (*)."

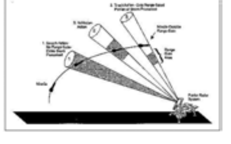
– The software failure was due to accumulated rounding error (†)

(*) This Scud subsequently hit an Army barracks, killing 28 Americans.

(†) – "Time is kept continuously by the system's internal clock in tenths of seconds"

– "The system had been in operation for over 100 consecutive hours"

– "Because the system had been on so long, the resulting inaccuracy in the time calculation caused the range gate to shift so much that the system could not track the incoming Scud"



Other Examples

The NASA's Climate Orbiter Loss on September 23, 1999

– A metric confusion error led to the loss of NASA's $125 million, Lockheed Martin built Mars Climate Orbiter on September 23, 1999 [11]

– "People sometimes make errors," said Edward Weiler, NASA's Associate Administrator for Space Science in a written statement. "The problem here was not the error, it was the failure of NASA's systems engineering, and the checks and balances in our processes to detect the error. That's why we lost the spacecraft."

11 Erroneous information was transmitted from the Mars Climate Orbiter spacecraft team in Colorado and the mission navigation team in California. One engineering team used metric units while the other used English units! The navigation mishap pushed the spacecraft dangerously close to the planet's atmosphere where it presumably burned and broke into pieces.

Is the metric system better?

```
while (1) {
   ...
   /* x in meters */
   x = x * 100.0;
   /* x in centimeters */
   ...
   x = x / 100.0;
   /* back to  x in meters */
   ...
}
```

Scaling in general can be the source of cumulated rounding errors.

Static Analysis with ASTRÉE

```
% cat -n scale.c
 1 int main () {
 2  float x; x = 0.70000001;
 3  while (1) {
 4   x = x / 3.0;
 5   x = x * 3.0;
 6   __ASTREE_log_vars((x));
 7   __ASTREE_wait_for_clock(());
 8  }
 9 }
```

```
% gcc scale.c
% ./a.out
x = 0.699999988079071
```

```
% cat scale.config
 __ASTREE_max_clock((1000000000));
% astree -exec-fn main -config-sem scale.config -unroll 0 scale.c\
 |& grep "x in" | tail -1
direct = <float-interval: x in [0.699999868870, 0.700000047684] >
%
```


NASA's Mars Global Surveyor (MGS) Loss in Nov. 2006

– The latest theory is that a software error caused the failure. In June 2006, ground controllers uploaded software to the wrong location [12].

– "Basically, the computer got confused," said Doug McCuistion, NASA's director of Mars Exploration in Washington, DC, US. But he cautions, "This is a preliminary thought."

[12] The error could have caused the solar array to move to the wrong spot when NASA commanded it to move on 2 November. The spacecraft told ground controllers that it was having a problem with the motor that moves the array. The spacecraft later went into a "safe" mode, where it awaited further instructions from Earth. In this mode, the spacecraft may have pointed one side of itself towards the Sun, overheating one of the coolers for a battery. This in turn could have caused a battery failure, meaning the spacecraft would not have enough power to survive.

Bugs Now Show-Up in Everyday Life

– Bugs now appear frequently in everyday life (banks, cars, telephones, ...)

– Example (HSBC bank ATM [13] at 19 Boulevard Sébastopol in Paris, failure on Nov. 21st 2006 at 8:30 am):

[13] cash machine, cash dispenser, automatic teller machine.

2. What can be done about bugs?

Warranty

Excerpt from an GPL open software licence:

> NO WARRANTY. . . . *BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.*

You get nothing for free!

Warranty

Excerpt from Microsoft software licence:

> DISCLAIMER OF WARRANTIES. . . . *MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE, AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND MICROSOFT AND ITS SUPPLIERS HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. . . .*

You get nothing for your money either!

Consequences of the Absence of Legal Warranty on Software

- No one is legally responsible for bugs
- So, no one cares about software validity
- Bugs can even be the source of revenues (customers must buy the next version to get around bugs in software)
- End-users cannot put any pressure on computer scientists responsible for errors at the origin of catastrophies [14]
- The general public might lose confidence in software-based technology [15]

[14] Contrary e.g. to car users.
[15] this is one of the explanations for the success of open software

Why is there no Legal Warranty on Software

– The state of the art is that all bugs cannot be eliminated *at a reasonable price*
– The law cannot enforce more than "best professional practice"

Improving the State of the Art

– The only hope is therefore to improve the state of the art of developping bug-free software
– The state of the art can be enforced by quality norms to be followed by professionals (such as DO178B)
– The law will then follow such "best professional practices"

Software Engineering

– Software engineering has focussed on methods for designing larger and larger, more and more complex computer applications (e.g. OO languages)
– Software quality has definitely not followed this dramatic progression
– Contray to other disciplines, software engineering offer very few tools (either *intellectual tools* such as specification and programming languages or *instruments* such as macro-assemblers, interpreters, compilers, make, code repositories, software managing platforms, etc).

Process-based Software Qualification

– The state of the art in software qualification is development process-based (e.g. DO178B level A)
  - the software has been produced using qualified tools (simulators, test execution tool, coverage tools, reporting tools, etc.)
  - the software has been tested according to well-established processes (code reviews, unit testing, integration testing, system testing, etc.)
  - the software qualification has been documented and approved by certification authorities

Example: DO-178B

- DO-178B (Software Considerations in Airborne Systems and Equipment Certification) is a standard for software development published by RTCA, Incorporated (Radio Technical Commission for Aeronautics).
- The standard dating Dec. 1, 1992, was developed by RTCA and EUROCAE (European Organisation for Civil Aviation Equipment).
- The FAA (Federal Aviation Administration) [16] accepts use of DO-178B for certifying software in avionics.

[16] In Europ, EASA (European Aviation Safety Agency), JAA (Joint Aviation Authorities) and CAA (Civil Aviation Authority).

Example: DO-178B (Cont'd)

- The ARP4761 safety assessment process and hazard analysis examining the effects of a failure condition in the system determines software levels;
- Depending on the software levels a number of objectives must be satisfied, some with independence (the person(s) who verify an objective must not be the developers of the item in question)
- In some cases, an automated tool may be equivalent to independence [17]

[17] RTCA/DO-178B "Software Considerations in Airborne Systems and Equipment Certification", p.82

Example: DO-178B (Cont'd)

A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
C	Major	Failure is significant, but has a lesser impact (e.g. passenger discomfort but no injuries).
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (e.g. causing passenger inconvenience or a routine flight plan change).
E	No effect	Failure has no impact on safety.

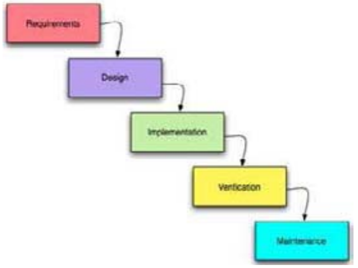
Example: DO-178B (Cont'd)

Processes	Documents
Planning	Plan for software aspects of certification (PSAC), Software development plan (SDP), Software verification plan (SVP), ...
Development	Software design description (SDD), Source code, Executable object code, ...
Verification	Software verification results (SVR): Reviews, Tests, Code coverage analysis, ...
Configuration management	Software life cycle environment configuration index (SECI), ...
Quality assurance	Software conformity review (SCR), ...
Certification liaison	a Designated Engineering Representative (DER) working for e.g. FAA in an airplane manufacturing company.

The Waterfall Software Development Process

The following phases are followed perfectly in order [18]:

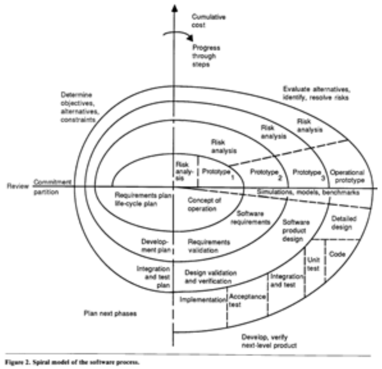
1. Requirements specification
2. Design
3. Construction (aka: coding or implementation)
4. Integration
5. Testing and debugging (aka: validation)
6. Installation
7. Maintenance



[18] Winston Royce, "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON 26(August 1970): 1-9.

The Spiral Lifecycle Software Development Process

– Iterative devlopment of software proptotypes [19], each one using the waterfall model.
– Iteration until the customer is satisfied that the refined prototype represents the final product desired.

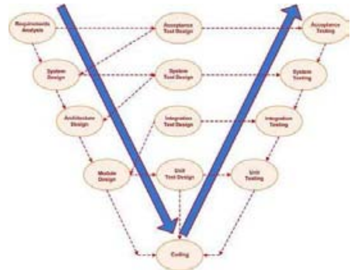


[19] Barry Boehm. A Spiral Model of Software Development and Enhancement, Computer, pp. 61–72, 1988.

The V Software Development Process

The V-Model or VEE-Model was developped by the german administration [20]:

– A decomposition of requirements phase (↘): Requirements analysis, System Design, Architecture Design, Module Design, Coding,
– A validation phase (↗): Unit Testing, Integration Testing, System Testing, User Acceptance Testing.
– In practice, design, implementation, and testing are mixed.



[20] Systems Engineering for Intelligent Transportation Systems, pp 10., US Dept. of Transportation, Jan. 2007

Guarantees Offered by Software Qualification

– Process-based software qualification is taxonomic and organizational but has no scientific rigor
– Process-based software qualification offers high (self)-confidence but no correctness guarantee
– Malfunctionning software can be/has been easily qualified
– No significative progress of Software Engineering this last decade despite many variants of the “Software Qualification Models”.

Hamburger Model [21]

[21] Students write reports containing the three parts of the hamburger model: top bun = introduction, patty I, patty II, patty III, bottom bun = conclusion. The same structure was proposed for building design, hardware development!.

A Strong Need for Software Better Quality

– Poor software quality is not acceptable in safety and mission critical software applications.




– The present state of the art in software engineering does not offer sufficient quality garantees

Tool-Based Software Design Methods

– New tool-based software design methods will have to emerge to face the unprecedented growth and complexification of critical software

– E.g. FCPC (Flight Control Primary Computer)
  - A220: 20 000 LOCs,
  - A340:
    130 000 LOCS (V1),
    250 000 LOCS (V2),
  - A380: 1.000.000 LOCS

Product-based Software Qualification

– An avenue is therefore opened for formal methods which are product-based

The software is shown to satisfy a specification

– Main approaches:
  - theorem-proving & proof checking
  - model-checking
  - static analysis

Bug Finding versus Bug Absence Proving

– Bug-finding methods : unit, integration, and system testing, dynamic verification, bounded model-checking, error pattern mining, . . .

$\rightarrow$ Helpful but very partial

– Absence of bug proving methods : formally prove that the semantics of a program satisfies a specification

$\rightarrow$ Successful in the small but must scale up in the large

Success Story 1: the B-Method

– 80 000 LOCs C safety-control software embedded in trains of the Paris Métro Line 14 was specified and proved correct using Abrial's B-method [10, 11]

– No bug found in operation in the parts proved in B [22]

– Not required for the future automatization of the metro line A in Paris (high cost does not compete at call-time with light-weight informal methods)

Reference

[10] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996. ISBN 0-521-49619-5.

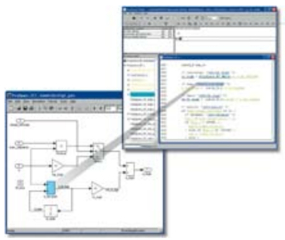
[11] Patrick Behm, Paul Benoit, Alain Faivre, Jean-Marc Meynadier. *Météor: A Successful Application of B in a Large Project.* In *Proc. FM'99 - Formal Methods: World Congress on Formal Methods in the Development of Computing Systems, Vol. I*, J.M. Wing, J. Woodcock, J. Davies (Eds.), Toulouse, France, Sep. 1999, LNCS 1708, Springer, pp. 369–387.

[22] when interface requirements were satisfied.

Success Story 2: PolySpace Verifier

– The PolySpace Verifier [23] of PolySpace Technologies (now part of The MathWorks) aims at proving the absence of runtime errors in Ada and C/C++ code at compile-time (without requiring code modification or execution or test cases)

– Typical run-time errors detected:
  - Overflows and underflows
  - Division by zero and other arithmetic errors



[23] Now split in the PolySpace™ Server for C/C++ or Ada static analyzer and PolySpace™ Client for C/C++ or Ada to view server results

  - Out-of-bounds array accesses
  - Read-only accesses to noninitialized data
  - Illegally dereferenced pointers
  - Dangerous type conversions
  - Dead code

– Mostly successful in the embedded software industry [24]

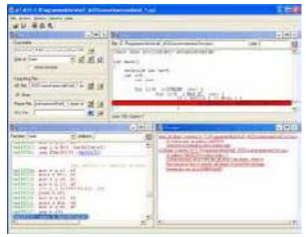
– Many false alarms (typically 10% of the runtime-checks cannot be proved valid at compile time), mostly useful for bug-finding

– Certification is only partial (but without any bug omission)

[24] PolySpace Model Link SL extends PolySpace™ Client for C/C++ and PolySpace™ Server for C/C++ with tools to trace PolySpace™ results from generated C code directly to the SIMULINK™ model.

Success Story 3: aiT WCET Analyzers

– The aiT WCET Analyzers of AbsInt, statically compute tight bounds for the worst-case execution time (WCET) of tasks in real-time systems



– Analyze binary executables and take the intrinsic cache and pipeline behavior into account

– Widely used in the embedded software industry (like Airbus France)

Problems with Formal Methods

– Formal specifications (abstract machines, temporal logic, ...) are costly, complex, error-prone, difficult to maintain, not mastered by casual programmers
– Formal semantics of the specification and programming language are inexistant, informal, irrealistic or complex
– Formal proofs are partial (static analysis), do not scale up (model checking) or need human assistance (theorem proving & proof assistants)
– High costs (for specification, proof assistance, etc).

Avantages of Static Analysis

– Formal specifications are implicit (no need for explicit, user-provided specifications)
– Formal semantics are approximated by the static analyzer (no user-provided models of the program)
– Formal proofs are automatic (no required user-interaction)
– Costs are low (no modification of the software production methodology)
– Scales up to 100.000 to 1.000.000 LOCS
– Large diffusion in embedded software production industries

Disadvantages of Static Analysis and Remedies

– Imprecision (acceptable in some applications like WCET or program optimization)
– Incomplete for program verification
– False alarms are due to unsuccessful automatic proofs in 5 to 15% of the cases
  → specialization to specific program properties [25]
  → specialization to specific families of programs [26]
  → possibility of refinement [27]

[25] For example, ASTRÉE is specialized for runtime errors
[26] For example, ASTRÉE is designed for the proof of runtime-errors in real-time synchrnonous control/command programs
[27] For example, ASTRÉE offers parametrizations and analysis directives

The Impact of Tools

– Research is presently changing the state of the art (e.g. ASTRÉE)
– We can check for the absence of large categories of bugs (may be not all of them but a significant portion of them, such as runtime errors)
– The verification can be made automatically by mechanical tools
– Some bugs can be found completely automatically, without any human intervention
– Whence responsabilities can be established a posteriori, by reanalyzing the failed sofware

Towards a New State of the Art

- The state of the art will change towards complete automation, at least for common categories of bugs
- Whence the standards will change (by adjusting to the new state of the art)
- Whence the law will change (by adjusting to the new standards)
- So responsabilities can be established (at least for automatically detectable bugs)
- To ensure at least partial software verification.

3. Informal Introduction to Abstract Interpretation

Mathematics and computers can help

- Software behavior can be mathematically formalized $\rightarrow$ semantics
- Computers can perform semantics-based program analyses to realize verification $\rightarrow$ static analysis
  - but computers are finite so there are intrinsic limitations $\rightarrow$ undecidability, complexity
  - which can only be handled by semantics approximations $\rightarrow$ abstract interpretation

Abstract Interpretation

There are two fundamental concepts in computer science (and in sciences in general) :

- **Abstraction** : to reason on complex systems
- **Approximation** : to make effective undecidable computations

These concepts are formalized by abstract interpretation

References

[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *ACM POPL*.

[Thesis '78] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse ès sci. math. Grenoble, march 1978.

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *ACM POPL*.

Applications of Abstract Interpretation

- Static Program Analysis [CC77a], [CH78], [CC79] including Dataflow Analysis; [CC79], [CC00], Set-based Analysis [CC95], Predicate Abstraction [Cou03], . . .
- Grammar Analysis and Parsing [CC03];
- Hierarchies of Semantics and Proof Methods [CC92b], [Cou02];
- Typing & Type Inference [Cou97];
- (Abstract) Model Checking [CC00];
- Program Transformation (including program optimization, partial evaluation, etc) [CC02b];

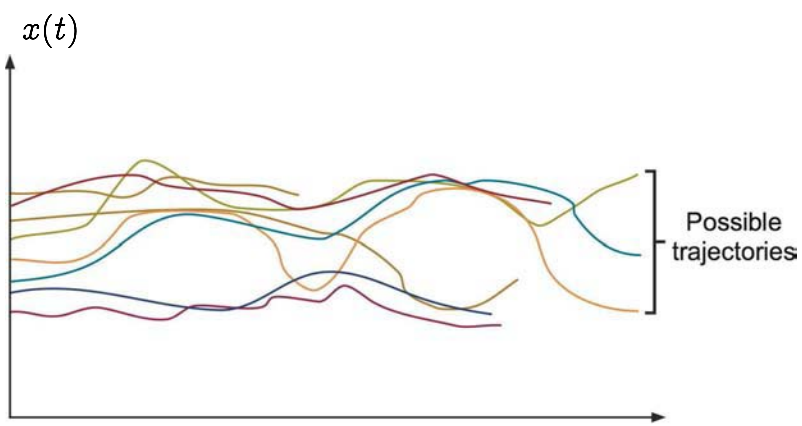
Applications of Abstract Interpretation (Cont'd)

- Software Watermarking [CC04];
- Bisimulations [RT04];
- Language-based security [GM04];
- Semantics-based obfuscated malware detection [PCJD07].
- Databases [AGM93, BPC01, BS97]
- Computational biology [Dan07]
- Quantum computing [JP06, Per06]

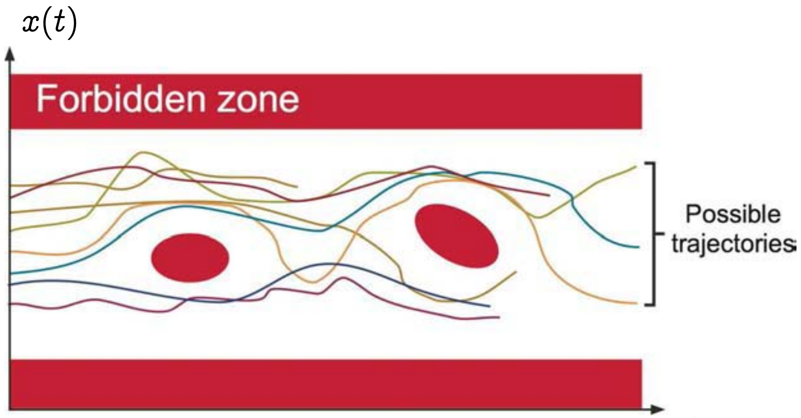
All these techniques involve sound approximations that can be formalized by abstract interpretation

Approximation

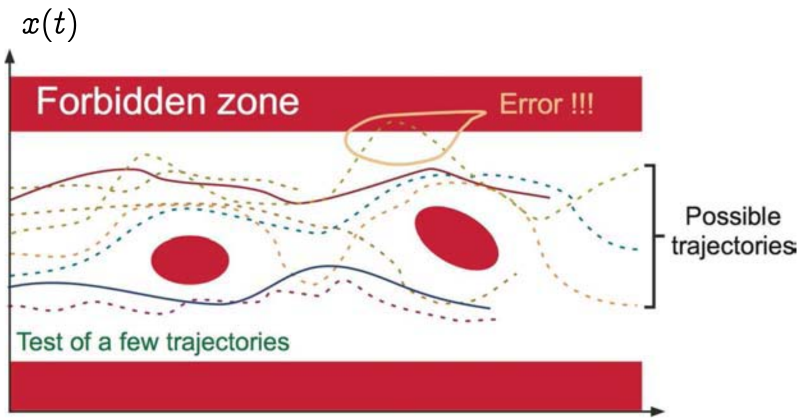
Operational semantics



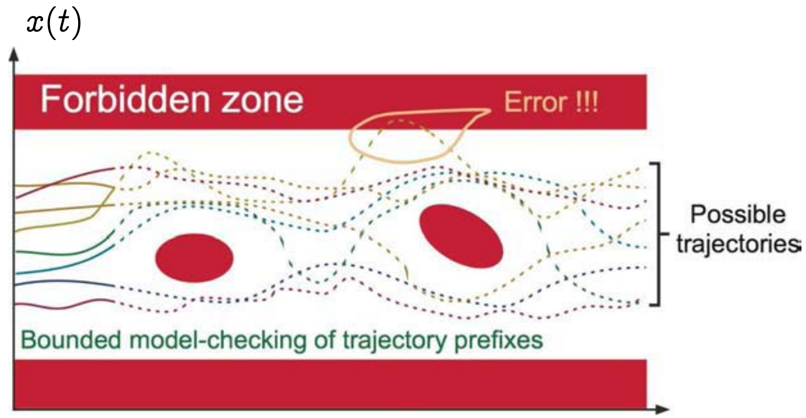
Safety property



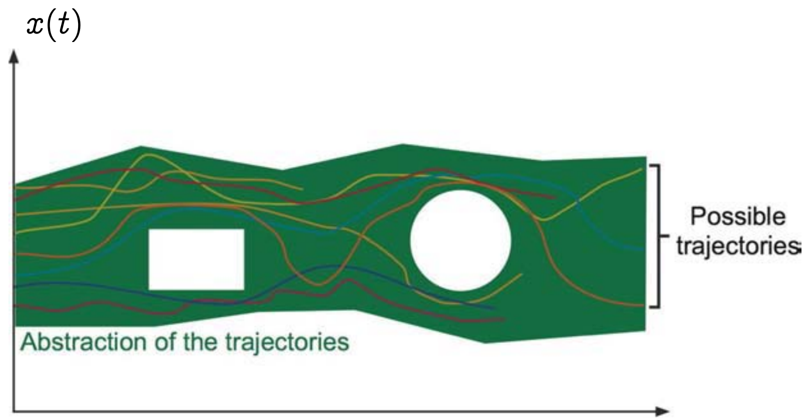
Test/Debugging is Unsafe



Bounded Model Checking is Unsafe



Over-Approximation



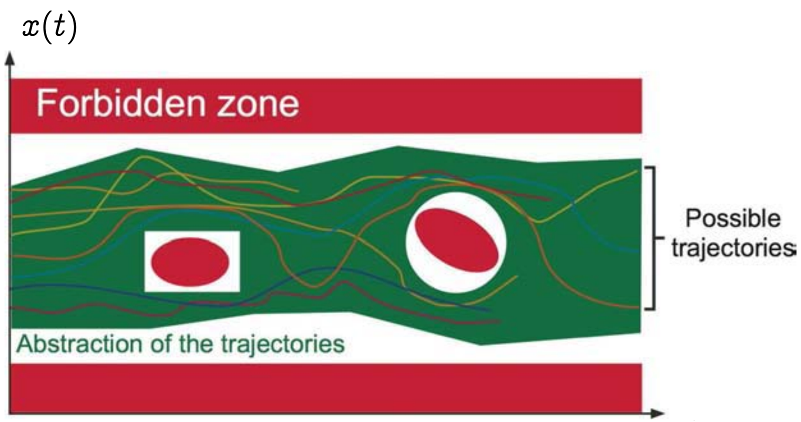
Over-Approximation (Cont'd)

– In general the set of traces of interest is neither computer-representable nor computable (by undecidability).

– Abstract interpretation exploits the facts that an *over-approximation* of the program set of traces is sound: when considering more possibilities, no actual execution can ever be omitted.

– The theory is used to design sound approximations of the mathematical structures involved in the formal description of this set of traces.

Abstract Interpretation is Sound



Correctness Proof

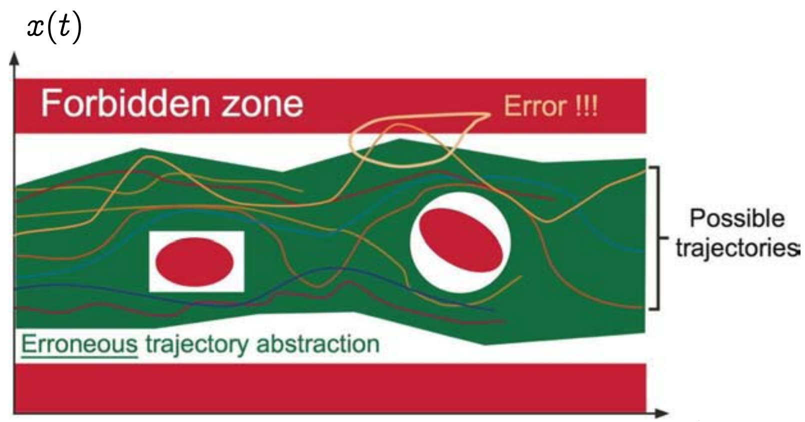
The *correctness proof* has two phases.

– In the first *analysis phase*, the program trace semantics is computed iteratively. [28]

– The *verification phase* then checks that none of these execution traces can reach a state in which a runtime error can occur.

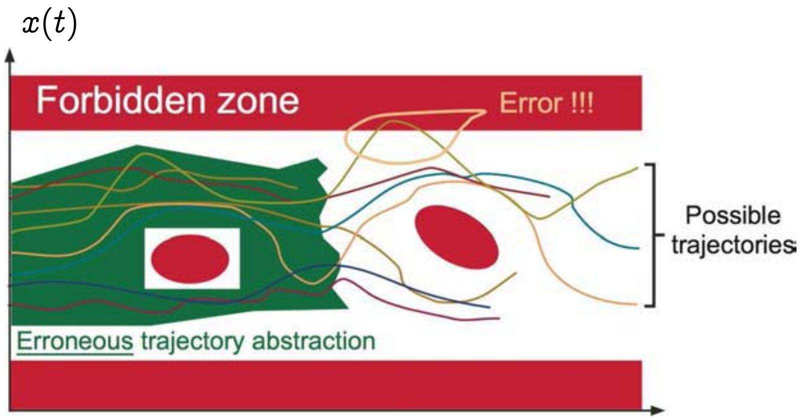
[28] From a purely mathematical point of view, the set of all execution traces can in principle be formally constructed starting from initial states, then extending iteratively the partial traces from one state to the next one according to the program transition steps until termination on final or error states or passing to the limit for infinite traces (corresponding to non-terminating executions).

Soundness Requirement: Erroneous Abstraction [29]



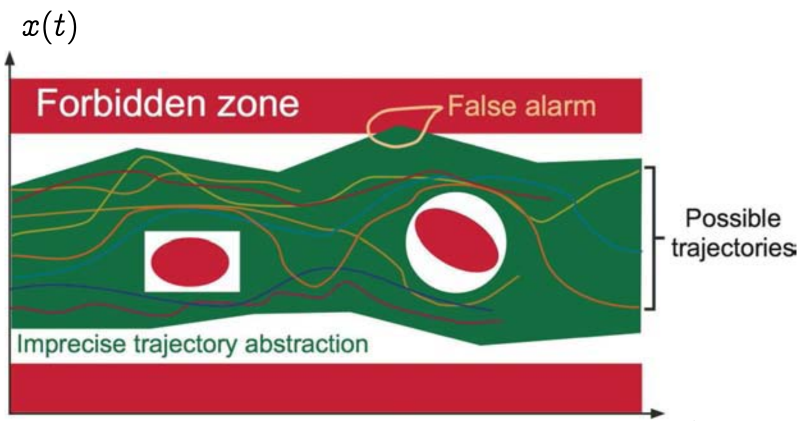
[29] This situation is always excluded in static analysis by abstract interpretation.

Soundness Requirement: Erroneous Abstraction[30]

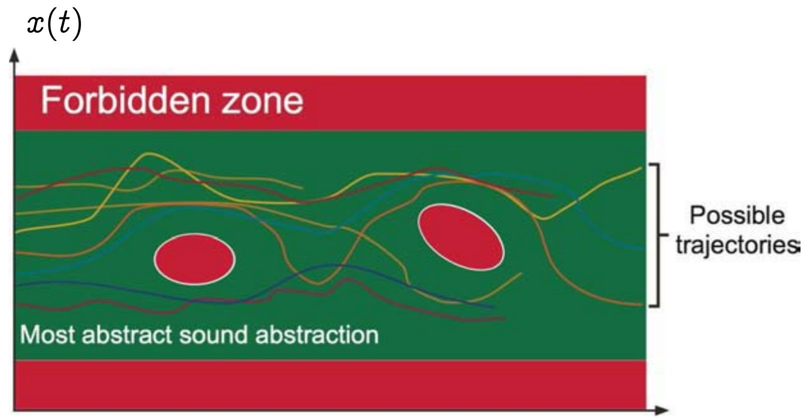


[30] This situation is always excluded in static analysis by abstract interpretation.

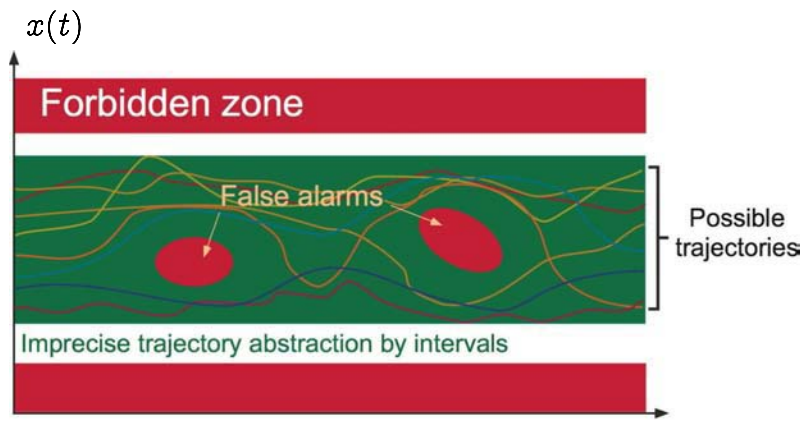
Imprecision ⇒ False Alarms



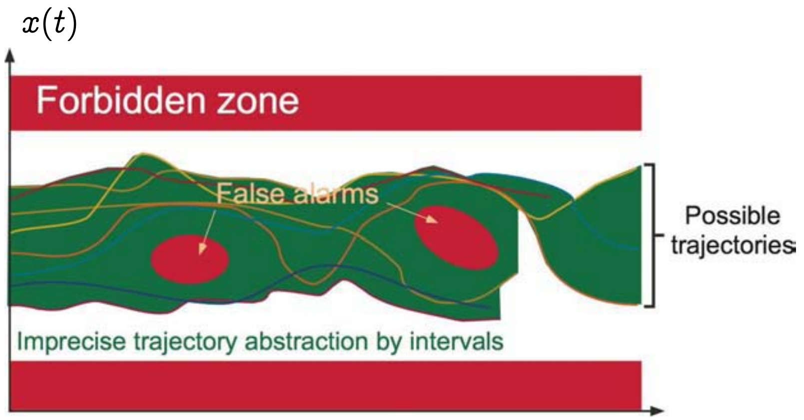
The Most Abstract Sound Abstraction



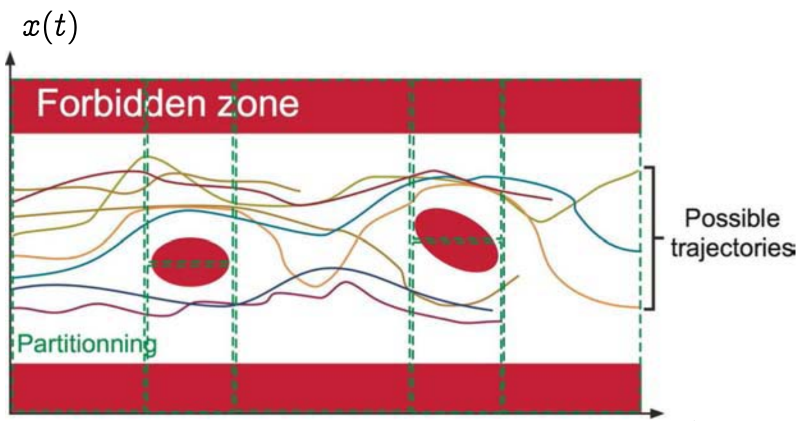
Global Interval Abstraction → False Alarms



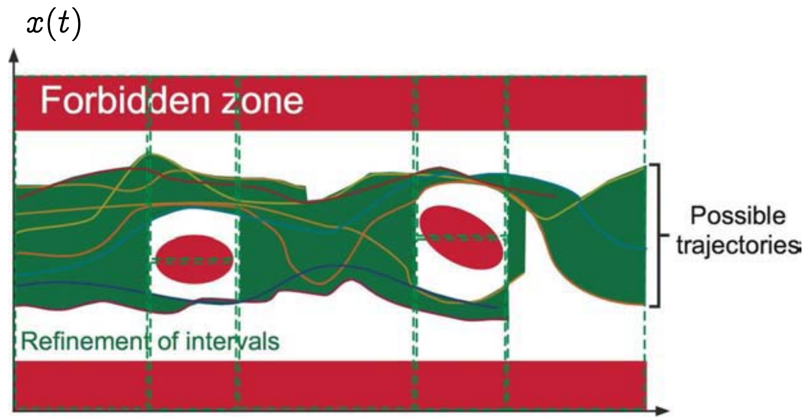
Local Interval Abstraction → False Alarms



Refinement by Partitionning



Intervals with Partitionning



Iterator and Abstract Domains

Iterator and Abstract Domains

– an *iterator* for approximating the step by step iterative computation of traces [12], and

– *abstract domains* representing the effect of program steps and passage to the limit (widening/narrowing [12]).

References

[12] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

A small graphical language

– objects;

– operations on objects.

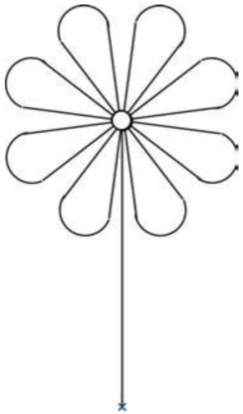
Objects

An object is a pair:

– an origin (a reference point $\times$);

– a finite set of black pixels (on a white background).

Example of an object: a flower



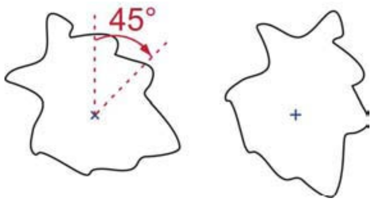
Operations on objects : constants

– constant objects;
for example:

petal = 

Operations on objects : rotation

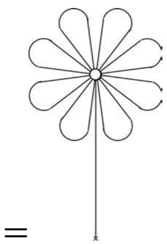
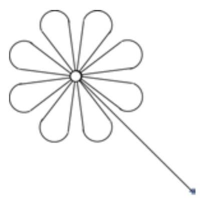
– rotation r[a](o) of objects o (of some angle a around the origin):



Example 1 of rotation

petal =  r[45](petal) = 

Example 2 of rotation

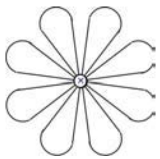
flower =  r[-45](flower) = 

Operations on objects : union

– union $o_1 \cup o_2$ of objects o_1 and o_2 = superposition at the origin;

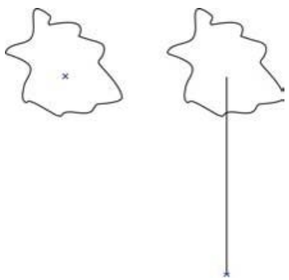
for example:

$$\text{corolla} = \text{petal} \cup \text{r}[45](\text{petal}) \cup \text{r}[90](\text{petal}) \cup \text{r}[135](\text{petal}) \cup \text{r}[180](\text{petal}) \cup \text{r}[225](\text{petal}) \cup \text{r}[270](\text{petal}) \cup \text{r}[315](\text{petal})$$



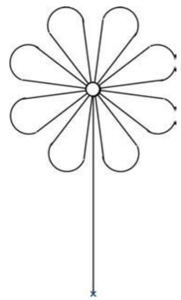
Operations on objects : add a stem

– stem(o) adds a stem to an object o (up to the origin, with new origin at the root);



Flower

$$\text{flower} = \text{stem}(\text{corolla})$$



Fixpoints

– $\text{corolla} = \text{lfp}^{\subseteq}\, F$

$F(X) = \text{petal} \cup \text{r}[45](X)$

Contraints

– A corolla is the $\subseteq$-least object X satisfying the two constraints:

- A corolla contains a petal:

$$\text{petal} \subseteq X$$

- and, a corolla contains its own rotation by 45 degres:

$$\text{r}[45](X) \subseteq X$$

– Or, equivalently[31]:

$$F(X) \subseteq X, \qquad \text{where} \qquad F(X) = \text{petal} \cup \text{r}[45](X)$$

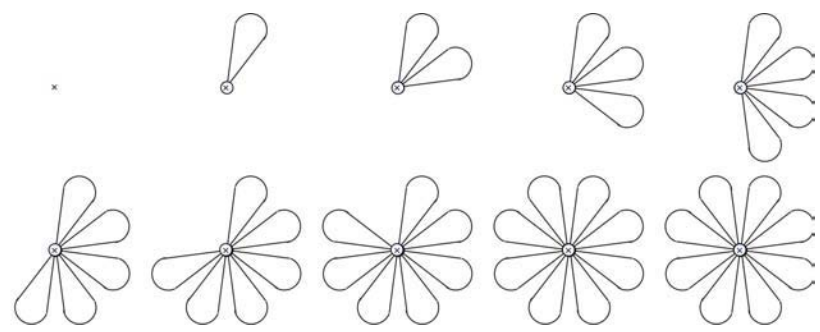
[31] By Tarski's fixpoint theorem, the least solution is $\text{lfp}^{\subseteq} F$.

Iterates to fixpoints

– The iterates of F from the infimum $\varnothing$ are:

$$\begin{aligned}
X^0 &= \varnothing\ ,\\
X^1 &= F(X^0)\ ,\\
&\ldots\ldots\ldots,\\
X^{n+1} &= F(X^n)\ ,\\
&\ldots\ldots\ldots\ ,\\
\text{lfp}^{\subseteq} F &= X^{\omega} = \bigcup_{n \geq 0} X^n\ .
\end{aligned}$$

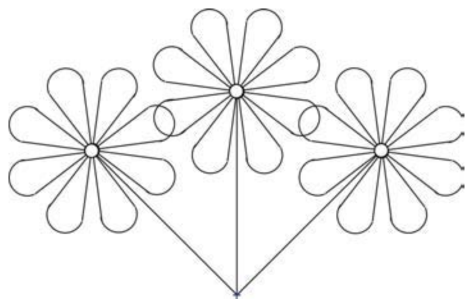
Iterates for the corolla



The bouquet

– bouquet = r[-45](flower) $\cup$ flower $\cup$ r[45](flower)

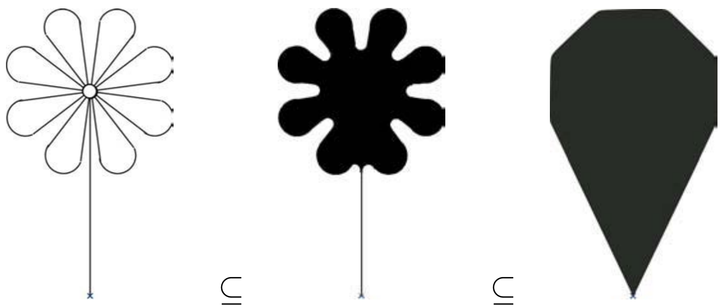
– The bouquet :



Upper-approximation

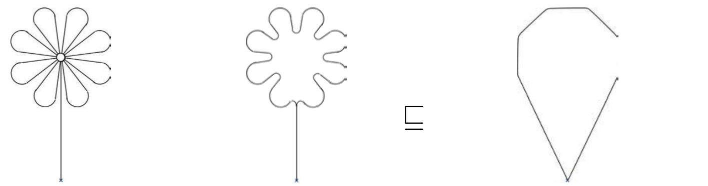
– An upper-approximation of an object is a object with:
  - same origin;
  - more pixels.

Examples of upper-approximations of flowers



Abstract objects

– an abstract object is a mathematical/computer representation of an approximation of a concrete object;



concrete object   abstract object   more abstract object

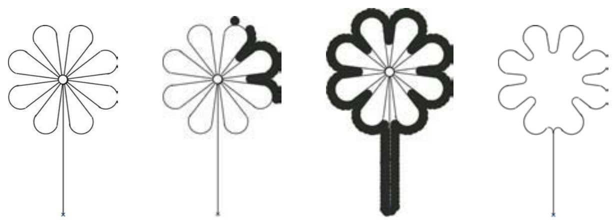
Abstract domain

– an abstract domain is a set of abstract objects plus abstract operations (approximating the concrete ones);

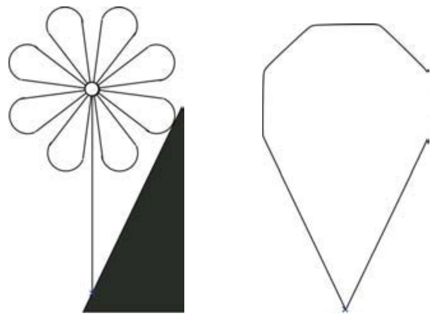
Abstraction

– an abstraction function α maps a concrete object o to an approximation represented by an abstract object $\alpha(o)$.

Example 1 of abstraction



Example 2 of abstraction



Comparing abstractions

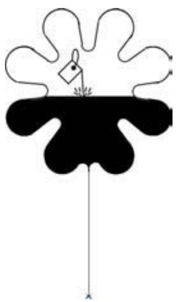
– larger pen diameters : more abstract;

– different pen shapes : may be non comparable abstractions.

Concretization

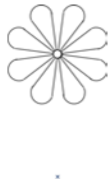
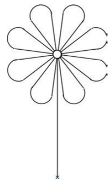
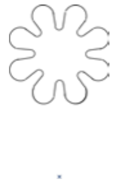
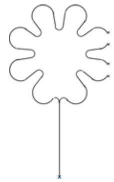
– a concretization function γ maps an abstract object $\overline{o}$ to the concrete object $\gamma(\overline{o})$ that is represents (that is to its concrete meaning/semantics).

Example of concretization



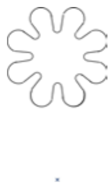
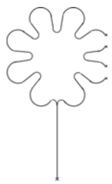
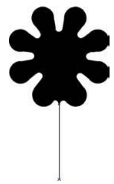
Galois connection 1/4

– α is monotonic.

 $\subseteq$  implies  $\sqsubseteq$ 

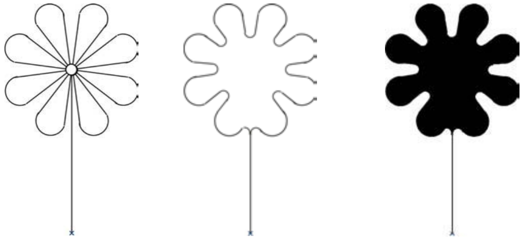
Galois connection 2/4

– γ is monotonic.

 $\sqsubseteq$  implies  $\subseteq$ 

Galois connection 3/4

– for all concrete objects x, $\gamma \circ \alpha(x) \supseteq x$ [32].

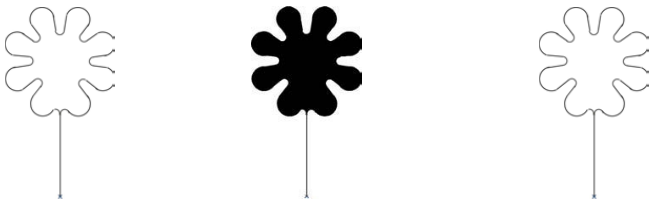


flower α(flower) $\gamma(\alpha$(flower))

[32] $f \circ g \triangleq \lambda x \cdot f(g(x))$

Galois connection 4/4

– for all abstract objects y, $\alpha \circ \gamma(y) \sqsubseteq y$.



abstract flower γ(abstract flower) $\alpha(\gamma$(abstract flower))

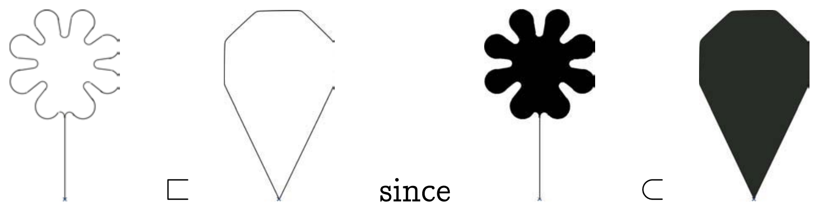
Galois connections

$$\langle \mathcal{D}, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{D}}, \sqsubseteq \rangle$$

$$\begin{array}{ll}
\text{iff} & \forall x, y \in \mathcal{D} : x \subseteq y \Longrightarrow \alpha(x) \sqsubseteq \alpha(y) \\
& \wedge\ \forall \overline{x}, \overline{y} \in \overline{\mathcal{D}} : \overline{x} \sqsubseteq \overline{y} \Longrightarrow \gamma(\overline{x}) \subseteq \gamma(\overline{y}) \\
& \wedge\ \forall x \in \mathcal{D} : x \subseteq \gamma(\alpha(x)) \\
& \wedge\ \forall \overline{y} \in \overline{\mathcal{D}} : \alpha(\gamma(\overline{y})) \sqsubseteq \overline{x} \\
\\
\text{iff} & \forall x \in \mathcal{D}, \overline{y} \in \overline{\mathcal{D}} : \alpha(x) \sqsubseteq y \Longleftrightarrow x \subseteq \gamma(y)
\end{array}$$

Abstract ordering

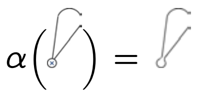
– $x \sqsubseteq y$ is defined as $\gamma(x) \subseteq \gamma(y)$.



Specification of abstract operations

- $\overline{\text{op}/0} \triangleq \alpha(\text{op}/0)$ 0-ary
- $\overline{\text{op}/1}(y) \triangleq \alpha(\text{op}/1(\gamma(y)))$ unary
- $\overline{\text{op}/2}(y, z) \triangleq \alpha(\text{op}/2(\gamma(y), \gamma(z)))$ binary
- ...

Abstract petal

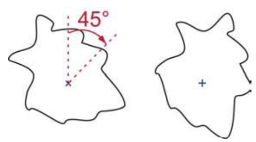


Abstract rotations

- $\bar{\text{r}}[a](y) \triangleq \alpha(\text{r}[a](\gamma(y)))$

Abstract rotations

- $\bar{\text{r}}[a](y) \triangleq \alpha(\text{r}[a](\gamma(y)))$
  $= \text{r}[a](y)$



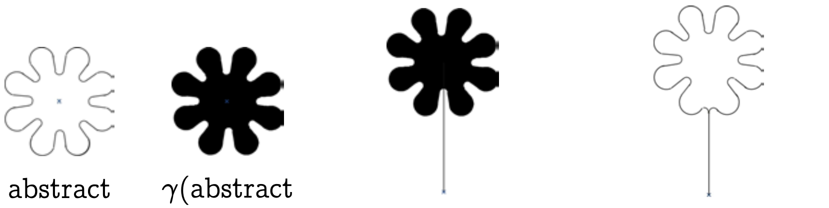
A commutation theorem on abstract rotations

- $\alpha(\mathrm{r}[a](x))$
  $= \alpha(\gamma(\alpha(\mathrm{r}[a](x))))$ [33]
  $= \alpha(\gamma(\mathrm{r}[a](\alpha(x))))$ [34]
  $= \alpha(\mathrm{r}[a](\gamma(\alpha(x))))$ [35]
  $= \bar{\mathrm{r}}[a](\alpha(x))$ [36]

[33] In a Galois connection: $\alpha = \alpha \circ \gamma \circ \alpha$
[34] Rotation is the same before or after abstraction
[35] Rotation is the same before or after concretization
[36] Def. $\bar{\mathrm{r}}[a]$

Abstract stems

- $\overline{\mathrm{stem}}(y) \triangleq \alpha(\mathrm{stem}(\gamma(y)))$



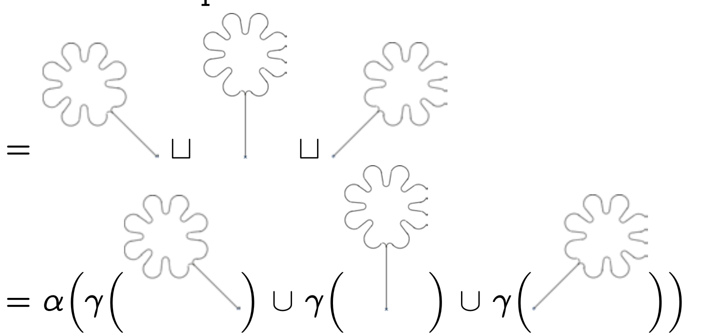
abstract corolla | γ(abstract corolla) | stem(γ(abstract corolla)) | α(stem(γ(abstract corolla)))

Abstract union

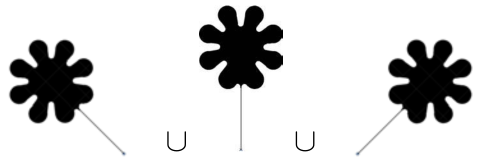
- $x \sqcup y \triangleq \alpha(\gamma(x) \cup \gamma(y))$

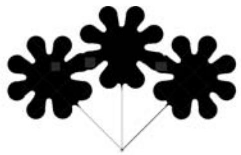
Abstract bouquet:

abstract bouquet

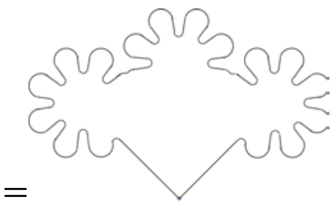


Abstract bouquet: (cont'd)

$= \alpha\Big($  $\Big)$

$= \alpha($  $)$

Abstract bouquet: (end)

$=$ 

A theorem on the abstract bouquet

abstract flower $= \alpha$(concrete flower)

abstract bouquet

$=$ $\bar{\mathrm{r}}$[-45](abstract flower) $\sqcup$ abstract flower $\sqcup$ $\bar{\mathrm{r}}$[-45](abstract flower)

$=$ $\bar{\mathrm{r}}$[-45](α(concrete flower)) $\sqcup$ α(concrete flower) $\sqcup$ $\bar{\mathrm{r}}$[-45](α(concrete flower))

$=$ α(r[-45](concrete flower)) $\sqcup$ α(concrete flower) $\sqcup$ α(r[-45](concrete flower))

$=$ α(r[-45](concrete flower) $\cup$ concrete flower $\cup$ r[-45](concrete flower))

$=$ α(concrete bouquet)

Abstract fixpoint

– abstract corolla $= \alpha$(concrete corolla) $= \alpha(\mathrm{lfp}^{\subseteq}\ F)$
  where $F(X) =$ petal $\cup$ r[45]$(X))$

Abstract transformer $\overline{F}$

– $\alpha(F(X))$
$= \alpha(\text{petal} \cup \text{r}[45](X))$
$= \alpha(\text{petal}) \sqcup \alpha(\text{r}[45](X))$
$= \alpha(\text{petal}) \sqcup \overline{\text{r}}[45](\alpha(X))$
$= \text{abstract petal} \sqcup \overline{\text{r}}[45](\alpha(X))$
$= \overline{F}(\alpha(X))$

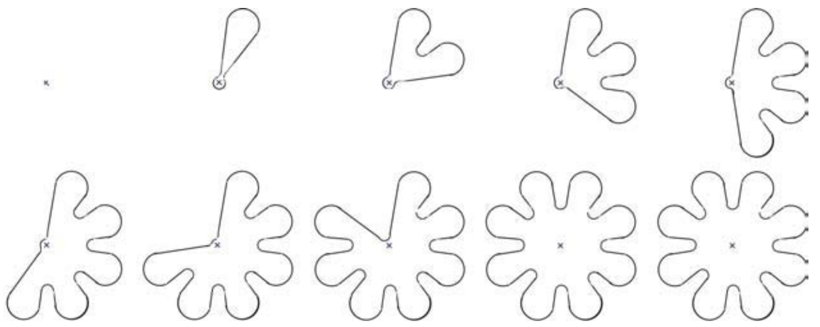
by defining

$$\overline{F}(X) = \text{abstract petal} \sqcup \overline{\text{r}}[45](X)$$

and so:

– abstract corolla $= \alpha(\text{concrete corolla}) = \alpha(\text{lfp}^{\subseteq} F) = \text{lfp}^{\sqsubseteq} \overline{F}$

Iterates for the abstract corolla



Abstract interpretation of the (graphic) language

– Similar, but by syntactic induction on the structure of programs of the language;

On abstracting properties of graphic objects

– A graphic object is a set of (black) pixels (ignoring the origin for simplicity);
– So a property of graphic objects is a set of graphic objects that is a set of sets of (black) pixels (always ignoring the set of origins for simplicity);
– Was there something wrong?

On abstracting properties of graphic objects

– No, because we implicitly used the following implicit looseness abstraction:

$$\langle \wp(\wp(\mathcal{P})), \subseteq \rangle \xrightleftharpoons[\alpha_0]{\gamma_0} \langle \wp(\mathcal{P}), \subseteq \rangle$$

where:

$\mathcal{P}$ is a set of pixels (e.g. pairs of coordinates)

$\alpha_0(X) = \bigcup X$

$\gamma_0(Y) = \{G \in \mathcal{P} \mid G \subseteq Y\}$

Is it for fun (only)?

– Yes, but see image processing by *morphological filtering*:

J. Serra. Morphological filtering: An overview, Signal Processing 38 (1994) 3–11.

It can be entirely formalized by abstract interpretation.

4. Elements of Abstract Interpretation

Semantics

Semantics

– The semantics $\mathcal{S}[\![\mathbb{p}]\!]$ of a software and hardware system $\mathbb{p} \in \mathbb{P}$ is a formal model of the execution of this system $\mathbb{p}$.

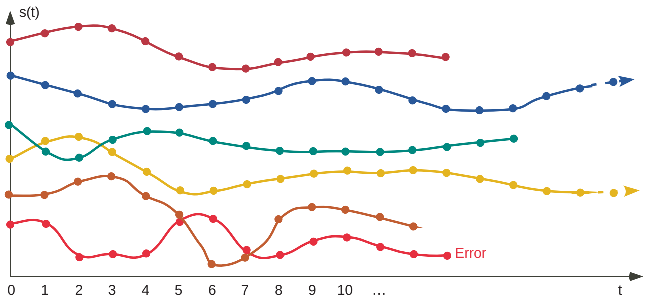
– A semantic domain $\mathcal{D}$ is a set of such formal models, so

$$\forall \mathbb{p} \in \mathbb{P} : \mathcal{S}[\![\mathbb{p}]\!] \in \mathcal{D}$$ [37]

[37] To be more precise one might consider $\mathcal{D}[\![\mathbb{p}]\!], \mathbb{p} \in \mathbb{P}$.

Example: Operational Semantics

– The operational semantics describes all possible program executions as a set of *maximal execution traces*



States and Traces

– States in Σ, describe an instantaneous snapshot of the execution

– Traces are finite or infinite sequences of states in Σ, two successive states corresponding to an elementary program step.

– In that case

- $\Sigma^n \triangleq [0, n[\mapsto \Sigma$ traces of length $n = 1, \ldots, +\infty$ [38].
- $\mathcal{T} \triangleq \bigcup_{n=1}^{+\infty} \Sigma^n$ all possible traces
- $\mathcal{D} \triangleq \wp(\mathcal{T})$ [39] semantic domain

[38] $[0, n[= \{0, 1, \ldots, n-1\}$ with $[0, 0[= \emptyset$.

[39] $\wp(S) \triangleq \{S' \mid S' \subseteq S\}$ is the powerset of S.

Properties and Specifications

Properties and Specifications

- A specification is a required *property* of the semantics of the system.
- The interpretation of a property is therefore a set of semantic models that satisfy this property
- Formally, the set of properties is

$$\mathcal{P} \triangleq \wp(\mathcal{D}) \ .$$

Example: Properties of a Trace Semantics

– $\mathcal{T}$	all possible traces
– $\mathcal{D} \triangleq \wp(\mathcal{T})$	semantic domain (sets of traces)
– $\mathcal{P} \triangleq \wp(\mathcal{D}) \triangleq \wp(\wp(\mathcal{T}))$	properties (sets of sets of traces)

The Complete Lattice of Semantic Properties

The semantic properties have a complete lattice (indeed Boolean lattice) structure:

$$\langle \wp(\mathcal{D}),\ \subseteq,\ \emptyset,\ \mathcal{D},\ \cup,\ \cap,\ \neg \rangle$$

The implication/set inclusion $\subseteq$ is a partial order:

- *reflexive*: $\forall X \in \wp(\mathcal{D}) : X \subseteq X$
- *antisymmetric*:
  $$\forall X, Y \in \wp(\mathcal{D}) : X \subseteq Y \wedge Y \subseteq X \Longrightarrow X = Y$$
- *transitive*:
  $$\forall X, Y, Z \in \wp(\mathcal{D}) : X \subseteq Y \wedge Y \subseteq Z \Longrightarrow X \subseteq Z$$

The join/union $\cup$ is the least upper bound (lub):

- $\cup$ is an *upper bound*: $\forall \langle X_i \in \wp(\mathcal{D}),\ i \in \Delta \rangle : \forall j \in \Delta :$
  $$X_j \subseteq \bigcup_{i \in \Delta} X_i$$
- $\cup$ is the *least one*: $\forall \langle X_i \in \wp(\mathcal{D}),\ i \in \Delta \rangle : \forall Y \in \wp(\mathcal{D}) :$
  $$(\forall j \in \Delta : X_j \subseteq Y) \Longrightarrow (\bigcup_{i \in \Delta} X_i \subseteq Y)$$

The meet/union $\cap$ is the greatest lower bound (glb):

– $\cap$ is a *lower bound*: $\forall \langle X_i \in \wp(\mathcal{D}),\ i \in \Delta\rangle : \forall j \in \Delta :$

$$\bigcap_{i\in\Delta} X_i \subseteq X_j$$

– $\cap$ is the *greatest one*: $\forall \langle X_i \in \wp(\mathcal{D}),\ i \in \Delta\rangle : \forall Y \in \wp(\mathcal{D}) :$

$$(\forall j \in \Delta : Y \subseteq X_j) \Longrightarrow (Y \subseteq \bigcap_{i\in\Delta} X_i)$$

The infimum/empty set is $\emptyset$ such that

– $\forall X \in \wp(\mathcal{D}) : \emptyset \subseteq X$

– $\emptyset = \bigcap \wp(\mathcal{D}) = \bigcup \emptyset$

The supremum is $\mathcal{D}$ such that

– $\forall X \in \wp(\mathcal{D}) : X \subseteq \mathcal{D}$

– $\mathcal{D} = \bigcup \wp(\mathcal{D}) = \bigcap \emptyset$

The complement $\neg X \triangleq \mathcal{D} \setminus X$ satisfies

– $X \cap \neg X = \emptyset$

– $X \cup \neg X = \mathcal{D}$

and is unique

Lattices

Lattice Theory

– Lattice theory was introduced by Garrett Birkhoff [13]

– Weakens set theory while keeping essential results

Reference

[13] G. Birkhoff. Lattice Theory. AMS Colloquium publications Vol. 25, 3rd Ed., 1973.

Partial Order

$$\langle L,\ \sqsubseteq\rangle$$

- L is a set
- The relation $\sqsubseteq$ on L is a reflexive, antisymmetric and transitive

The lub/glb might not exist for finite subsets of L.

Lattices

$$\langle L,\ \sqsubseteq,\ \sqcup,\ \sqcap\rangle$$

- $\langle L,\ \sqsubseteq\rangle$ is a partial order
- The lub $x \sqcup y$ exists for all $x, y \in L$ (whence for any finite subset of L)
- The glb $x \sqcap y$ exists for all $x, y \in L$ (whence for any finite subset of L)

The lub/glb might not exist for infinite subsets of L.

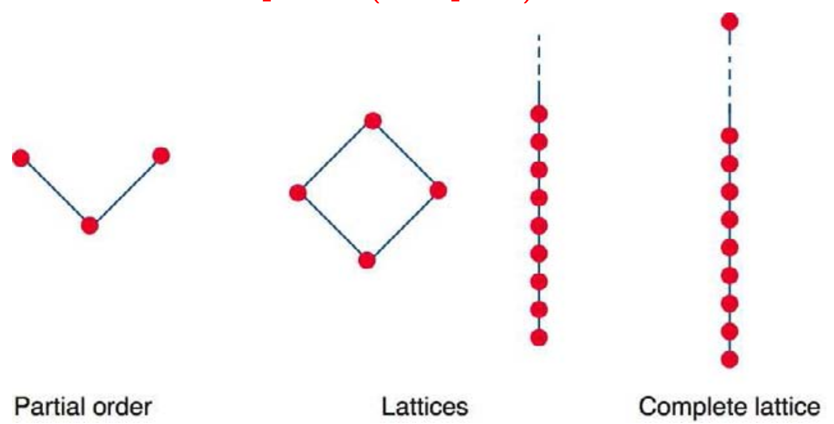
Complete Lattices

$$\langle L,\ \sqsubseteq,\ \bot,\ \top,\ \sqcup,\ \sqcap\rangle$$

- $\langle L,\ \sqsubseteq\rangle$ is a partial order
- The lub $\bigsqcup X$ does exist for all subsets X of L
- It follows that the glb $\bigsqcap X \triangleq \bigsqcup\{y \mid \forall x \in X : y \sqsubseteq x\}$ does exist for all subsets X of L
- It follows that L has in infimum $\bot = \sqcap L = \sqcup\varnothing$ and a supremum $\top = \sqcup L = \sqcap\varnothing$

The complement may not exist for all elements of L and may not be unique. Any finite lattice is complete.

Examples of (Complete) Lattices



Duality Principle

– The dual of $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ is $\langle L, \sqsupseteq, \top, \bot, \sqcap, \sqcup\rangle$
– If a statement is true in lattice theory, it's dual is also true
– Hence, there is no need for a dual of abstract interpretation theory [40]!

[40] Despite numerous counter-examples, see e.g. E.M. Clarke, O. Grumberg, and D.E. Long, Model Checking and Abstraction, TOPLAS 16:5(1512–1542), 1994.

Collecting Semantics

– The strongest property of a system $\mathbb{p} \in \mathbb{P}$ is its semantics $\{\mathcal{S}[\![\mathbb{p}]\!]\}$, called the collecting semantics

$$\mathcal{C}[\![\mathbb{p}]\!] \triangleq \{\mathcal{S}[\![\mathbb{p}]\!]\} .$$

Verification

Verification

– The satisfaction of a specification $P \in \mathcal{P}$ by a system $\mathbb{p}$ (more precisely by the system semantics $\mathcal{S}[\![\mathbb{p}]\!]$) is

$$\mathcal{S}[\![\mathbb{p}]\!] \in P$$

– Satisfaction can equivalently be defined as the proof that

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

i.e. *the strongest program property implies its specification.*

Undecidability

– The proof that

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

is not mechanizable (Gödel, Turing).

Abstraction

Abstraction

To prove

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq P$$

one can use a sound over-approximation of the collecting semantics

$$\mathcal{C}[\![\mathbb{p}]\!] \subseteq \overline{\mathcal{C}}[\![\mathbb{p}]\!]$$

and a sound under-approximation of the property

$$\overline{P} \subseteq P$$

and make the correctness proof *in the abstract*

$$\overline{\mathcal{C}}[\![\mathbb{p}]\!] \subseteq \overline{P}$$

Abstract Domain

– For automated proofs, $\overline{\mathcal{C}}[\![\mathbb{p}]\!]$ and $\overline{P}$ must be computer-representable

– Hence, they are not chosen in the mathematical concrete domain

$$\langle \mathcal{P}, \subseteq \rangle$$

but in a computer-representable abstract domain

$$\langle \overline{\mathcal{P}}, \sqsubseteq \rangle$$

Concretization Function

– The abstract to concrete correspondence is given by a concretization function

$$\gamma \in \overline{\mathcal{P}} \mapsto \mathcal{P}$$

providing the meaning $\gamma(\overline{P})$ of abstract properties $\overline{P}$

– For abstract reasonings to be valid in the concrete, γ should preserve the abstract implication

$$\forall Q_1, Q_2 \in \overline{\mathcal{P}} : (Q_1 \sqsubseteq Q_2) \Longrightarrow (\gamma(Q_1) \subseteq \gamma(Q_2))$$

Soundness of the Abstraction

– The soundness of the abstract over-approximation of the collecting semantics is now

$$\mathcal{C}[\![\mathtt{p}]\!] \subseteq \gamma(\overline{\mathcal{C}}[\![\mathtt{p}]\!])$$

– The soundness of the abstract under-approximation of the property is now

$$\gamma(\overline{P}) \subseteq P$$

Abstract Proofs

– Then, the abstract proof

$$\overline{\mathcal{C}}[\![\mathtt{p}]\!] \sqsubseteq \overline{P}$$

implies

$$\gamma(\overline{\mathcal{C}}[\![\mathtt{p}]\!]) \subseteq \gamma(\overline{P})$$

and by soundness of the abstraction

$$\mathcal{C}[\![\mathtt{p}]\!] \subseteq \gamma(\overline{\mathcal{C}}[\![\mathtt{p}]\!]) \quad \text{and} \quad \gamma(\overline{P}) \subseteq P$$

we have *proved correctness in the concrete*

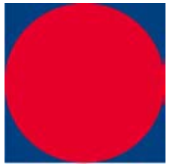
$$\mathcal{C}[\![\mathtt{p}]\!] \subseteq P \ .$$

Galois Connections

Best Abstraction

– If we want to over-approximate a disk in two dimensions by a polyhedron there is no best (smallest) one, as shown by Euclid.

– However if we want to over-approximate a disk by a rectangular parallelepiped which sides are parallel to the axes, then there is definitely a best (smallest) one.



Best Abstraction (Cont'd)

– In case of best over-approximation, there is an abstraction function

$$\alpha \in \mathcal{P} \mapsto \overline{\mathcal{P}}$$

such that

- for all $P \in \mathcal{P}$, $\alpha(P) \in \overline{\mathcal{P}}$ is an abstract over-approximation of P, so

$$P \subseteq \gamma(\alpha(P))$$

and,

- it is the most precise abstract over-approximation, so

$$\forall Q \in \overline{\mathcal{P}} : P \subseteq \gamma(Q) \Longrightarrow \alpha(P) \sqsubseteq Q$$

(whence $\gamma(\alpha(P)) \subseteq \gamma(Q)$ by monotony of γ).

Best Abstraction and Galois Connection

– It follows in that case of existence of a best abstraction, that the pair $\langle \alpha, \gamma \rangle$ is a Galois connection [14].

$$\forall P \in \mathcal{P} : \forall Q \in \overline{\mathcal{P}} : P \subseteq \gamma(Q) \Longleftrightarrow \alpha(P) \sqsubseteq Q$$

written

$$\langle \mathcal{P}, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{P}}, \sqsubseteq \rangle$$

Reference

[14] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. 6^{th} *ACM POPL*, 269–282, 1979.

Galois Connection Preserve Existing Joins

If

$$\text{poset} \to \langle L, \sqsubseteq\rangle \xleftarrow[\alpha]{\gamma} \langle \overline{L}, \overline{\sqsubseteq}\rangle \leftarrow \text{poset} \qquad (1)$$

and $\bigsqcup_i X_i$ exists in $\langle L, \sqsubseteq\rangle$ then

$$\alpha(\bigsqcup_i X_i) = \overline{\bigsqcup_i} \alpha(X_i)$$

Reciprocally, if α preserves existing joins then it has a unique adjoint γ satisfying Eq. (1)

Examples of Abstractions I

I.1 — Traditional View of Program Properties

– In the operational trace semantics example $\mathcal{D} \triangleq \wp(\mathcal{T})$ so properties are

$$\mathcal{P} \triangleq \wp(\wp(\mathcal{T}))$$

where $\mathcal{T}$ is the set of traces.

– The traditional view of program properties as set of traces [15], [16] is an abstraction.

References

[15] B. Alpern and F. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, 1985.

[16] A. Pnueli. The temporal logic of programs. *18th ACM FOCS*, 46–57, 1977.

I.1 — Example of Program Properties

– An example of progam property is

$$P_{01} \triangleq \{\{\sigma 0 \mid \sigma \in \mathcal{T}\}, \{\sigma 1 \mid \sigma \in \mathcal{T}\}\} \in \mathcal{P}$$

specifying that executions of the system always terminate with 0 or always terminate with 1.

– This cannot be expressed in the traditional view of program properties as set of traces [15], [16].

I.1 — Looseness Abstraction

– This traditional understanding of a program property is given by the looseness abstraction

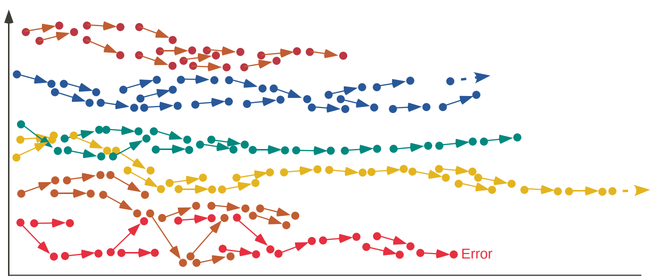
$$\alpha_\cup \in \wp(\wp(\mathcal{T})) \mapsto \wp(\mathcal{T}),$$
$$\alpha_\cup(P) \triangleq \bigcup P$$

with concretization

$$\gamma_\cup \in \wp(\mathcal{T}) \mapsto \wp(\wp(\mathcal{T})),$$
$$\gamma_\cup(Q) \triangleq \wp(Q) \ .$$

– An example is $\alpha_\cup(P_{01}) = \{\sigma 0, \sigma 1 \mid \sigma \in \mathcal{T}\}$ specifying that execution always terminate, either with 0 or with 1.

I.2 — Transition Abstraction



I.2 — Transition Abstraction

– The transition abstraction

$$\alpha_\tau \in \wp(\mathcal{T}) \mapsto \wp(\Sigma \times \Sigma)$$

collects transitions along traces.

$$\alpha_\tau(\sigma_0 \ldots \sigma_n) \triangleq \{\sigma_i \to \sigma_{i+1} \mid 0 \leqslant i < n\},$$
$$\alpha_\tau(\sigma_0 \ldots \sigma_i \ldots) \triangleq \{\sigma_i \to \sigma_{i+1} \mid i \geqslant 0\}, \quad \text{and}$$
$$\alpha_\tau(T) \triangleq \bigcup\{\alpha(\sigma) \mid \sigma \in T\} \ .$$

– The concretization $\gamma_\tau \in \wp(\Sigma \times \Sigma) \mapsto \wp(\mathcal{T})$ is

$$\gamma_\tau(\tau) \triangleq \bigcup_{n=1}^{+\infty} \{\sigma \in [0, n[\mapsto \Sigma \mid \forall i < n : \langle \sigma_i,\, \sigma_{i+1}\rangle \in \tau\} \ .$$

I.2 — Transition System Abstraction

– The abstraction may also collect initial states

$$\alpha_\iota(T) \triangleq \{\sigma_0 \mid \sigma \in T\}$$
$$\text{so} \quad \alpha_{\iota\tau}(T) \triangleq \langle \alpha_\iota(T),\, \alpha_\tau(T)\rangle \ .$$

– We let

$$\gamma_{\iota\tau} \triangleq \gamma_\iota(\iota) \cap \gamma_\tau(\tau)$$
$$\text{where} \quad \gamma_\iota(\iota) \triangleq \{\sigma \in \mathcal{T} \mid \sigma_0 \in \iota\}$$

– $\langle \alpha_{\iota\tau},\, \gamma_{\iota\tau}\rangle$ is a Galois connection.

I.2 — Transition System Abstraction (Cont'd)

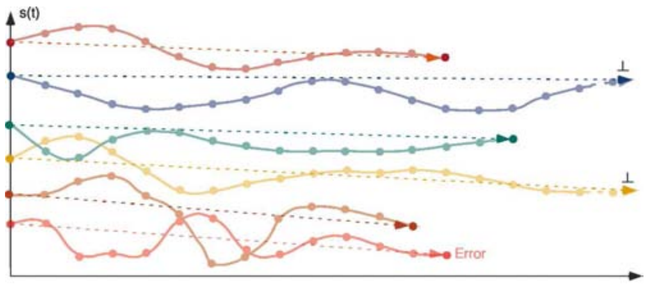
– The transition system abstraction [17] underlies small-step operational semantics.

– This is an approximation since traces can express properties not expressible by a transition system (like fairness of parallel processes).

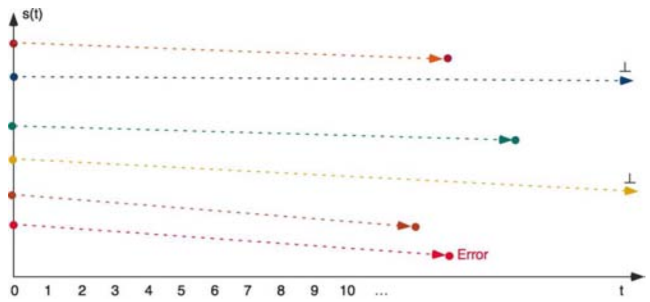
References

[17] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'État ès sci. math., Univ. sci. et médicale de Grenoble, 1978.

I.3 — Input-Output Abstraction



I.3 — Input-Output Abstract Semantics



I.3 — Input-Output Abstraction

– The input-output abstraction

$$\alpha_{io} \in \wp(\mathcal{T}) \mapsto \wp(\Sigma \times (\Sigma \cup \{\bot\}))$$

collects initial and final states of traces (and maybe $\bot$ for infinite traces to track nontermination).

$$\alpha_{io}(\sigma_0 \ldots \sigma_n) = \langle \sigma_0,\ \sigma_n \rangle,$$
$$\alpha_{io}(\sigma_0 \ldots \sigma_i \ldots) = \langle \sigma_0,\ \bot \rangle,$$

and

$$\alpha_{io}(T) = \{\alpha_{io}(\sigma) \mid \sigma \in T\}\ .$$

I.3 — Input-Output Abstraction (Cont'd)

– The input-output abstraction α_{io} underlies
  - denotational semantics, as well as big-step operational, predicate transformer and axiomatic semantics extended to nontermination [21], and
  - interprocedural static analysis using relational procedure summaries [18], [19], [20].

References

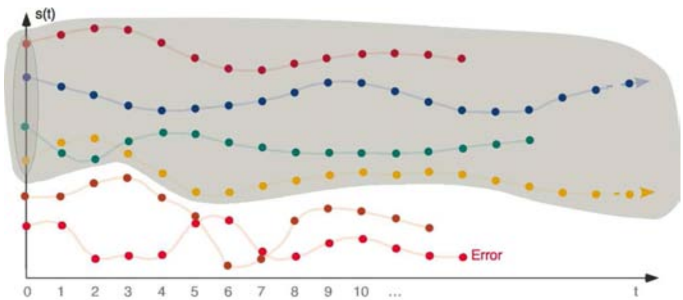
[18] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *IFIP Conf. on Formal Description of Programming Concepts*, 237–277, North-Holland, 1977.

[19] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

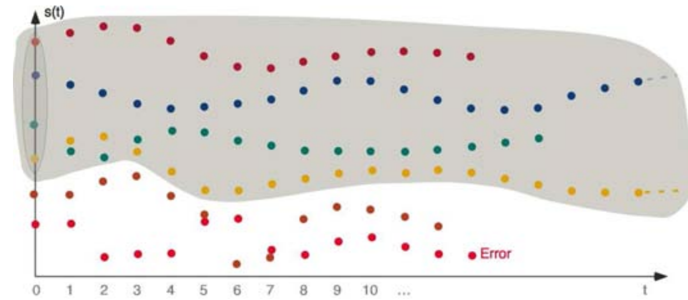
[20] P. Cousot and R. Cousot. Modular static program analysis. 11^{th} CC, LNCS 2304, 159–178, Springer, 2002.

[21] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

I.4 — Reachability Abstraction



I.4 — Reachability Semantics (System Invariant)



I.4 — Reachability Abstraction

– The reachability abstraction collects states along traces.

$$\begin{aligned}
\alpha_r &\in \wp(\mathcal{T}) \mapsto \wp(\Sigma)\\
\alpha_r(T) &\triangleq \{\sigma_i \mid \exists n \in [0, +\infty] : \sigma \in \Sigma^n \cap T \wedge\\
&\qquad\qquad i \in [0, n[\}\\
&\subseteq^{41} \{s' \in \Sigma \mid \exists s \in \iota : \langle s,\, s'\rangle \in \tau^\star\}
\end{aligned}$$

where $\alpha_{\iota\tau}(T) = \langle \iota,\, \tau\rangle$ is the transition abstraction

and $\tau^\star$ is the reflexive transitive closure of τ.

[41] We may have $\subsetneq$ when $T \neq \gamma_{\iota\tau}(\alpha_{\iota\tau}(T))$. We assume $T = \gamma_{\iota\tau}(\alpha_{\iota\tau}(T))$ in the rest of the talk.

I.4 — Invariants

– Expressed in logical form, the reachability abstraction α provides a system invariant

$$\alpha(\mathcal{C}[\![\mathbb{p}]\!])$$

that is the set of all states that can be reached along some execution of the system $\mathbb{p}$ [22], [23].

References

[22] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

[23] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM POPL*, 238–252, 1977.

I.4 — Floyd's Proof Method

Floyd's method [24] to prove a reachability property

$$\alpha_\tau(T) \subseteq P$$

consists in finding an invariant I stronger than P, i.e.

$$I \subseteq P$$

which is inductive, i.e.

$$\iota \subseteq I$$

and $$\tau[I] \subseteq I$$

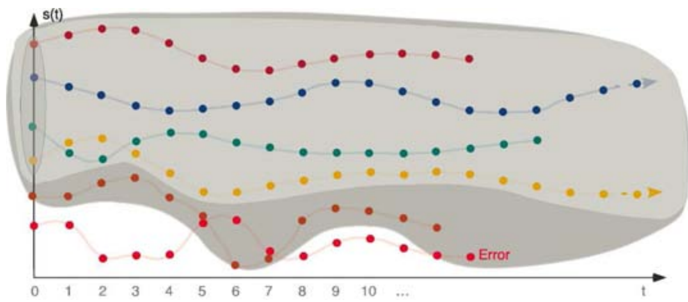
where $$\tau[I] \triangleq \{s' \mid \exists s \in I : \langle s,\ s'\rangle \in \tau\}$$

is the right-image transformer for the transition system $\langle \iota,\ \tau\rangle = \alpha_{\iota\tau}(T)$.

References

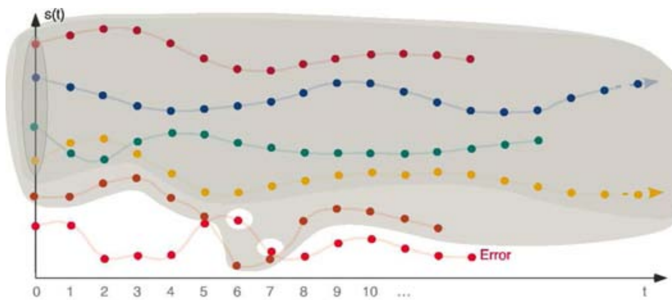
[24] R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, vol. 19, 19–32. AMS, 1967.

Example of invariant (I)



Not inductive (and too weak)!

Example of invariant (II)



Inductive and precise enough!

I.4 — Floyd's Proof Method (Cont'd)

- This induction principle has many equivalent variants [25], all underlying different static analysis methods (the equivalence may not be preserved by abstraction).
- In particular backward analyzes are based on

$$\langle \tau^{-1},\ \alpha_\varphi(T)\rangle$$

where τ^{-1} is the inverse of τ

and $\alpha_\varphi(T) \triangleq \{\sigma_{n-1} \mid n < +\infty \wedge \sigma \in T \cap \Sigma^n\}$

collects final states.

References

[25] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs. *Tools & Notions for Program Construction*, 43–119. Cambridge U. Press, 1982.

In Absence of Best Abstraction

Example of Absence of Best Abstraction

- $\mathbb{Z}$: set of integers
- $\wp(\mathbb{Z})$: integer properties [42]
- $\{\bot, \dot{+}, \dot{-}, \top\}$: abstract signs, with

$$\gamma(\bot) = \varnothing \qquad \gamma(\top) = \mathbb{Z}$$

$$\gamma(\dot{+}) = \{n \in \mathbb{Z} \mid n \geqslant 0\} \qquad \gamma(\dot{-}) = \{n \in \mathbb{Z} \mid n \leqslant 0\}$$

- 0 has no best abstraction (can be either $\dot{+}$ or $\dot{-}$)

[42] e.g. posssible values of an integer variable at runtime

In Absence of Best Abstraction (Cont'd)

- Among the possible choices, one may be locally preferable, e.g.
  - $0 + \dot{+}$, 0 should be abstracted to $\dot{+}$ (since $\dot{+} + \dot{+} = \dot{+}$ while $\dot{-} + \dot{+} = \top$)
  - $0 + \dot{-}$, 0 should be abstracted to $\dot{-}$ (since $\dot{-} + \dot{-} = \dot{-}$ while $\dot{+} + \dot{-} = \top$)

What to do in Absence of Best Abstraction

1. Close the abstract domain by intersection (Moore family)

e.g. $\{\bot, \dot{+}, 0, \dot{-}, \top\}$: abstract signs (with $\gamma(0) = \{0\}$ so $0 + \dot{+} = \dot{+}$ and $0 + \dot{-} = \dot{-}$)

$\Rightarrow$ In general, there are infinitely many possible choices so the Moore closure is quite complex [43]

2. Try all possible choices and locally keep the best one [44]

$\Rightarrow$ Make arbitrary choices [45]

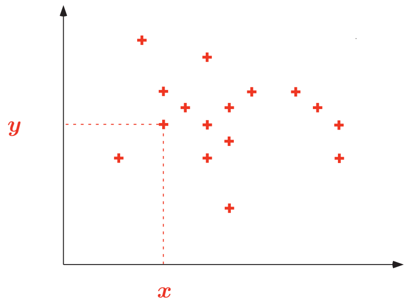
[43] e.g. polyedra can be closed in convex sets much harder to represent in machines

[44] In general impossible due to combinatorial explosion

[45] Using a concretization function γ this choice can be made locally, while with an α it is made globally, once for all, see P. Cousot & R. Cousot. *Abstract interpretation frameworks.* Journal of Logic and Computation, 2(4):511—547, Aug. 1992.

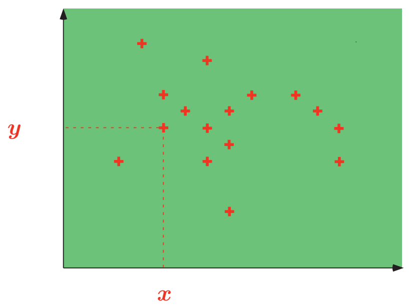
Examples of Abstractions II

Effective computable approximations of an [in]finite set of points;



$\langle x, y\rangle \in \{\langle 19, 77\rangle, \langle 20, 07\rangle, \ldots\}$

Effective computable approximations of an [in]finite set of points; Signs [46]



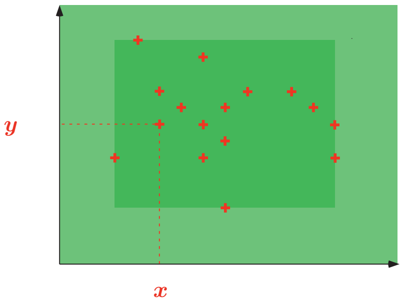
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Non-relational

Best abstraction (with 0).

[46] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

Effective computable approximations of an [in]finite set of points; Intervals [47]



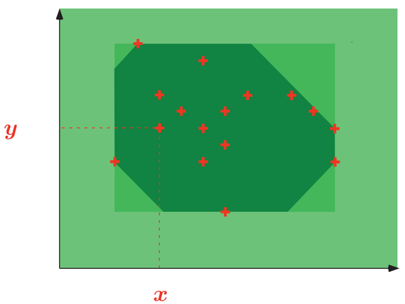
$$\begin{cases} x \in [19,\ 77] \\ y \in [20,\ 07] \end{cases}$$

Non-relational

Best abstraction.

[47] P. Cousot & R. Cousot. *Static determination of dynamic properties of programs.* Proc. 2nd Int. Symp. on Programming, Dunod, 1976.

Effective computable approximations of an [in]finite set of points; Octagons [48]



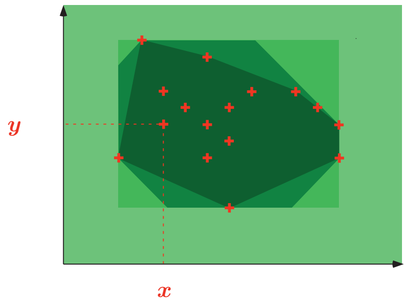
$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

Weakly relational

Best abstraction.

[48] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices.* PADO'2001. LNCS 2053, pp. 155–172. Springer 2001. See the *The Octagon Abstract Domain Library* on `http://www.di.ens.fr/~mine/oct/`

Effective computable approximations of an [in]finite set of points; Polyhedra [49]



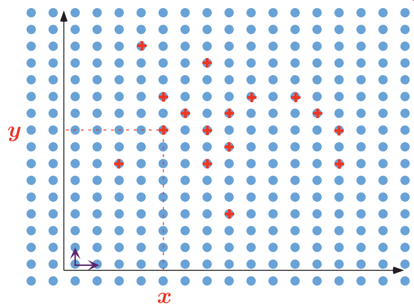
$$\begin{cases} 19x + 77y \leq 2004 \\ 20x + 03y \geq 0 \end{cases}$$

Relational

No best abstraction.

[49] P. Cousot & N. Halbwachs. *Automatic discovery of linear restraints among variables of a program.* ACM POPL, 1978, pp. 84–97.

Effective computable approximations of an [in]finite set of points; Simple congruences [50]



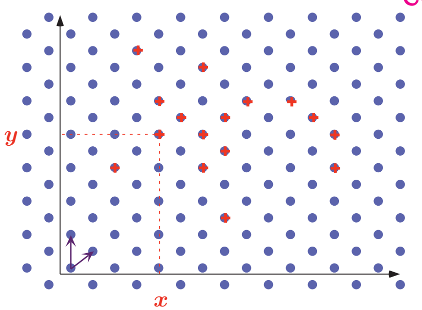
$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

Non-relational

Best abstraction.

[50] Ph. Granger. *Static Analysis of Arithmetical Congruences.* Int. J. Comput. Math. 30, 1989, pp. 165–190.

Effective computable approximations of an [in]finite set of points; Linear congruences [51]



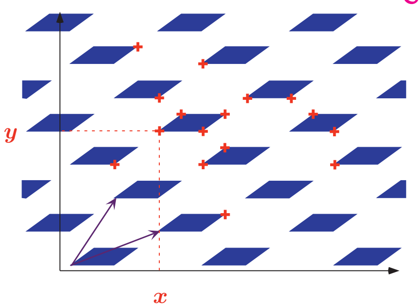
$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

Relational

Best abstraction.

[51] Ph. Granger. *Static Analysis of Linear Congruence Equalities among Variables of a Program.* TAPSOFT '91, pp. 169–192. LNCS 493, Springer, 1991.

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences [52]



$$\begin{cases} 1x + 9y \in [0, 77] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

Relational

No best abstraction.

[52] F. Masdupuy. *Array Operations Abstraction Using Semantic Analysis of Trapezoid Congruences.* ACM ICS '92.

Properties of Abstractions

Soundness of Abstractions

– An abstraction is sound [26] if the proof in the abstract implies the concrete property

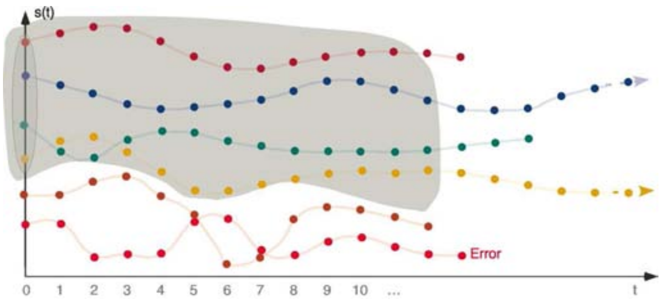
$$\overline{C}[\![\mathrm{p}]\!] \sqsubseteq \overline{P} \Longrightarrow C[\![\mathrm{p}]\!] \subseteq P\ .$$

– Abstract interpretation provides an effective theory to design sound abstractions.

References

[26] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. 6^{th} *ACM POPL*, 269–282, 1979.

Example of Unsound Abstraction (Bounded Model Checking)



Completeness of Abstractions

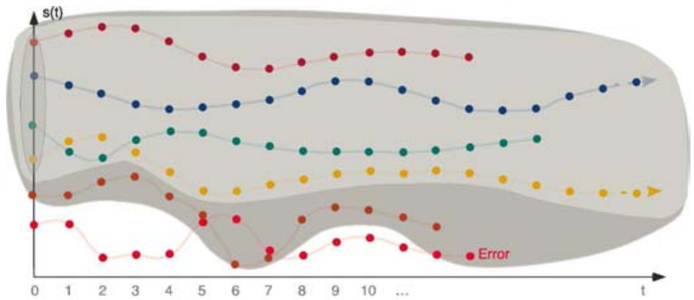
– An abstraction is complete [27] if the fact that the system is correct can always be proved in the abstract

$$\mathcal{C}[\![\mathrm{p}]\!] \subseteq P \Longrightarrow \overline{\mathcal{C}}[\![\mathrm{p}]\!] \sqsubseteq \overline{P}.$$

References

[27] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. 6^{th} *ACM POPL*, 269–282, 1979.

Example of Incomplete Abstraction (Static Analysis)



No error is reachable in the concrete but an error is reachable in the abstract ⇒ the proof fails in the abstract (false alarm)!

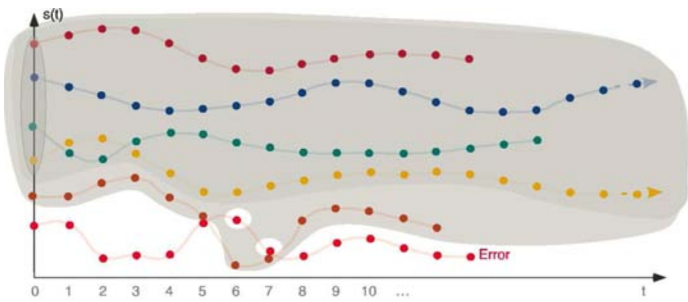
Refinement of Abstractions

– False alarms can always be avoided by refinement of the abstraction [28].

References

[28] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

Example of Refined Abstraction (Static Analysis)



No error is reachable in the abstract whence in the concrete ⇒ the proof succeeds in the abstract!

Incompleteness of the Refinement of Abstractions

– This refinement is not effective (i.e. the algorithm does not terminate in general).

– For example in model-checking any abstraction of a trace logic may be incomplete [29].

References

[29] R. Giacobazzi and F. Ranzato. Incompleteness of states w.r.t. traces in model checking. *Inform. and Comput.*, 204(3):376–407, Mar. 2006.

Adequation of Abstractions

– The reachability abstraction is sound and complete for invariance/safety proofs [53].

– That means that if $S \subseteq \Sigma$ is a set of safe states so that $\gamma_r(S)$ is a set of safe traces then the safety proof $\mathcal{C}[\![\mathtt{p}]\!] \subseteq \gamma_r(S)$ can always be done as $\alpha_r(\mathcal{C}[\![\mathtt{p}]\!]) \subseteq S$.

– This is the fundamental remark of Floyd [30] that it is not necessary to reason on traces to prove invariance properties.

References

[30] R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, vol. 19, 19–32. AMS, 1967.

53 Again, assuming $T = \gamma_{\iota r}(\alpha_{\iota r}(T))$

Adequation of Abstractions (Cont'd)

– This does not mean that this abstraction is adequate, that is, informally, the most simple way to do the proof.

– For example Burstall's intermittent assertions may be simpler than Floyd's invariant assertions [31]

– or, in static analysis trace partitioning may be more adequate that state-based reachability analysis [32].

References

[31] P. Cousot and R. Cousot. Sometime = always + recursion ≡ always: on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informat.*, 24:1–31, 1987.

[32] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. 14^{th} *ESOP*, LNCS 3444, 5–20. Springer, 2005.

Combinations of Abstractions

Reference

[POPL '79] P. Cousot & R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

Reduced Product of Abstract Domains

To combine abstractions

$$\langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha_1]{\gamma_1} \langle \mathcal{D}^\sharp_1, \sqsubseteq_1 \rangle \text{ and } \langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha_2]{\gamma_2} \langle \mathcal{D}^\sharp_2, \sqsubseteq_2 \rangle$$

the reduced product is

$$\alpha(X) \triangleq \sqcap\{\langle x, y \rangle \mid X \subseteq \gamma_1(x) \wedge X \subseteq \gamma_2(y)\}$$

such that $\sqsubseteq \triangleq \sqsubseteq_1 \times \sqsubseteq_2$ and

$$\langle \mathcal{D}, \subseteq \rangle \xleftrightarrow[\alpha]{\gamma_1 \times \gamma_2} \langle \alpha(\mathcal{D}), \sqsubseteq \rangle$$

Example: $\mathrm{x} \in [1, 9] \wedge \mathrm{x} \bmod 2 = 0$ reduces to $\mathrm{x} \in [2, 8] \wedge \mathrm{x} \bmod 2 = 0$

Reduction in ASTRÉE

– The computation of an abstract transformer $\overline{F}_1$ for an abstract domain $\overline{\mathcal{D}}_1$ can use an abstract invariant computed by another abstract domain $\overline{\mathcal{D}}_2$

– The two abstract domains communicate symbolically through a channel [54]

– A fixed communication order is used (so reduction cannot prevent to widening/narrowing convergence enforcement)

[33] P. Cousot and R. Cousot and J. Feret and L. Mauborgne and A. Miné and D. Monniaux and X. Rival. Combination of Abstractions in the ASTRÉE Static Analyzer. In 11^{th} *ASIAN 06*, Tokyo, Japan, 6–8 Dec. 2006, LNCS , Springer.

[54] using a common language to communicate whereas the representation of invariants may be quite different.

Transformers

Semantic Transformer

– The *concrete/semantic transformer* F describes the effect of program commands: if P describes behaviors before/after a command, then $F(P)$ describes behaviors after/before this command

$$F \in \mathcal{P} \overset{\text{mon}}{\longmapsto} \mathcal{P}$$

– Assumed to be monotonic: $\forall P, P' \in \mathcal{P} : (P \subseteq P') \Longrightarrow (F(P) \subseteq F(P'))$.

– Intuition: the more behaviors before/after a command, the more after/before the command

Abstract Transformer

– The *abstract transformer* $\overline{F}$ is

$$\overline{F} \in \overline{\mathcal{P}} \mapsto \overline{\mathcal{P}}$$

– Might not be monotonic (because of non-monotonic widening/narrowing, see later)

Sound Abstract Transformer

– The abstract transformer $\overline{F}$ overapproximates the concrete transformer F (for all abstract properties $\overline{P}$ considered in the concrete $\gamma(\overline{P})$):

$$\forall \overline{P} \in \overline{\mathcal{P}} : F(\gamma(\overline{P})) \subseteq \gamma(\overline{F}(\overline{P}))$$

– We speak of (exact) abstraction when

$$\forall \overline{P} \in \overline{\mathcal{P}} : F(\gamma(\overline{P})) = \gamma(\overline{F}(\overline{P}))$$

Best Abstract Transformer

Given $\langle \subseteq, \mathcal{P} \rangle$, $\langle \sqsubseteq, \overline{\mathcal{P}} \rangle$, $\gamma \in \overline{\mathcal{P}} \mapsto \mathcal{P}$ and $F \in \mathcal{P} \overset{\text{mon}}{\longmapsto} \mathcal{P}$, $\overline{F} \in \overline{\mathcal{P}} \mapsto \overline{\mathcal{P}}$ is the best approximation of F iff

(1) $\overline{F}$ is an *over approximation* of F:

$$\forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(\overline{P})) \Longrightarrow (F(P) \subseteq \gamma(\overline{F}(\overline{P}))$$

(2) $\overline{F}$ is the *most precise* over approximation of F: if

$$\forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(\overline{P})) \Longrightarrow (F(P) \subseteq \gamma(\overline{G}(\overline{P}))$$

then

$$\forall \overline{P} \in \overline{\mathcal{P}} : \overline{F}(\overline{P}) \sqsubseteq \overline{G}(\overline{P}) \; .$$

Given γ, the best abstract transformer might not exist!

Existence of a Best Abstract Transformer for Galois Connections

If

$$\langle \mathcal{P}, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{P}}, \sqsubseteq \rangle$$

then the best overapproximation of $F \in \mathcal{P} \xrightarrow{\text{mon}} \mathcal{P}$ is

$$\overline{F} \triangleq \alpha \circ F \circ \gamma \; .$$

PROOF

– $P \subseteq \gamma(\overline{P})$
  $\Longrightarrow F(P) \subseteq F(\gamma(\overline{P}))$ monotony of F
  $\Longrightarrow \alpha(F(P)) \sqsubseteq \alpha(F(\gamma(\overline{P})))$ monotony of α
  $\Longrightarrow F(P) \subseteq \gamma(\alpha \circ F \circ \gamma(\overline{P}))$ def. Galois connection
  proving $\alpha \circ F \circ \gamma$ to be an abstract overapproximation of F.
– If $\overline{G}$ is an abstract overapproximation of F:
  $$\forall P \in \mathcal{P}, \overline{P} \in \overline{\mathcal{P}} : (P \subseteq \gamma(\overline{P})) \Longrightarrow (F(P) \subseteq \gamma(\overline{G}(\overline{P}))$$
  then
  $\Longrightarrow F(\gamma(\overline{P})) \subseteq \gamma(\overline{G}(\overline{P})$ for $P = \gamma(\overline{P})$
  $\Longrightarrow \alpha \circ F \circ \gamma(\overline{P}) \sqsubseteq \overline{G}(\overline{P})$ def. Galois connection
  proving $\alpha \circ F \circ \gamma$ to be the *best* abstract overapproximation of F. ∎

Fixpoints

Fixpoint

– A fixpoint of F is X such that $X = F(X)$

– May not exist, may have [infinitely] many

– May have a least one $\mathsf{lfp}^{\sqsubseteq} F$ for a partial order $\sqsubseteq$:

- $F(\mathsf{lfp}^{\sqsubseteq} F) = \mathsf{lfp}^{\sqsubseteq} F$ *fixpoint*
- $X = F(X) \Longrightarrow \mathsf{lfp}^{\sqsubseteq} F \sqsubseteq X$ *least one*

Fixpoint Induction

$$(\mathsf{lfp}^{\sqsubseteq} F \sqsubseteq P) \Longleftrightarrow (\exists I : F(I) \sqsubseteq I \land I \sqsubseteq P)$$

Soundness $\Leftarrow$: $I \in \{x \in L \mid F(x) \sqsubseteq x\}$ so $\mathsf{lfp}^{\sqsubseteq} F = \bigsqcap\{x \in L \mid F(x) \sqsubseteq x\} \sqsubseteq I \sqsubseteq P$

Completeness $\Rightarrow$: choose $I = \mathsf{lfp}^{\sqsubseteq} F$

Examples:

– Floyd's *invariance proof* method

– *Static analysis*: any postfixpoint *overapproximates* the least fixpoint

Fixpoint Theorem I (Tarski)

– The set of fixpoints of a monotone operator $F \in L \xrightarrow{\text{mon}} L$ on a complete lattice $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ is a complete lattice [55]

– The least fixpoint is the least post-fixpoint:

$$\mathsf{lfp}^{\sqsubseteq} F = \bigsqcap\{x \in L \mid F(x) \sqsubseteq x\}$$

[55] Hence, not empty!

Fixpoint Theorem II (Kleene)

– The transfinite iterates of F on a poset $\langle L, \sqsubseteq\rangle$

- $X^0 = \bot$
- $X^{\eta+1} \triangleq F(X^\eta)$ $\eta + 1$ successor ordinal
- $X^\lambda \triangleq \bigsqcup_{\eta<\lambda} X^\eta$ λ limit ordinal

– If F is monotone and the lubs $\sqcup$ do exist [56] then the iterates are increasing, ultimately stationnary, with limit $\mathsf{lfp}^{\sqsubseteq} F$

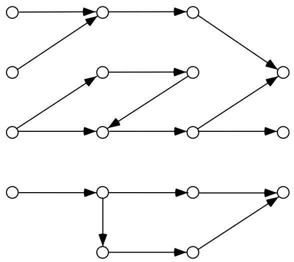
So $\mathsf{lfp}^{\sqsubseteq} F$ can always be computed iteratively.

[56] e.g. in a complete lattice or a cpo for which lubs of increasing chains do exist.

Example of Fixpoint: Reflexive Transitive Closure

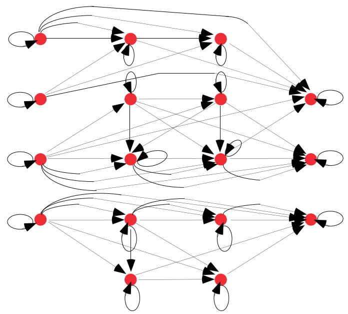
Example: Reflexive Transitive Closure

– $\tau \in \wp(\Sigma \times \Sigma)$ Transition relation



Example: Reflexive Transitive Closure (Cont'd)

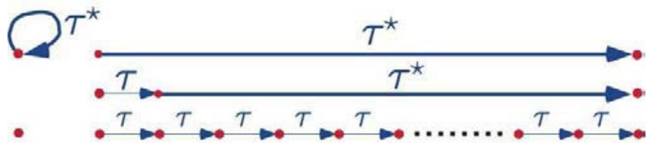
– $\tau^\star \in \wp(\Sigma \times \Sigma)$ Reflexive transitive closure



Example: Reflexive Transitive Closure (Cont'd)

$\tau^\star$ is such that

– $\tau^\star = 1_\Sigma \cup \tau \circ \tau^\star$ *fixpoint* [57]



– if $r = 1_\Sigma \cup \tau \circ r$ then $r \subseteq \tau^\star$ *least one*

so $\tau^\star = \mathsf{lfp}^{\subseteq} F$ where $F(X) = 1_\Sigma \cup \tau \circ X$

[57] $1_\Sigma \triangleq \{\langle x,\ x\rangle \mid x \in \Sigma\}$, $r_1 \circ r_2 \triangleq \{\langle x,\ z\rangle \mid \exists y : \langle x,\ y\rangle \in r_1 \wedge \langle y,\ z\rangle \in r_2\}$.

Example: Reflexive Transitive Closure (Cont'd)

$$\tau^\star = \mathsf{lfp}^{\subseteq} F \quad \text{where} \quad F(X) = 1_\Sigma \cup \tau \circ X$$
$$= \bigcup_{n \geqslant 0} t^n$$

and

- $t^0 = 1_\Sigma = \{\langle x,\, x\rangle \mid x \in \Sigma\}$,
- $t^{n+1} = t^n \circ t = t \circ t^n$

Example: Reflexive Transitive Closure (Cont'd)

$$\tau^* = \mathsf{lfp}^{\subseteq} \lambda X \cdot \tau^0 \cup X \circ \tau$$

PROOF

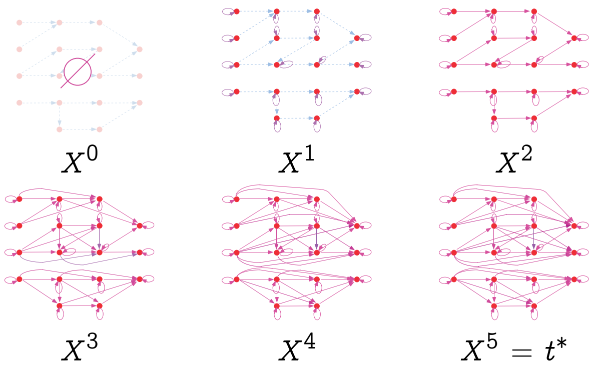
$$\begin{aligned}
X^0 &= \varnothing && \textit{basis}\\
X^1 &= \tau^0 \cup X^0 \circ \tau = \tau^0 &&\\
X^2 &= \tau^0 \cup X^1 \circ \tau = \tau^0 \cup \tau^0 \circ \tau = \tau^0 \cup \tau^1 &&\\
\ldots & \quad \ldots &&\\
X^n &= \bigcup_{0 \leq i < n} \tau^i && (\text{ nduct on hypothes s})
\end{aligned}$$

$$\begin{aligned}
X^{n+1} &= \tau^0 \cup X^n \circ \tau && \textit{induction}\\
&= \tau^0 \cup \Big(\bigcup_{0 \leq i < n} \tau^i\Big) \circ \tau &&\\
&= \tau^0 \cup \bigcup_{0 \leq i < n} (\tau^i \circ \tau) &&\\
&= \tau^0 \cup \bigcup_{1 \leq i+1 < n+1} (\tau^{i+1}) &&\\
&= \tau^0 \cup \Big(\bigcup_{1 \leq j < n+1} \tau^j\Big) \circ \tau &&\\
&= \bigcup_{0 \leq i < n+1} \tau^i &&\\
\ldots & \quad \ldots &&
\end{aligned}$$

$$\begin{aligned}
X^\omega &= \bigcup_{n \geq 0} X^n && \textit{limit}\\
&= \bigcup_{n \geq 0} \bigcup_{0 \leq i < n} \tau^i &&\\
&= \bigcup_{n \geq 0} \tau^n &&\\
&= \tau^*
\end{aligned}$$

$$
\begin{aligned}
X^{\omega+1} &= \tau^0 \cup X^{\omega} \circ \tau && \textit{convergence}\\
&= \tau^0 \cup (\bigcup_{n\geq 0} \tau^n) \circ \tau\\
&= \tau^0 \cup \bigcup_{n\geq 0} (\tau^n \circ \tau)\\
&= \tau^0 \cup \bigcup_{n\geq 0} \tau^{n+1}\\
&= \tau^0 \cup \bigcup_{k\geq 1} \tau^{k}\\
&= \bigcup_{n\geq 0} \tau^{n}\\
&= \tau^* && \blacksquare
\end{aligned}
$$

Iterates



Exact Fixpoint Abstraction

Exact Fixpoint Abstraction

- $F \in L \mapsto L$ monotonic on the complete lattice $\langle L,\ \sqsubseteq,\ \bot,\ \top,\ \sqcup,\ \sqcap\rangle$
- $\overline{F} \in \overline{L} \mapsto \overline{L}$ monotonic on $\langle \overline{L},\ \overline{\sqsubseteq},\ \overline{\bot},\ \overline{\top},\ \overline{\sqcup},\ \overline{\sqcap}\rangle$
- $\langle L,\ \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{L},\ \overline{\sqsubseteq}\rangle$ Galois connection
- $\alpha \circ F = \overline{F} \circ \alpha$ Commutation

implies

$$\alpha(\mathsf{lfp}^{\sqsubseteq} F) = \mathsf{lfp}^{\overline{\sqsubseteq}} \overline{F}$$

PROOF

Let X^δ, $\delta \in \mathbb{O}$ stationary at rank ϵ and Y^δ, $\delta \in \mathbb{O}$ stationary at ϵ' be the iterates for F and $\overline{F}$

- $\alpha(X^0) = \alpha(\bot) = \overline{\bot} = Y^0$ since $\bot \subseteq \gamma(\overline{\bot})$ so $\alpha(\bot) = \overline{\bot}$
- $\alpha(X^\delta) = Y^\delta$ induction hypothesis

$\Rightarrow \overline{F}(\alpha(X^\delta)) = \overline{F}(Y^\delta)$ $\overline{F}$ monotonic

$\Rightarrow \alpha(F(X^\delta)) = \overline{F}(Y^\delta)$ F commutation

$\Rightarrow \alpha(X^{\delta+1}) = Y^{\delta+1}$ def. iterates

- $\alpha(X^\delta) = Y^\delta$, $\delta < \lambda$, λ limit ordinal induction hypothesis

$\Rightarrow \displaystyle\bigsqcup_{\delta<\lambda} \alpha(X^\delta) = \overline{\bigsqcup_{\delta<\lambda}} Y^\delta$ lub

$\Rightarrow \alpha(\bigsqcup_{\delta<\lambda} X^\delta) = \overline{\bigsqcup_{\delta<\lambda}} Y^\delta$ α preserves lub

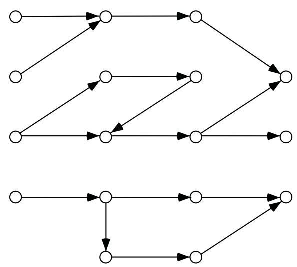
$\Rightarrow \alpha(X^\lambda) = Y^\lambda$ def. iterates

- $\alpha(\mathsf{lfp}^{\sqsubseteq} F) = \alpha(X^\epsilon) = \alpha(X^{\max(\epsilon,\epsilon')}) = Y^{\max(\epsilon,\epsilon')} = Y^{\epsilon'} = \mathsf{lfp}^{\overline{\sqsubseteq}} \overline{F}$ ∎

Example of Exact Fixpoint Abstraction: Reachable States

Example: Transition System

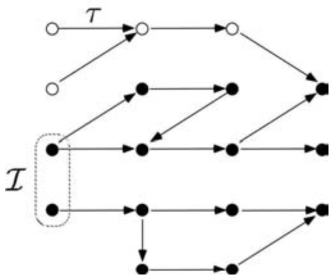
- $\langle \Sigma, \tau \rangle$ Transition system



Example: Reachable States

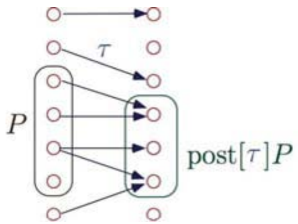
– $\mathcal{I} \subseteq \Sigma$ Initial states

– $\mathcal{R} \triangleq \{s' \mid \exists s \in \mathcal{I} : \tau^\star(s, s')\}$ Reachable states



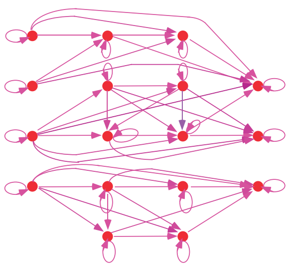
Example: Post-Image

$$\text{post}[\tau]\mathcal{I} = \{s' \mid \exists s \in \mathcal{I} : \langle s, s' \rangle \in \tau\}$$

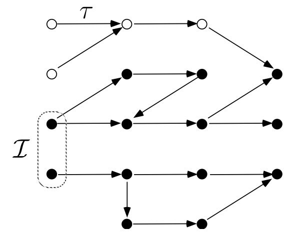


We have $\text{post}[\bigcup_{i \in \Delta} \tau^i]\mathcal{I} = \bigcup_{i \in \Delta} \text{post}[\tau^i]\mathcal{I}$ so $\alpha = \lambda\tau\cdot\text{post}[\tau]\mathcal{I}$ is the lower adjoint of a Galois connection.

Example: Reachable States



τ^*



$\text{post}[\tau^*]\mathcal{I}$

Example: Postimage Galois Connection

Given $\mathcal{I} \in \wp(\Sigma)$,

$$\langle \wp(\Sigma \times \Sigma), \subseteq \rangle \underset{\lambda\tau\cdot\text{post}[\tau]\mathcal{I}}{\overset{\gamma}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq \rangle$$

where

$$\gamma(R) \triangleq \{\langle s, s' \rangle \mid (s \in \mathcal{I}) \Rightarrow (s' \in R)\}$$

Proof

$$\begin{aligned}
& \text{post}[\tau]\mathcal{I} \subseteq R \\
\Leftrightarrow\ & \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in \tau\} \subseteq R \\
\Leftrightarrow\ & \forall s' \in \Sigma : (\exists s \in \mathcal{I} : \langle s, s'\rangle \in \tau) \Rightarrow (s' \in R) \\
\Leftrightarrow\ & \forall s', s \in \Sigma : (s \in \mathcal{I} \wedge \langle s, s'\rangle \in \tau) \Rightarrow (s' \in R) \\
\Leftrightarrow\ & \forall s', s \in \Sigma : \langle s, s'\rangle \in \tau \Rightarrow ((s \in \mathcal{I}) \Rightarrow (s' \in R)) \\
\Leftrightarrow\ & \tau \subseteq \{\langle s, s'\rangle \mid (s \in \mathcal{I}) \Rightarrow (s' \in R)\} \triangleq \gamma(R) \qquad \blacksquare
\end{aligned}$$

Example: Reachable States (Cont'd)

Reachability is an abstraction of the transitive closure:

$$\begin{aligned}
\alpha &\in \wp(\Sigma \times \Sigma) \mapsto \wp(\Sigma) \\
\alpha(t) &\triangleq \text{post}[t]\mathcal{I} \triangleq \{s' \mid \exists s \in \mathcal{I} : t(s, s')\} \\
\mathcal{R} &= \alpha(\tau^\star) \\
&= \alpha(\mathsf{lfp}^{\sqsubseteq} F) \quad \text{where} \quad F(X) = 1_\Sigma \cup \tau \circ X
\end{aligned}$$

Example: Reachable states in fixpoint form

$$\begin{aligned}
& \text{post}[\tau^*]\mathcal{I},\ \mathcal{I} \subseteq \Sigma \text{ g ven} \\
=\ & \alpha(\tau^*) \quad \text{where } \alpha(\tau) = \text{post}[\tau]\mathcal{I} = \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in \tau\} \\
=\ & \alpha(\mathsf{lfp}^{\subseteq} \lambda X \cdot \tau^0 \cup X \circ \tau) \\
=\ & \mathsf{lfp}^{\subseteq} \overline{F} \text{ ???}
\end{aligned}$$

Example: Discovering $\overline{F}$ by calculus

$$\begin{aligned}
& \alpha \circ F \\
& \alpha \circ (\lambda X \cdot \tau^0 \cup X \circ \tau) \\
=\ & \lambda X \cdot \alpha(\tau^0 \cup X \circ \tau) \\
=\ & \lambda X \cdot \alpha(\tau^0) \cup \alpha(X \circ \tau) \\
=\ & \lambda X \cdot \text{post}[\tau^0]\mathcal{I} \cup \text{post}[X \circ \tau]\mathcal{I}
\end{aligned}$$

We go on by cases.

$$\begin{aligned}
& \mathrm{post}[\tau^0]\mathcal{I} \\
= & \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in \tau^0\} \\
= & \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in \{\langle s, s\rangle \mid s \in S\}\} \\
= & \{s' \mid \exists s \in \mathcal{I}\} \\
= & \mathcal{I}
\end{aligned}$$

$$\begin{aligned}
& \mathrm{post}[X \circ \tau]\mathcal{I} \\
= & \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in (X \circ \tau)\} \\
= & \{s' \mid \exists s \in \mathcal{I} : \langle s, s'\rangle \in \{\langle s, s''\rangle \mid \exists s' : \langle s, s''\rangle \in X \wedge \langle s', s''\rangle \in \tau\}\} \\
= & \{s' \mid \exists s \in \mathcal{I} : \exists s'' \in S : \langle s, s''\rangle \in X \wedge \langle s', s''\rangle \in \tau\} \\
= & \{s' \mid \exists s'' \in S : (\exists s \in \mathcal{I} : \langle s, s''\rangle \in X) \wedge \langle s', s''\rangle \in \tau\} \\
= & \{s' \mid \exists s'' \in S : s'' \in \{s'' \mid \exists s \in \mathcal{I} : \langle s, s''\rangle \in X\} \wedge \langle s', s''\rangle \in \tau\} \\
= & \{s' \mid \exists s'' \in S : s'' \in \mathrm{post}[X]\mathcal{I} \wedge \langle s', s''\rangle \in \tau\} \\
= & \mathrm{post}[\tau](\mathrm{post}[X]\mathcal{I}) \\
= & \mathrm{post}[\tau](\alpha(X))
\end{aligned}$$

$$\begin{aligned}
\alpha \circ F &= \alpha \circ (\lambda X \cdot \tau^0 \cup X \circ \tau) \\
&= \ldots \\
&= \lambda X \cdot \mathrm{post}[\tau^0]\mathcal{I} \cup \mathrm{post}[X \circ \tau]\mathcal{I} \\
&= \lambda X \cdot \mathcal{I} \cup \mathrm{post}[\tau](\alpha(X)) \\
&= \lambda X \cdot \overline{F}(\alpha(X))
\end{aligned}$$

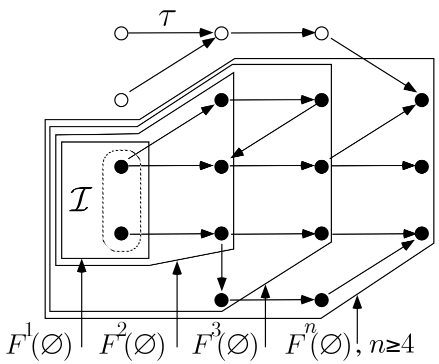
by defining:

$$\overline{F} = \lambda X \cdot \mathcal{I} \cup \mathrm{post}[\tau](X)$$

proving:

$$\mathrm{post}[\tau^*](\mathcal{I}) = \mathsf{lfp}^{\subseteq}\, \lambda X \cdot \mathcal{I} \cup \mathrm{post}[\tau](X)$$

Example: iteration



Fixpoint Approximation

Fixpoint Approximation

- $F \in L \mapsto L$ monotonic on the complete lattice $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$
- $\overline{F} \in \overline{L} \mapsto \overline{L}$ on $\langle \overline{L}, \overline{\sqsubseteq} \rangle$
- $\gamma \in \overline{L} \mapsto L$, monotonic, such that $F \circ \gamma \sqsubseteq \gamma \circ \overline{F}$

implies

$$\overline{F}(X) \mathrel{\overline{\sqsubseteq}} X \Rightarrow \mathsf{lfp}^{\sqsubseteq} F \sqsubseteq \gamma(X)$$

PROOF

$$
\begin{array}{lll}
 & \overline{F}(X) \mathrel{\overline{\sqsubseteq}} X & \\
\Longrightarrow & \gamma(\overline{F}(X)) \sqsubseteq \gamma(X) & \gamma \text{ monotone} \\
\Longrightarrow & F(\gamma(X)) \sqsubseteq \gamma(X) & F \circ \gamma \sqsubseteq \gamma \circ \overline{F} \\
\Longrightarrow & \mathsf{lfp}^{\sqsubseteq} F \sqsubseteq \gamma(X) & \text{Tarski} \quad \blacksquare
\end{array}
$$

Example: Sign Analysis

1) Reachable states of `X` in

```
0: X := 100;
1: while X > 0 do
2:   X := X - 1;
3: od;
4:
```

$$
\begin{aligned}
X_0 &= \mathbb{Z} \\
X_1 &= \{100\} \cup X_3 \\
X_2 &= X_1 \cap \{z \in \mathbb{Z} \mid z > 0\} \\
X_3 &= \{z - 1 \mid z \in X_2\} \\
X_4 &= X_1 \cap \{z \in \mathbb{Z} \mid z \leqslant 0\}
\end{aligned}
$$

of the form:

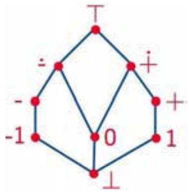
$$\vec{X} = \vec{F}(\vec{X}) \quad \text{where}$$
$$\vec{X} = \langle X_0, X_1, \ldots, X_4 \rangle$$

Example: Sign Analysis (Cont'd)

2) Overapproximation by the sign of X in

```
0: X := 100;
1: while X > 0 do
2:    X := X - 1;
3: od;
4:
```

$$\begin{aligned}\overline{X}_0 &= \top\\ \overline{X}_1 &= + \sqcup \overline{X}_3\\ \overline{X}_2 &= \overline{X}_1 \sqcap +\\ \overline{X}_3 &= \overline{X}_2 \ominus 1\\ \overline{X}_4 &= \overline{X}_1 \sqcap \dot{-}\end{aligned}$$



Widening/Narrowing

Reference

[POPL '77] P. Cousot & R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

Example: Sign Analysis (Cont'd)

3) Iterative resolution

$$\begin{aligned}\overline{X}_0 &= \top\\ \overline{X}_1 &= + \sqcup \overline{X}_3\\ \overline{X}_2 &= \overline{X}_1 \sqcap +\\ \overline{X}_3 &= \overline{X}_2 \ominus 1\\ \overline{X}_4 &= \overline{X}_1 \sqcap \dot{-}\end{aligned}$$

of the form

$$\vec{\overline{X}} = \vec{\overline{F}}(\vec{\overline{X}}) \quad \text{where}$$
$$\vec{\overline{X}} = \langle \overline{X}_0, \overline{X}_1, \ldots, \overline{X}_4 \rangle$$

Iterate	0	1	2	3
$\overline{X}_0$	$\bot$	$\top$	$\top$	$\top$
$\overline{X}_1$	$\bot$	$+$	$\dot{+}$	$\dot{+}$
$\overline{X}_2$	$\bot$	$+$	$+$	$+$
$\overline{X}_3$	$\bot$	$\dot{+}$	$\dot{+}$	$\dot{+}$
$\overline{X}_4$	$\bot$	0	0	0

Convergence Problem

– The iterates of a monotone transformer $\overline{F} \in \overline{L} \mapsto \overline{L}$ on a cpo $\langle \overline{L}, \sqsubseteq \rangle$ may not converge

– The Interval analysis of

```
x:=1; while true do x:=x+2 od
```

consists in solving

$$X = [1,1] \sqcup (X + [2,2]) \ .$$

Iteratively,

$$\varnothing,\ [1,1],\ [1,3],\ \ldots,\ [1,2n+1],\ \ldots$$

Convergence Hypotheses

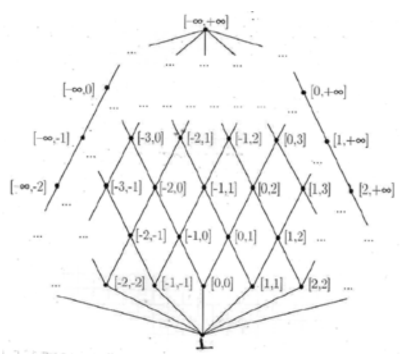
– We can assume $\overline{L}$
  - to be finite[58], or
  - to satisfy the ascending chain condition (ACC) (so $X^0 = \bot, \ldots, X^{n+1} = \overline{F}(X^n), \ldots$ which is ascending is finite)

– This is provably less precise than using $\overline{L}$ not satisfying the ACC

[58] As in Boolean model-checking

Interval Abstract Domain

The interval abstract domain does not satisfy the ACC



Enforcing Convergence

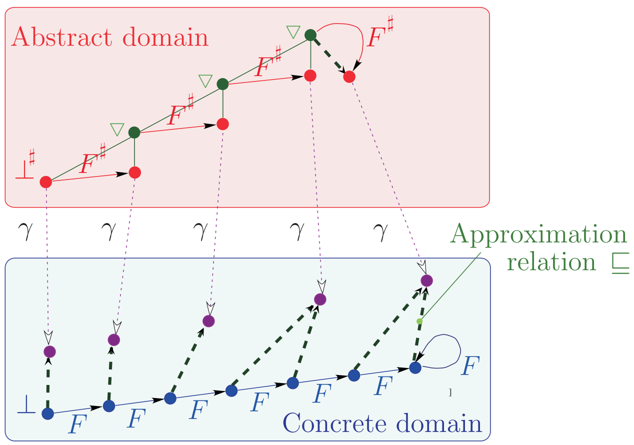
– The convergence of the iterates

$$X^0 = \bot, \ldots, X^{n+1} = \overline{F}(X^n), \ldots$$

of a monotone transformer $\overline{F} \in \overline{L} \mapsto \overline{L}$ on a cpo $\langle \overline{L}, \sqsubseteq \rangle$ can be forced to converge to an over-approximation of $\mathsf{lfp}^{\sqsubseteq}\,\overline{F}$ using a widening

- $X^0 = \bot$
- $X^{n+1} = X^n \mathbin{\nabla} \overline{F}(X^n)$ if $\overline{F}(X^n) \not\sqsubseteq X^n$
- $X^{\ell+1} = X^\ell$ convergence to X^ℓ if $\overline{F}(X^\ell) \sqsubseteq X^\ell$

Convergence acceleration with widening



Definition of the Widening

– The widening overapproximates:

$$x \sqsubseteq x \mathbin{\nabla} y \qquad y \sqsubseteq x \mathbin{\nabla} y$$

– The widening enforces convergence:

for all increasing chains

$$x^0 \sqsubseteq x^1 \sqsubseteq \ldots,$$

the increasing chain defined by

$$y^0 = x^0, \ldots, y^{i+1} = y^i \mathbin{\nabla} x^{i+1}, \ldots$$

is not strictly increasing.

Example of Widening for the Interval Abstract Domain

– $\overline{L} = \{\bot\} \cup \{[\ell, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{+\infty\} \wedge \ell \leq u\}$

– The widening extrapolates unstable bounds to infinity:

$$\bot \mathbin{\nabla} X = X$$
$$X \mathbin{\nabla} \bot = X$$
$$[\ell_0, u_0] \mathbin{\nabla} [\ell_1, u_1] = [\text{f } \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0, \text{ f } u_1 > u_0 \text{ then } +\infty \text{ else } u_0]$$

Example of Iteration with Widening

– `x:=1; while (x <= 1000) do x:=x+2 od`

– $X = ([1,1] \sqcup (X + [2,2])) \sqcap [-\infty, 1000] = \overline{F}(X)$

– $X^0 = \bot$,

– $X^1 = X^0 \mathbin{\nabla} (([1,1] \sqcup (X^0 + [2,2])) \sqcap [-\infty, 1000])$
$= \bot \mathbin{\nabla} [1,1] = [1,1]$

– $X^2 = X^1 \mathbin{\nabla} (([1,1] \sqcup (X^1 + [2,2])) \sqcap [-\infty, 1000])$
$= [1,1] \mathbin{\nabla} [3,3] = [1, +\infty]$

convergence[59] accelerated to $X^\ell = [1, +\infty]$, $\ell = 2$

[59] $(([1,1] \sqcup (X^2 + [2,2])) \sqcap [-\infty, 1000]) = [11000] \sqsubseteq [1, +\infty] = X^2$

Soundness and Convergence of the Iterates with Widening

– Soundness: convergence to X^ℓ such that $\overline{F}(X^\ell) \sqsubseteq X^\ell$ so $\mathsf{lfp}^{\sqsubseteq} = \sqcap\{Y \mid \overline{F}(Y) \sqsubseteq Y\}$[60] $\sqsubseteq X^\ell$

– Convergence: $X^0 = \bot, \ldots, X^{i+1} = X^i \mathbin{\nabla} \overline{F}(X^i), \ldots$ is not strictly increasing.

[60] Tarski's fixpoint theorem

Widening is not Monotone

- Not monotone.
- For example $[0, 1] \sqsubseteq [0, 2]$ but $[0, 1] \mathbin{\nabla} [0, 2] = [0, +\infty]$ $\not\sqsubseteq$ $[0, 2] = [0, 2] \mathbin{\nabla} [0, 2]$
- The limit X^ℓ depends upon the iteration strategy!

Widening Cannot Be Monotone

Proof by contradiction:

- Let ∇ be a widening operator
- Define $x \mathbin{\nabla'} y = \text{if } y \sqsubseteq x \text{ then } x \text{ else } x \mathbin{\nabla} y$
- Assume $x \sqsubset y = \overline{F}(x)$ (during iteration)

  then: $\begin{array}{lllll} x \mathbin{\nabla'} y & = & x \mathbin{\nabla} y \sqsupseteq y & & (\textit{soundness}) \\ \sqsubseteq \quad\;\; \sqsubseteq & & \sqsubseteq & & (\textit{monotony hypothesis}) \\ y \mathbin{\nabla'} y & = & y & & (\textit{termination}) \end{array}$

$\Rightarrow$ $x \mathbin{\nabla} y = y$, by antisymmetry!

$\Rightarrow$ $x \mathbin{\nabla} \overline{F}(x) = \overline{F}(x)$ during iteration $\Rightarrow$ convergence cannot be enforced with monotone widening (so widening by finite abstraction is less powerful!)

Improving a Fixpoint Overapproximation

- If $X = \overline{F}(X)$ and $X \sqsubseteq \overline{F}(Y) \sqsubseteq Y$ then
- $X = \overline{F}(X) \sqsubseteq \overline{F}^n(Y) \sqsubseteq \overline{F}^{n-1}(Y) \sqsubseteq \ldots \sqsubseteq Y$ ind. hyp.
- Hence $X = \overline{F}(X) \sqsubseteq \overline{F}(\overline{F}^n(Y)) \sqsubseteq \overline{F}(\overline{F}^{n-1}(Y)) \sqsubseteq \ldots \sqsubseteq \overline{F}(Y)$ by monotony
- So $X = \overline{F}(X) \sqsubseteq \overline{F}^{n+1}(Y) \sqsubseteq \overline{F}^n(Y)) \sqsubseteq \ldots \sqsubseteq \overline{F}(Y) \sqsubseteq Y$ by hyp.
- Proving $X = \overline{F}(X) \sqsubseteq \displaystyle\bigsqcap_{n \geq 0} \overline{F}^n(Y)$

  by def. $\sqcap$ and $\overline{F}^0(Y) = Y$

Convergence Problem, Again

- The decreasing iterates $\overline{F}^n(Y)$, $n \geq 0$ may not converge
- We can assume
  - $\overline{L}$ to be finite[61], or
  - to satisfy the descending chain condition (DCC) (so $X^0 = Y, \ldots, X^{n+1} = \overline{F}(X^n), \ldots$ which is descending is finite)
- This is provably less precise than using $\overline{L}$ not satisfying the DCC

[61] As in Boolean model-checking

Enforcing Convergence

- The convergence of the iterates (where $\overline{F}(X^\ell) \sqsubseteq X^\ell$)

$$Y^0 = X^\ell, \ldots, Y^{n+1} = \overline{F}(Y^n), \ldots,$$

of a monotone $\overline{F} \in \overline{L} \mapsto \overline{L}$ on a cpo $\langle \overline{L}, \sqsubseteq \rangle$ can be forced to converge to an over-approximation of $\mathsf{lfp}^{\sqsubseteq} \overline{F}$ using a narrowing
  - $Y^0 = X^\ell$
  - $Y^{n+1} = Y^n \,\Delta\, \overline{F}(Y^n)$ if $\overline{F}(Y^n) \neq Y^n$
  - $Y^{\eta+1} = Y^\eta$ convergence to Y^η if $\overline{F}(Y^\eta) = Y^\eta$

Definition of the Narrowing

- A narrowing operator Δ is such that:
  - $\forall x, y : x \sqsubseteq y \Longrightarrow x \sqsubseteq x \,\Delta\, y \sqsubseteq y$;
  - for all decreasing chains

$$x^0 \sqsupseteq x^1 \sqsupseteq \ldots$$

the decreasing chain defined by

$$y^0 = x^0, \ldots, y^{i+1} = y^i \,\Delta\, x^{i+1}, \ldots$$

is not strictly decreasing.

Example of Narrowing for the Interval Abstract Domain

- The narrowing improves infinite bounds only:

$$\begin{array}{l} \bot \,\Delta\, X = \bot \\ [\ell_0,\, u_0] \,\Delta\, [\ell_1,\, u_1] = [(\ell_0 = -\infty \;?\; \ell_1 : \ell_0)^{62}, \\ \qquad (u_0 = +\infty \;?\; u_1 : u_0)] \end{array}$$

Example of Iteration with Narrowing

- `x:=1; while (x <= 1000) do x:=x+2 od`
- $X = ([1,1] \sqcup (X + [2,2])) \sqcap [-\infty, 1000] = \overline{F}(X)$
- $Y^0 = X^\ell = [1, +\infty]$
- $Y^1 = Y^0 \Delta(([1,1] \sqcup (Y^0 + [2,2])) \sqcap [-\infty, 1000]) = [1, +\infty] \Delta [1, 1000] = [1, 1000]$
- $\overline{F}(Y^1) = ([1,1] \sqcup (Y^1 + [2,2])) \sqcap [-\infty, 1000] = Y^1 = [1, 1000]$

convergence accelerated to $Y^\eta = [1, +1000]$, $\eta = 1$

Soundness of the Iterates with Narrowing

- $\mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq Y^0 = X^\ell$ — basis (with $\overline{F}(X^\ell) \sqsubseteq X^\ell$)
- $\Rightarrow \mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq \overline{F}(Y^0) = \overline{F}(X^\ell) \sqsubseteq X^\ell = Y^0$ — monotony
- $\mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq \overline{F}(Y^n) \sqsubseteq Y^n \sqsubseteq X^\ell$ — ind. hyp.
- $\Rightarrow \mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq \overline{F}(Y^n) \sqsubseteq Y^n \mathbin{\Delta} \overline{F}(Y^n) = Y^{n+1} \sqsubseteq Y^n \sqsubseteq X^\ell$ — def. iterates with Δ
- $\Rightarrow \mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq \overline{F}(Y^{n+1}) \sqsubseteq \overline{F}(Y^n) \sqsubseteq Y^n \sqsubseteq X^\ell$ — monotony
- $\Rightarrow \mathsf{lfp}^{\sqsubseteq}\, \overline{F} \sqsubseteq \overline{F}(Y^{n+1}) \sqsubseteq Y^n \mathbin{\Delta} \overline{F}(Y^n) = Y^{n+1} \sqsubseteq X^\ell$ — def. Δ & transitivity

One can stop at any iteration, all overapproximate $\mathsf{lfp}^{\sqsubseteq}\, \overline{F}$!

Convergence of the Iterates with Narrowing

- The decreasing chain defined by

$$Y^0 = X^\ell, \ldots, Y^{i+1} = Y^i \mathbin{\Delta} \overline{F}(Y^i), \ldots$$

is not strictly decreasing $\Rightarrow$ convergence.

Widening/Narrowing May be Too Imprecise

- `x:=1; while (x <> 1000) do x:=x+2 od`
- $X = ([1,1] \sqcup (X + [2,2])) \sqcap [-\infty, 999] \sqcup [1001, +\infty] = [1,1] \sqcup (X + [2,2])$
- Iteration with widening: $[1, +\infty]$
- Iteration with narrowing: $[1, +\infty] \mathbin{\Delta} ([1,1] \sqcup ([1,+\infty] + [2,2])) = [1, +\infty]$ no improvement!

$\Rightarrow$ Need to widen to threshold 1000!

Improving the Widening: Cutpoints

- Widen only at loop cutpoints (only once around each loop)
- Astrée proceeds by structural induction on the program abstract syntax (so inner loops are stabilized first)

Improving the Widening: Thresholds

– Extrapolate to thresholds in $T \supseteq \{-\infty, +\infty\}$:

$$[\ell_0,\, u_0] \mathbin{\nabla} [\ell_1,\, u_1] = [\, \mathrm{f}\ \ell_1 \geqslant \ell_0 \ \mathrm{then}\ \ell_0, \\ \mathrm{else}\ \max\{\ell \in \mathrm{T} \mid \ell \leqslant \ell_1\}, \\ \mathrm{f}\ u_1 < u_0 \ \mathrm{then}\ u_0 \\ \mathrm{else}\ \mathrm{m\ n}\{u \in \mathrm{T} \mid u_1 \leqslant u\}]$$

– ASTRÉE's widening thresholds (parametrizable): `-1, 0, 1, 2, 3, 4, 5, 17, 41, 32767, 32768, 65535, 65536;`

Improving the Widening: Delays

– Do not widen an interval (more generally an abstract predicate) at each iteration, but delay the widening to given numbers of changes

– ASTRÉE's widening delays (parametrizable): `3, 6, 9, 10, 12, ..., 150, 150 + 1 * Z`

Improving the Widening: History-based Widening

– Do not widen/narrow an abstract predicate which was computed for the first time since the last widening/narrowing;

– Otherwise, do not widen/narrow the abstract values of variables which were not "assigned to" [63] since the last widening/narrowing.

[63] more precisely which did not appear in abstract transformers corresponding to an assignment to these variables.

Example with Widening/Warrowing at Cut-points:

```
{ i:_0_; j:_0_ }
  i := 1;
{ i:[1,+oo]; j:[1,+oo]? }
  while (i < 1000) do
    { i:[1,999]; j:[1,+oo]? }
      j := 1;
    { i:[1,+oo]; j:[1,+oo] }
      while (j < i) do
        { i:[2,+oo]; j:[1,1073741822] }
          j := (j + 1)
        { i:[2,+oo]; j:[2,+oo] }
      od;
    { i:[1,+oo]; j:[1,+oo] }
      i := (i + 1);
    { i:[2,+oo]; j:[1,+oo] }
  od
{ i:[1000,+oo]; j:[1,+oo]? }
```

– `_0_` is the "uninitialized" value

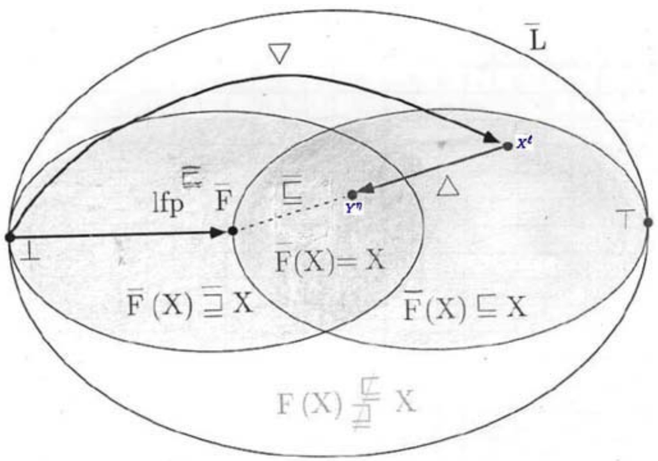
– `+oo` = 2147483647 (maximum integer)

– $[\ell, u]? = [\ell, u] \cup \{\_0\_\}$

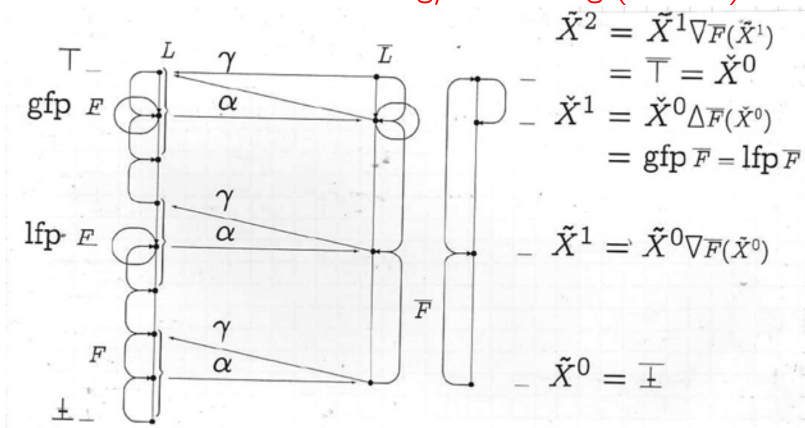
Example with History-Based Widening/Narrowing:

```
{ i:_O_; j:_O_ }
  i := 1;
{ i:[1,1000]; j:[1,999]? }
  while (i < 1000) do
    { i:[1,999]; j:[1,999]? }
      j := 1;
    { i:[1,999]; j:[1,999] }
      while (j < i) do
        { i:[2,999]; j:[1,998] }
          j := (j + 1)
        { i:[2,999]; j:[2,999] }
      od;
    { i:[1,999]; j:[1,999] }
      i := (i + 1);
    { i:[2,1000]; j:[1,999] }
  od
{ i:[1000,1000]; j:[1,999]? }
```


Iteration with Widening/Narrowing



Iteration with Widening/Narrowing (Cont'd)



Widening/narrowing are not Dual

- The iteration with widening starts from below the least fixpoint and stabilizes above to a postfixpoint;
- The iteration with narrowing starts from above the least fixpoint and stabilizes above;
- The iteration with dual widening starts from above the greatest fixpoint and stabilizes below to a prefixpoint;
- The iteration with dual narrowing starts from below the greatest fixpoint and stabilizes below;

Widening/Narrowing and their Duals

	Iteration starts from	Iteration stabilizes
Widening ∇	below	above
Narrowing Δ	above	above
Dual widening $\widetilde{\nabla}$	above	below
Dual narrowing $\widetilde{\Delta}$	below	below

Whence that's four different notions.

On Monotony

– The abstract transformer $\overline{F}$ need not be monotone [64]

– So it can contain ∇ and Δ

– The monotony is required only for concrete transformers (which is the case since predicate transformers are monotonic)

64 Contrary to what is often assumed for simplicity.

Non-Existence of Finite Abstractions

Let us consider the infinite family of programs parameterized by the *mathematical constants* n_1, n_2 ($n_1 \leq n_2$):

$\mathtt{X} := n_1$; **while** $\mathtt{X} \leq n_2$ do $\mathtt{X} := \mathtt{X} + 1$; **od**

– An Interval analysis with widening/narrowing will discover the loop invariant $\mathtt{X} \in [n_1, n_2]$;

– The abstract domain must contain all such intervals to avoid false alarm for all programs in the family;

$\Rightarrow$ No single finite abstract domain will do for all programs!

References

[34] P. Cousot & R. Cousot. – Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *PLILP '92*, LNCS 631, pp. 269–295. Springer, 1992.

– Yes, but predicate abstraction with refinement will do (?) for each program in the family (since it is equivalent to a widening) [65]!

– Indeed no, since:

  - Predicate abstraction is unable to express limits of infinite sequences of predicates;
  - Not all widening proceed by eliminating constraints:
  - A narrowing is necessary anyway in the refinement loop (to avoid infinitely many refinements);
  - Not speaking of costs!

65 T. Ball, A. Podelski, S.K. Rajamani. Relative Completeness of Abstraction Refinement for Software Model Checking. TACAS 2002: 158-172.

5. Static Analysis

Principle of Static Analysis

Static Analysis

– Static code analysis is the analysis of computer system
  - by direct inspection of the source or object code describing this system
  - with respect to a semantics of this code (no user-provided model)
  - without executing programs as in dynamic analysis.

– The static code analysis is performed by an automated tool, as opposed to program understanding or program comprehension by humans.

Verification by Static Analysis

– The proof

$$\mathcal{C}[\![\mathtt{p}]\!] \subseteq P$$

is done in the abstract

$$\mathcal{C}^\sharp[\![\mathtt{p}]\!] \sqsubseteq P^\sharp ,$$

which involves the static analysis of $\mathtt{p}$ that is the effective computation of the abstract semantics

$$\mathcal{C}^\sharp[\![\mathtt{p}]\!] ,$$

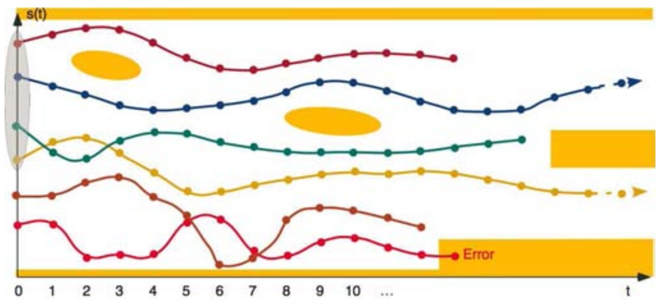
as formalized by abstract interpretation [35], [36].

References

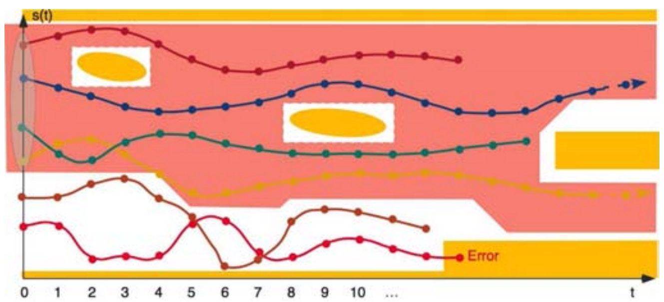
[35] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 1978.

[36] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. 6^{th} *ACM POPL*, 269–282, 1979.

Semantics and Specification

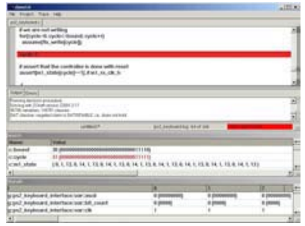


Abstract Semantics and Verification



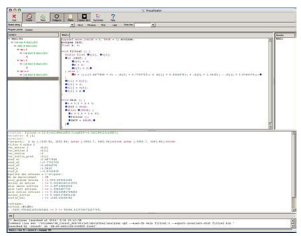
Example 1: CBMC

- CBMC is a Bounded Model Checker for ANSI-C programs (started at CMU in 1999).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, also supports dynamic memory allocation using `malloc`.
- Done by unwinding the loops in the program and passing the resulting equation to a SAT solver.
- **Problem (a.o.): does not scale up!**



Example 2: ASTRÉE

- ASTRÉE is an abstract interpretation-based static analyzer for ANSI-C programs (started at ENS in 2001).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, does not support dynamic memory allocation.
- Done by abstracting the reachability fixpoint equations for the program operational semantics.
- **Advantage (a.o.): does scale up!**



Design of a Static Analyzer

1. Design of the concrete semantics of programs
2. Definition of properties of programs (collecting semantics)
3. Definition of properties to be verified (specification)
4. Choice of abstractions and their combinations
5. Design of the abstract semantics of programs (iterator and abstract properties)
6. Design of the abstract semantics overapproximation (iteration acceleration)
7. Design of the abstract specification verification algorithm (proof)

Model versus Property and Program versus Language-based Abstraction

Property-based Abstraction

– Property-based abstraction over approximate the collecting semantics in the abstract
  - $\mathcal{C}[\![\mathbb{p}]\!] = \{\mathcal{S}[\![\mathbb{p}]\!]\} \in \mathcal{P}$ collecting semantics
  - $\langle \mathcal{P}, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{P}^\sharp, \sqsubseteq^\sharp \rangle$ abstraction
  - $\mathcal{C}^\sharp[\![\mathbb{p}]\!] \in \mathcal{P}^\sharp$ abstract semantics
  - $\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma(\mathcal{C}^\sharp[\![\mathbb{p}]\!])$ soundness

$\Rightarrow$ an abstract proof $(\mathcal{C}^\sharp[\![\mathbb{p}]\!] \sqsubseteq^\sharp P^\sharp)$ is valid in the concrete $(\mathcal{C}[\![\mathbb{p}]\!] \subseteq \gamma(P^\sharp))$.

Model-based Abstraction

– Let $\langle \iota, \tau \rangle$ be a transition system model of a software or hardware system $\mathbb{p} \in \mathbb{P}$ (so that $\mathcal{S}[\![\mathbb{p}]\!] \triangleq \gamma_{\iota\tau}(\langle \iota, \tau \rangle))$.

– A model-based abstraction is an abstract transition system $\langle \iota^\sharp, \tau^\sharp \rangle$ which over-approximates $\langle \iota, \tau \rangle$ (so that, up to concretization, $\iota \subseteq \iota^\sharp$ and $\tau \subseteq \tau^\sharp$).

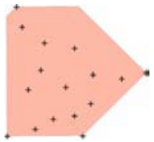
– The set of reachable abstract states for $\langle \iota^\sharp, \tau^\sharp \rangle$ over-approximate the reachable concrete states of $\langle \iota, \tau \rangle$

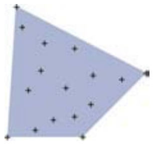
– Hence the model-based abstractions yields sound abstractions of the concrete reachability states.

Is the model-based abstraction "adequate"?

Limitations of Model-based Abstractions

– Some abstractions defined by a Galois connection of sets of (reachable) states are not be model-based abstractions, in particular when the abstract domain is not a representable as a powerset of states, e.g.





Octagons [38] Polyhedra [37]

References

[37] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. 5th POPL, pp. 84–97, ACM Press, 1978.

[38] A. Miné. The octagon abstract domain. *Higher-Order and Symb. Comp.*, 19:31–100, 2006.

Program-based versus Language-based Abstraction

– An abstraction specific to a given program can always be refined to be complete using a finite abstract domain [39].

– This is impossible in general for a language-based abstraction for which infinite abstract domains have been shown to always produce better results [40].

References

[39] P. Cousot. Partial completeness of abstract fixpoint checking. *SARA*, LNAI 1864, 1–25. Springer, 2000.

[40] P. Cousot & R. Cousot. – Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *PLILP '92*, LNCS 631, pp. 269–295. Springer, 1992.

Program-based versus Language-based Abstraction

– Static analysis has to define an abstraction $\alpha[\![\mathbb{p}]\!]$ for all programs $\mathbb{p} \in \mathbb{P}$ of a language $\mathbb{P}$.

– This is different from defining an abstraction specific to a *given* program (or model).

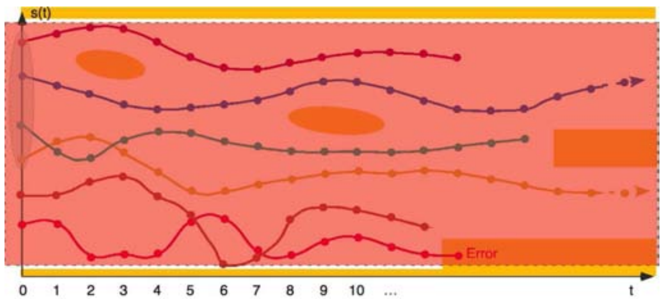
False Alarms

False Alarms

- Static analysis being undecidable, it relies on incomplete language-based abstractions.
- A false alarm is a case when a concrete property holds but this cannot be proved in the abstract for the given abstraction.

False Alarm

- An example in reachability analysis is when no inductive invariant can be expressed in the abstract.



False Alarms (Cont'd)

- The experience of ASTRÉE (`www.astree.ens.fr`, [41]) shows that it is possible to design precise language-based abstractions which produce no false alarm on a well defined *families of programs* [66].
- Nevertheless, by indecidability, the analyzer will produce false alarms on infinitely many programs (which can even be generated automatically).

References

[41] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. *ACM PLDI*, 196–207, 2003.

[66] Synchronous, time-triggered, real-time, safety critical, embedded software written or automatically generated in the C programming language for ASTRÉE.

Abstract Domains

Abstract Domain

An abstract domain defines

– The computer representation of abstract properties (corresponding to given concrete properties)

– The abstract operations requested by the iterator (including lattice operations $\sqsubseteq$, ..., operations involved in the abstract transformer $\overline{F}$, convergence acceleration ∇, etc)

An Abstract Domain in ASTRÉE [67]

```
module type INTEGER =
sig
  type t
  val zero: t
  val one: t
  val is_zero: t->bool
  val max: t->t->t
  val min: t->t->t
  val add: t->t->t
  val sub: t->t->t
  val mul: t->t->t
  val div: t->t->t
  val rem: t->t->t
  val neg: t->t
  val sgn: t->int
  val abs: t->t
  val compare: t->t->int
  val print: Format.formatter->t->unit
  val pred: t->t
  val succ: t->t
  val equal: t->t->bool
  val widening_sequence: t list
  val quick_widening_sequence: t list
  val shift_left: t->int->t
  val shift_right: t->int->t
  val shift_right_logical: t->int->t
  val to_int: t->int
  val of_int: int->t
  val logand: t->t->t
  val logor: t->t->t
  val logxor: t->t->t
  val lognot: t->t
  val to_string: t->string
end
```

67 More precisely, interface with the iterator.

Design of Abstractions

Design of Abstractions

– The design of a sound and precise language-based abstraction is difficult.

– First from a mathematical point of view, one must discover the appropriate set of abstract properties that are needed to represent the necessary inductive invariants.

– Of course mathematical completion techniques could be used [42] but because of undecidability, they do not terminate in general.

References

[42] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

Design of Abstractions (Cont'd)

- Second, from a computer-science point of view, one must find an appropriate computer representation of abstract properties and abstract transformers.
- Universal representations (e.g. using symbolic terms, automata or BDDs) are in general inefficient
- The discovery of appropriate computer representations is far from being automatized.

Local versus Global Abstractions

- A simple approach to static analysis is to use the same global abstraction everywhere in the program, which hardly scales up.
- More sophisticated abstractions, as used in ASTRÉE are not uniform, different local abstractions being in different program regions [43].

References

[43] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. *ACM PLDI*, 196–207, 2003.

Multiple versus Single Abstractions

- Because of the complexity of abstractions, it is simpler to design a precise abstraction by composing many elementary abstractions which are simple to understand and implement.
- These abstractions could hardly be encoded efficiently using a universal representation of program properties as found in theorem provers, proof assistants or model-checkers.

6. The ASTRÉE Static Analyzer

www.astree.ens.fr/ Nov. 2001—Nov. 2007

Project Members




Bruno Blanchet[68] Patrick Cousot Radhia Cousot Jérôme Feret




Laurent Mauborgne Antoine Miné David Monniaux[69] Xavier Rival

[68] Nov. 2001 — Nov. 2003.
[69] Nov. 2001 — Aug. 2007.

Programs Analyzed by Astrée and their Semantics

Programs analysed by Astrée

- Application Domain: large safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.
- C programs:
  - with
    - basic numeric datatypes, structures and arrays
    - pointers (including on functions),
    - floating point computations
    - tests, loops and function calls
    - limited branching (forward `goto`, `break`, `continue`)

- with (cont'd)
  - `union` NEW [Min06a]
  - pointer arithmetics & casts NEW [Min06a]
- without
  - dynamic memory allocation
  - recursive function calls
  - unstructured/backward branching
  - conflicting side effects
  - C libraries, system calls (parallelism)

Such limitations are quite common for embedded safety-critical software.

The Class of Considered Periodic Synchronous Programs

```
declare volatile input, state and output variables;
initialize state and output variables;
loop forever
   - read volatile input variables,
   - compute output and state variables,
   - write to output variables;
  __ASTREE_wait_for_clock ();
end loop
```

Task scheduling is static:

– Requirements: the only interrupts are clock ticks;
– Execution time of loop body less than a clock tick, as verified by the aiT WCET Analyzers [FHL+01].

Challenging aspects

– Size: 100/1000 kLOC, 10/150 000 global variables
– Floating point computations
including interconnected networks of filters, non linear control with feedback, interpolations...
– Interdependencies among variables:
  - Stability of computations should be established
  - Complex relations should be inferred among numerical and boolean data
  - Very long data paths from input to outputs

Concrete Operational Semantics

– International norm of C (ISO/IEC 9899:1999)
– *restricted by* implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
– *restricted by* user-defined programming guidelines (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
– *restricted by* program specific user requirements (e.g. `assert`, execution stops on first runtime error [70])

[70] semantics of C unclear after an error, equivalent if no alarm

Different Classes of Run-time Errors

1. Errors terminating the execution [71]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.
2. Errors not terminating the execution with predictable outcome [72]. ASTRÉE warns and continues with worst-case assumptions.
3. Errors not terminating the execution with unpredictable outcome [73]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.

⇒ ASTRÉE is sound with respect to C standard, unsound with respect to C implementation, unless no false alarm.

[71] floating-point exceptions e.g. (invalid operations, overflows, etc.) when traps are activated
[72] e.g. overflows over signed integers resulting in some signed integer.
[73] e.g. memory corruptions.

Trace semantics

– From this small-step semantics we derive a discrete-time complete trace semantics [74];

– This trace semantics is abstracted into many different abstract properties as implemented by various abstract domains defining compat finite representations of specifica properties;

– ASTRÉE computes a weak reduced product for these abstractions [75].

[74] posibly limited, for synchronous control/command programs, to a maximum number of clock ticks (`__ASTREE_wait_for_clock(())`, as specified by a configuration file.

[75] for efficiency, only a number of useful reductions are performed amongst all possible ones, via communications between abstract domains.

Specification Proved by ASTRÉE

Implicit Specification: Absence of Runtime Errors

– No violation of the norm of C (e.g. array index out of bounds, division by zero)

– No implementation-specific undefined behaviors (e.g. maximum short integer is 32767, NaN)

– No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)

– No violation of the programmer assertions (must all be statically verified).

Characteristics of ASTRÉE

Characteristics of the ASTRÉE Analyzer

Sound: – ASTRÉE is a bug eradicator: finds all bugs in a well-defined class (runtime errors)

- ASTRÉE is not a bug hunter: finding some bugs in a well-defined class (e.g. by *bug pattern detection* like FindBugs™, PREfast or PMD)
- ASTRÉE is exhaustive: covers the whole state space (≠ MAGIC, CBMC)
- ASTRÉE is comprehensive: never omits potential errors (≠ UNO, CMC from coverity.com) or sort most probable ones to avoid overwhelming messages (≠ Splint)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Multiabstraction: uses many numerical/symbolic abstract domains (≠ symbolic constraints in Bane or the canonical abstraction of TVLA)

Infinitary: all abstractions use infinite abstract domains with widening/narrowing (≠ model checking based analyzers such as Bandera, Bogor, Java PathFinder, Spin, VeriSoft)

Efficient: always terminate (≠ counterexample-driven automatic abstraction refinement BLAST, SLAM)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Static: compile time analysis (≠ run time analysis Rational Purify, Parasoft Insure++)

Program Analyzer: analyzes programs not micromodels of programs (≠ PROMELA in SPIN or Alloy in the Alloy Analyzer)

Automatic: no end-user intervention needed (≠ ESC Java, ESC Java 2), or PREfast (annotate functions with intended use)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Extensible/Specializable: can easily incorporate new abstractions (and reduction with already existing abstract domains) (≠ general-purpose analyzers PolySpace Verifier)

Domain-Aware: knows about control/command (e.g. digital filters) (as opposed to specialization to a mere programming style in C Global Surveyor)

Parametric: the precision/cost can be tailored to user needs by options and directives in the code

Characteristics of the ASTRÉE Analyzer (Cont'd)

Automatic Parametrization: the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

Modular: an analyzer instance is built by selection of O-CAML modules from a collection each implementing an abstract domain

Precise: very few or no false alarm when adapted to an application domain ⟶ it is a VERIFIER!

The ASTRÉE Abstract Interpretor

Abstract Semantics

- Reachable states for the concrete trace operational semantics (with partial history)
- Volatile environment is specified by a *trusted* configuration file.

Requirements:

- Soundness: absolutely essential
- Precision: few or no false alarm[76] (full certification)
- Efficiency: rapid analyses and fixes during development

[76] Potential runtime error signaled by the analyzer due to overapproximation but impossible in any actual program run.

ASTRÉE's Architecture

C-preprocessor
↕
C99 parser
↕
Link editor
↕
Intermediate code generation and typing
↕
Constant propagation and simplification
↕
Local and global dependence analysis
↕
Abstract Interpreter

The Abstract Interpreter

Iterator

$\updownarrow$

Trace partitionning

$\updownarrow$

Memory model and aliases

$\updownarrow$

Reduced product of numerical abstract domains

$\updownarrow$ $\quad$ $\updownarrow$ $\quad$ $\updownarrow$ $\quad$ $\updownarrow$

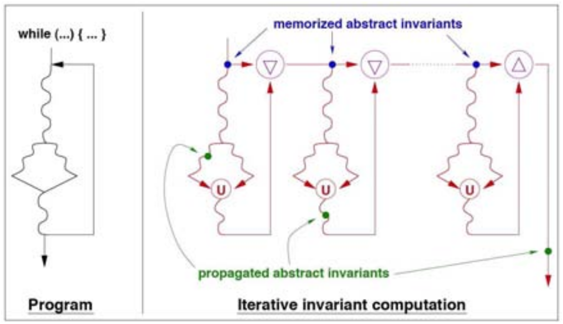
Intervals Octagons Decision trees . . .

$\updownarrow$

intervals

The Iterator

- Flow through all possible program executions, following the program syntactic structure
- Example: loops



Handling Functions

- No recursion $\Rightarrow$ functions can be handled without any abstraction
- we get a flow and context sensitive static analysis using an abstract stack for control/parameters/local variables (isomorphic to the concrete execution stack)

$\Rightarrow$ the analysis is extremely precise.

Handling Simple Variables

- In a given context, the abstraction of variable properties $\wp(\mathcal{X} \mapsto \mathcal{V})$ for fixed variables in $\mathcal{X}$ is given
  - everywhere by non-relational abstract domains (abstracting $\wp(\mathcal{V})$ by bitstring, set of values, simple congruence, interval, etc);
  - in chosen contexts, as a component of a relational abstract domains (octagons, etc).

  and subject to widening/narrowing and interdomain reductions.
- $\mathcal{X} \mapsto \alpha(\wp(\mathcal{V}))$ is represented by a balanced tree.

$\Rightarrow$ the parametrization of the analysis allows for a fine tuning of the cost/precision balance. .

Handling Arrays

The array T of size n can be handled

- as a collection of separate variables ($T[0]$, $T[1]$, $\ldots$, $T[n-1] \in \mathcal{X}$) handled individually as simple variables;
- as a single smashed variable $T \in \mathcal{X}$ which concrete value include *all possible values* of $T[0]$, $T[1]$, $\ldots$, and $T[n-1]$;

$\Rightarrow$ the parameterized analysis can be extremely precise when needed.

Handling Pointers

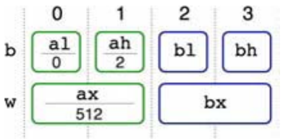
- No dynamic memory allocation $\Rightarrow$ the heap and aliases can be handled without any abstraction (using an abstract heap isomorphic to the concrete heap);
- A pointer is a basis plus an integer offset (abstracted separately by a set of bases $\subseteq \mathcal{X}$ and an auxiliary integer variable in $\mathcal{X}$ for the offset).

$\Rightarrow$ the analysis is extremely precise (but maybe for pointers to smashed array elements).

Memory Model

The union type, pointer arithmetics and pointer transtyping is handled by allowing aliasing at the byte level [44]:

```
union {
  struct { uint8  al,ah,bl,bh; } b;
  struct { uint16 ax,bx; } w;
} r;
r.w.ax = 0; r.b.ah = 2;
```



- A box (auxiliary variable) in $\mathcal{X}$ for each offset and each scalar type
- intersection semantics for overlapping boxes

Reference

[44] A. Miné. Field-Sensitive Value Analysis of Embedded C Programs with Union Types and Pointer Arithmetics. In *LCTES '2006*, pp. 54–63, June 2006, ACM Press.

Iteration Refinement: Loop Unrolling

Principle:

- Semantically equivalent to:

  `while (B) { C }` $\Longrightarrow$ `if (B) { C }; while (B) { C }`

- More precise in the abstract: less concrete execution paths are merged in the abstract.

Application:

- Isolate the initialization phase in a loop (e.g. first iteration).

Iteration Refinement: Trace Partitioning

Principle:

– Semantically equivalent to:

```
if (B) { C1 } else { C2 }; C3
⇓
if (B) { C1; C3 } else { C2; C3 };
```

– More precise in the abstract: concrete execution paths are merged later.

Application:

```
if (B)
  { X=0; Y=1; }
else
  { X=1; Y=0; }
R = 1 / (X-Y);
```

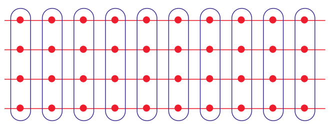
cannot result in a division by zero

Control Partitioning for Case Analysis

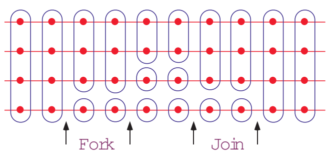
– Code Sample:

```
/* trace_partitionning.c */
void main() {
  float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
  float c[4] = {0.0, 2.0, 2.0, 0.0};
  float d[4] = {-20.0, -20.0, 0.0, 20.0};
  float x, r;
  int i = 0;

  ... found invariant −100 ≤ x ≤ 100 ...

  while ((i < 3) && (x >= t[i+1])) {
    i = i + 1;
  }
  r = (x - t[i]) * c[i] + d[i];
}
```

Control point partitionning:



Trace partitionning:

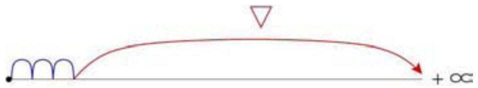


Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

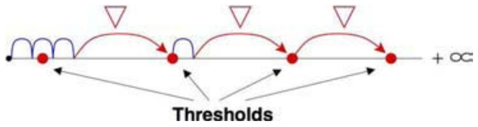
Convergence Accelerator: Widening

Principle:

– Brute-force widening:



– Widening with thresholds:



Examples:

– 1., 10., 100., 1000., etc. for floating-point variables;

– maximal values of data types;

– syntactic program constants, etc.

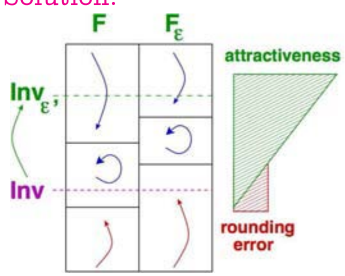
Fixpoint Stabilization for Floating-point

Problem:

– Mathematically, we look for an abstract invariant inv such that $F(\mathrm{inv}) \subseteq \mathrm{inv}$.

– Unfortunately, abstract computation uses floating-point and incurs rounding: maybe $F_\varepsilon(\mathrm{inv}) \not\subseteq \mathrm{inv}$!

Solution:



– Widen inv to $\mathrm{inv}_{\varepsilon'}$ with the hope to jump into a stable zone of F_ε.

– Works if F has some attractiveness property that fights against rounding errors (otherwise iteration goes on).

– ε' is an analysis parameter.

Examples of Abstractions in ASTRÉE

Examples of Abstractions (Cont'd)

– Set of complete traces $\xrightarrow{\alpha}$ set of reachable partial traces of a loop for 1, 2, …, n and $> n$ iterations [79] $\xrightarrow{\alpha}$ …

– Set of complete traces $\xrightarrow{\alpha}$ for each trace, a map of discrete time i in the trace to thevalue $\mathtt{X}_i$ of a variable $\mathtt{X}$ at that time i along that trace $\xrightarrow{\alpha}$ a map of dicrete time i in the traces to the maximum value $\bar{\mathtt{X}}_i$ of a variable $\mathtt{X}$ at that time i in all traces

[79] n is a parameter of ASTRÉE

Examples of Abstractions

– Set of complete traces $\xrightarrow{\alpha}$ reachable states $\xrightarrow{\alpha}$ [77] possible values of each variable in a given context (call stack & program point) $\xrightarrow{\alpha}$ [78] interval of possible values of each variable in each context;

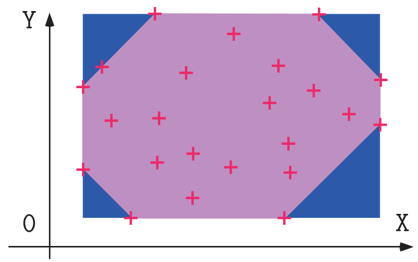
– … idem … $\xrightarrow{\alpha}$ simple congruence

– Set of complete traces $\xrightarrow{\alpha}$ reachable states $\xrightarrow{\alpha}$ set of vectors of possible values of all variables in a given context $\xrightarrow{\alpha}$ octagonal relations between pairs of variables in each context;

[77] cartesian abstraction

[78] interval abstraction

General-Purpose Abstract Domains: Intervals and Octagons



Intervals:
$$\begin{cases} 1 \le x \le 9 \\ 1 \le y \le 20 \end{cases}$$

Octagons [Min01]:
$$\begin{cases} 1 \le x \le 9 \\ x + y \le 77 \\ 1 \le y \le 20 \\ x - y \le 04 \end{cases}$$

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [CC77a, Min01, Min04a]

Floating-point linearization [Min04a, Min04b]

- Approximate arbitrary expressions in the form
  $[a_0, b_0] + \sum_k([a_k, b_k] \times V_k)$
- Example:
  `Z = X - (0.25 * X)` is linearized as
  $\mathtt{Z} = ([0.749\cdots, 0.750\cdots] \times \mathtt{x}) + (2.35\cdots 10^{-38} \times [-1, 1])$
- Allows simplification even in the interval domain
  if $\mathtt{X} \in$ [-1,1], we get $|\mathtt{Z}| \leq 0.750\cdots$ instead of $|\mathtt{Z}| \leq 1.25\cdots$
- Allows using a relational abstract domain (octagons)
- Example of good compromise between cost and precision

Symbolic abstract domain [Min04a, Min04b]

- Interval analysis: if $x \in [a, b]$ and $y \in [c, d]$ then $x - y \in [a - d, b - c]$ so if $x \in [0, 100]$ then $x - x \in [-100, 100]$!!!
- The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;
- Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

```
% cat -n x-x.c
     1  void main () { int X, Y;
     2          __ASTREE_known_fact(((0 <= X) && (X <= 100)));
     3          Y = (X - X);
     4          __ASTREE_log_vars((Y));
     5  }
```

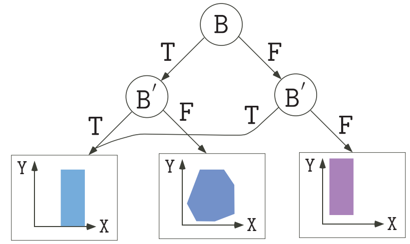
```
astree -exec-fn main -no-relational x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in [-100, 100]>
```

```
astree -exec-fn main x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in {0}> <symbolic: Y = (X -i X)>
```


Boolean Relations for Boolean Control

- Code Sample:

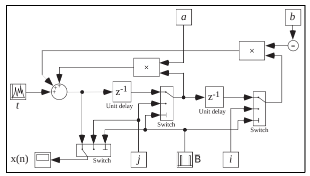
```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



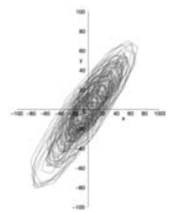
The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Ellipsoid Abstract Domain for Filters

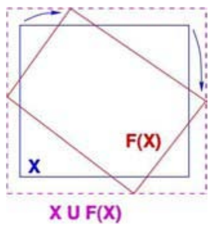
2^d Order Digital Filter:



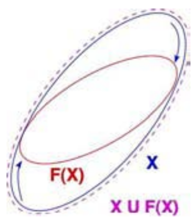
- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

Filter Example [Fer04]

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
            + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```


Arithmetic-Geometric Progressions (Example 1)

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }          ← potential overflow!
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree -exec-fn main -config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```


Arithmetic-geometric progressions [80] [Fer05]

– Abstract domain: $(\mathbb{R}^+)^5$

– Concretization:

$\gamma \in (\mathbb{R}^+)^5 \longmapsto \wp(\mathbb{N} \mapsto \mathbb{R})$

$$\gamma(M, a, b, a', b') =$$

$$\{f \mid \forall k \in \mathbb{N} : |f(k)| \leq \left(\boldsymbol{\lambda} x \cdot ax + b \circ (\boldsymbol{\lambda} x \cdot a'x + b')^k\right)(M)\}$$

i.e. any function bounded by the arithmetic-geometric progression.

80 here in $\mathbb{R}$

(Automatic) Parameterization

– All abstract domains of ASTRÉE are parameterized, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, . . . ;

– End-users can either parameterize by hand (analyzer options, directives in the code), or

– choose the automatic parameterization (default options, directives for pattern-matched predefined program schemata).

Example of Abstract Domain in ASTRÉE: The Arithmetic-Geometric Progression Abstract Domain

Reference

[45] J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.

Arithmetic-Geometric Progressions: Motivating Example

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P =  (P - ((((2.0 * P) - A) - B)
            * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1
+ 1.19209290217e-07)^clock
- 5.87747175411e-39 /
1.19209290217e-07 <= 23.0393526881
```


Objective

– In automatically generated programs using floating point arithmetics, some computations may diverge because of rounding errors.

– To prove the absence of floating point number overflows, we use non polynomial constraints:

  - we bound rounding errors at each loop iteration by a linear combination of the loop inputs;
  - we get bounds on the values depending exponentially on the program execution time.

– The abstract domain is both precise (no false alarm) and efficient (linear in memory / $n\ln(n)$ in time).

Running example (in $\mathbb{R}$)

1 : $X := 0; k := 0;$

2 : while $(k < 1000)$ {

3 :   if (?) $\{X \in [-10; 10]\};$

4 : $X := X/3;$

5 : $X := 3 \times X;$

6 : $k := k + 1;$

7 : }

Interval analysis: first loop iteration

$1: \ X := 0; k := 0;$

$\qquad\qquad X = 0$

$2: \ \texttt{while}\ (k < 1000)\ \{$

$\qquad\qquad X = 0$

$3: \quad \texttt{if}\ (?)\ \{X \in [-10; 10]\};$

$\qquad\qquad |X| \leq 10$

$4: \quad X := X/3;$

$\qquad\qquad |X| \leq \frac{10}{3}$

$5: \quad X := 3 \times X;$

$\qquad\qquad |X| \leq 10$

$6: \quad k := k + 1;$

$7: \quad \}$

Interval analysis: Invariant

$1: \ X := 0; k := 0;$

$\qquad\qquad X = 0$

$2: \ \texttt{while}\ (k < 1000)\ \{$

$\qquad\qquad |X| \leq 10$

$3: \quad \texttt{if}\ (?)\ \{X \in [-10; 10]\};$

$\qquad\qquad |X| \leq 10$

$4: \quad X := X/3;$

$\qquad\qquad |X| \leq \frac{10}{3}$

$5: \quad X := 3 \times X;$

$\qquad\qquad |X| \leq 10$

$6: \quad k := k + 1;$

$7: \quad \}$

$\qquad\qquad |X| \leq 10$

Including rounding errors [Miné–ESOP'04]

$1: \ X := 0; k := 0;$

$2: \ \texttt{while}\ (k < 1000)\ \{$

$3: \quad \texttt{if}\ (?)\ \{X \in [-10; 10]\};$

$4: \quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$

$5: \quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$

$6: \quad k := k + 1;$

$7: \quad \}$

The constants ε_1, ε_2, ε_3, and ε_4 (≥ 0) are computed by other domains.

Interval analysis

Let $M \geq 0$ be a bound:

$1: \ X := 0; k := 0;$

$\qquad\qquad X = 0$

$2: \ \texttt{while}\ (k < 1000)\ \{$

$\qquad\qquad |X| \leq M$

$3: \quad \texttt{if}\ (?)\ \{X \in [-10; 10]\};$

$\qquad\qquad |X| \leq max(M, 10)$

$4: \quad X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$

$\qquad\qquad |X| \leq (\varepsilon_1 + \frac{1}{3}) \times max(M, 10) + \varepsilon_2$

$5: \quad X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$

$\qquad\qquad |X| \leq (1 + a) \times max(M, 10) + b$

$6: \quad k := k + 1;$

$7: \quad \}$

with $a = 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3$ and $b = \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4$.

Ari.-geo. analysis: first iteration

1 : $X := 0; k := 0;$

$X = 0,\ k = 0$

2 : while $(k < 1000)$ {

$X = 0$

3 : if (?) $\{X \in [-10; 10]\};$

$|X| \leq 10$

4 : $X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$

$|X| \leq \left[v \mapsto \left(\frac{1}{3} + \varepsilon_1\right) \times v + \varepsilon_2\right](10)$

5 : $X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$

$|X| \leq f^{(1)}(10)$

6 : $k := k + 1;$

$|X| \leq f^{(k)}(10),\ k = 1$

7 : }

with $f = \left[v \mapsto \left(1 + 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3\right) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4\right]$.

Ari.-geo. analysis: Invariant

1 : $X := 0; k := 0;$

$X = 0,\ k = 0$

2 : while $(k < 1000)$ {

$|X| \leq f^{(k)}(10)$

3 : if (?) $\{X \in [-10; 10]\};$

$|X| \leq f^{(k)}(10)$

4 : $X := X/3 + [-\varepsilon_1; \varepsilon_1].X + [-\varepsilon_2; \varepsilon_2];$

$|X| \leq \left(\frac{1}{3} + \varepsilon_1\right) \times \left(f^{(k)}(10)\right) + \varepsilon_2$

5 : $X := 3 \times X + [-\varepsilon_3; \varepsilon_3].X + [-\varepsilon_4; \varepsilon_4];$

$|X| \leq f\left(f^{(k)}(10)\right)$

6 : $k := k + 1;$

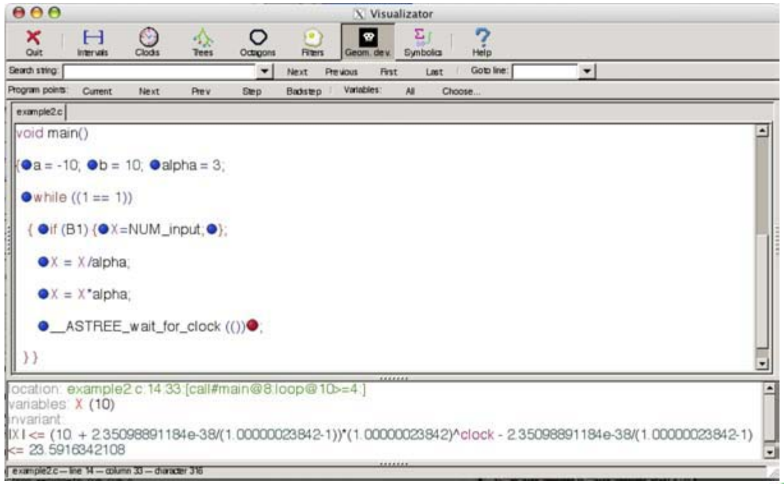
$|X| \leq f^{(k)}(10)$

7 : }

$|X| \leq f^{(1000)}(10)$

with $f = \left[v \mapsto \left(1 + 3 \times \varepsilon_1 + \frac{\varepsilon_3}{3} + \varepsilon_1 \times \varepsilon_3\right) \times v + \varepsilon_2 \times (3 + \varepsilon_3) + \varepsilon_4\right]$.

Analysis session



Arithmetic-geometric progressions (in $\mathbb{R}$)

An arithmetic-geometric progression is a 5-tuple in $\left(\mathbb{R}^+\right)^5$.

An arithmetic-geometric progression denotes a function in $\mathbb{N} \to \mathbb{R}^+$:

$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) \triangleq [v \mapsto a \times v + b]\left(\left[v \mapsto a' \times v + b'\right]^{(k)}(M)\right)$$

Thus,

- k is the loop counter;
- M is an initial value;
- $[v \mapsto a \times v + b]$ describes the current iteration;
- $[v \mapsto a' \times v + b']^{(k)}$ describes the first k iterations.

A concretization $\gamma_{\mathbb{R}}$ maps each element $d \in \left(\mathbb{R}^+\right)^5$ to a set $\gamma_{\mathbb{R}}(d) \subseteq (\mathbb{N} \to \mathbb{R}^+)$ defined as:

$$\{f \mid \forall k \in \mathbb{N},\ |f(k)| \leq \beta_{\mathbb{R}}(d)(k)\}$$

Monotonicity

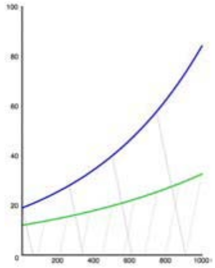
Let $d = (M, a, b, a', b')$ and $d = (M, a, b, a', b')$ be two arithmetic-geometric progressions.

If:

- $M \leq M$,
- $a \leq a$, $a' \leq a'$,
- $b \leq b$, $b' \leq b'$.

Then:

$$\forall k \in \mathbb{N},\ \beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(d)(k).$$



Disjunction

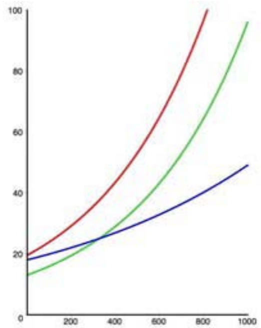
Let $d = (M, a, b, a', b')$ and $d = (M, a, b, a', b')$ be two arithmetic-geometric progressions.

We define:

$$d \sqcup_{\mathbb{R}} d \triangleq (M, a, b, a', b')$$

where:

- $M \triangleq max(M, M)$,
- $a \triangleq max(a, a)$, $a' \triangleq max(a', a')$,
- $b \triangleq max(b, b)$, $b' \triangleq max(b', b')$,



For any $k \in \mathbb{N}$, $\beta_{\mathbb{R}}(d \sqcup_{\mathbb{R}} d)(k) \geq max(\beta_{\mathbb{R}}(d)(k), \beta_{\mathbb{R}}(d)(k))$.

Conjunction

Let d and d be two arithmetic-geometric progressions.

1. If d and d are comparable (component-wise), we take the smaller one:

$$d \sqcap_{\mathbb{R}} d \triangleq Inf_{\leq}\{d; d\}.$$

2. Otherwise, we use a parametric strategy:

$$d \sqcap_{\mathbb{R}} d \in \{d; d\}.$$

For any $k \in \mathbb{N}$, $\beta_{\mathbb{R}}(d \sqcap_{\mathbb{R}} d)(k) \geq min(\beta_{\mathbb{R}}(d)(k), \beta_{\mathbb{R}}(d)(k))$.

Assignment

We have:

$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times (M + b' \times k) + b \quad \text{when } a' = 1$$
$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times \left((a')^k \times \left(M - \frac{b'}{1-a'}\right) + \frac{b'}{1-a'}\right) + b \quad \text{when } a' \neq 1.$$

Thus:

1. for any $a, a', M, b, b', \lambda \in \mathbb{R}^+$,

$$\lambda \times (\beta_{\mathbb{R}}(M, a, b, a', b')(k)) = \beta_{\mathbb{R}}(\lambda \times M, a, \lambda \times b, a', \lambda \times b')(k);$$

2. for any $a, a', M, b, b', M, b, b' \in \mathbb{R}^+$, for any $k \in \mathbb{N}$,

$$\beta_{\mathbb{R}}(M, a, b, a', b')(k) + \beta_{\mathbb{R}}(M, a, b, a', b)(k) = \beta_{\mathbb{R}}(M + M, a, b + b, a', b' + b')(k).$$

Projection I/II

$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times (M + b' \times k) + b$ when $a' = 1$

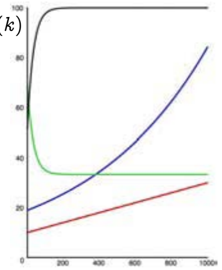
$\beta_{\mathbb{R}}(M, a, b, a', b')(k) = a \times \left((a')^k \times \left(M - \frac{b'}{1-a'}\right) + \frac{b'}{1-a'}\right) + b$ when $a' \neq 1$.

Thus, for any $d \in (\mathbb{R}^+)^5$, the function $[k \mapsto \beta_{\mathbb{R}}(d)(k)]$ is:

- either monotonic,
- or anti-monotonic.

$$\begin{cases} a' > 1, \\ a' = 1, \\ a' < 1 \text{ and } M < \frac{b'}{1-a'}, \\ a' < 1 \text{ and } M > \frac{b'}{1-a'}. \end{cases}$$

$\beta_{\mathbb{R}}(d)(k)$



k

Projection II/II

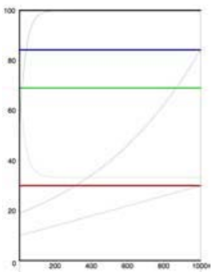
Let $d \in (\mathbb{R}^+)^5$ and $k_{max} \in \mathbb{N}$.

$bound(d, k_{max}) \stackrel{\Delta}{=} max(\beta_{\mathbb{R}}(d)(0), \beta_{\mathbb{R}}(d)(k_{max}))$

For any $k \in \mathbb{N}$ such that $0 \leqslant k \leqslant k_{max}$

$\beta(d)(k) \leq bound(d, k_{max})$.

$\beta_{\mathbb{R}}(d)(k)$



k

0 k_{max}

Incrementing the loop counter

We integrate the current iteration into the first k iterations:

- the first $k+1$ iterations are chosen as the worst case among the first k iterations and the current iteration;
- the current iteration is reset.

Thus:

$$next_{\mathbb{R}}(M, a, b, a', b') \stackrel{\Delta}{=} (M, 1, 0, max(a, a'), max(b, b')).$$

For any $k \in \mathbb{N}, d \in (\mathbb{R}^+)^5$, $\beta_{\mathbb{R}}(d)(k) \leq \beta_{\mathbb{R}}(next_{\mathbb{R}}(d))(k+1)$.

About floating point numbers

Floating point numbers occur:

1. in the concrete semantics:
   Floating point expressions are translated into real expressions with interval coefficients [Miné—ESOP'04].
   So the abstract domains, can handle real numbers.
2. in the abstract domain implementation:
   For efficiency purpose, each real primitive is implemented in floating point arithmetics: each real is safely approximated by an interval with floating point number bounds.

Using Astrée

Example application

– Primary flight control software of the Airbus A340 family/A380 fly-by-wire system


– C program, automatically generated from a proprietary high-level specification (à la Simulink/Scade)

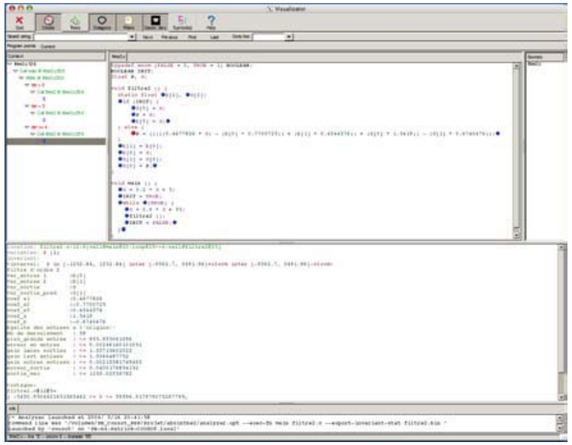
– A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays, now $\times$ 2

– A380: $\times$ 3/7

Digital Fly-by-Wire Avionics [81]

[81] The electrical flight control system is placed between the pilot's controls (sidesticks, rudder pedals) and the control surfaces of the aircraft, whose movement they control and monitor.

Example of Analysis Session



Benchmarks (Airbus A340 Primary Flight Control Software)

– V1 [82], 132,000 lines, 75,000 LOCs after preprocessing
– Comparative results (commercial software):
  4,200 (false?) alarms, 3.5 days;
– Our results:
  0 alarms,
  40mn on 2.8 GHz PC, 300 Megabytes
  $\longrightarrow$ A world première in Nov. 2003!

82 "Flight Control and Guidance Unit" (FCGU) running on the "Flight Control Primary Computers" (FCPC). The three primary computers (FCPC) and two secondary computers (FCSC) which form the A340 and A330 electrical flight control system are placed between the pilot's controls (sidesticks, rudder pedals) and the control surfaces of the aircraft, whose movement they control and monitor.

The main loop invariant for the A340 V1

A textual file over 4.5 Mb with
– 6,900 boolean interval assertions ($x \in [0; 1]$)
– 9,600 interval assertions ($x \in [a; b]$)
– 25,400 clock assertions ($x + \mathrm{clk} \in [a; b] \wedge x - \mathrm{clk} \in [a; b]$)
– 19,100 additive octagonal assertions ($a \leq x + y \leq b$)
– 19,200 subtractive octagonal assertions ($a \leq x - y \leq b$)
– 100 decision trees
– 60 ellipse invariants, etc ...

involving over 16,000 floating point constants (only 550 appearing in the program text) $\times$ 75,000 LOCs.

(Airbus A380 Primary Flight Control Software)

– Now at 1,000,000 lines!
– 0 alarms (Nov. 2004), after some additional parametrization and simple abstract domains developments
  34h,
  8 Gigabyte
  $\longrightarrow$ A world grand première!

Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:
– Abstract transformers (not best possible) $\longrightarrow$ improve algorithm;
– Automatized parametrization (e.g. variable packing) $\longrightarrow$ improve pattern-matched program schemata;
– Iteration strategy for fixpoints $\longrightarrow$ fix widening [83];
– Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract $\longrightarrow$ add a new abstract domain to the reduced product (e.g. filters).

83 This can be very hard since at the limit only a precise infinite iteration might be able to compute the proper abstract invariant. In that case, it might be better to design a more refined abstract domain.

7. Conclusion

Conclusion

– The behaviors of computer systems are too large and complex for enumeration (state/combinatorial explosion);

– Abstraction is therefore necessary to reason or compute behaviors of computer systems;

– Making explicit the rôle of abstract interpretation in formal methods might be fruitful;

– In particular to apply formal methods to complex industrial applications [46].

References

[46] D. Delmas and J. Souyris. ASTRÉE: *from Research to Industry*. Proc. 14th Int. Symp. SAS '07, G. Filé and H. Riis-Nielson (eds), 22–24 Aug. 2007, Kongens Lyngby, DK, LNCS 4634, pp. 437–451, Springer.

THE END, THANK YOU

8. Bibliography

[AGM93] G. Amato, F. Giannotti, and G. Mainetto. Data sharing analysis for a database programming language via abstract interpretation. In R. Agrawal, S. Baker, and D.A.Bell, editors, *Proc. 19th Int. Conf. on Very Large Data Bases*, pages 405–415, Dublin, IE, 24–27 Aug. 1993. MORGANKAUFMANN.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, 1985.

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proc. ACM SIGPLAN '2003 Conf. PLDI*, pages 196–207, San Diego, CA, US, 7–14 June 2003. ACM Press.

[BPC01] J. Bailey, A. Poulovassilis, and C. Courtenage. Optimising active database rules by partial evaluation and abstract interpretation. In *Proc. 8th Int. Work. on Database Programming Languages*, LNCS 2397, pages 300–317, Frascati, IT, 8–10 Sep. 2001. Springer.

[BS97] V. Benzaken and X. Schaefer. Static integrity constraint management in object-oriented database programming languages via predicate transformers. In M. Aksit and S. Matsuoka, editors, *Proc. 11th European Conf. on Object-Oriented Programming, ECOOP '97*, LNCS 1241. Springer, Jyväskylä, FI, 9–13 June 1997.

[CC77a] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[CC77b] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E.J. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*, pages 237–277. North-Holland, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[CC82] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs, invited chapter. In D. Néel, editor, *Tools & Notions for Program Construction*, pages 43–119. Cambridge U. Press, 1982.

[CC87] P. Cousot and R. Cousot. Sometime = always + recursion ≡ always: on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informat.*, 24:1–31, 1987.

[CC92a] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4th Int. Symp. PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer, 1992.

[CC92b] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *19th POPL*, pages 83–94, Albuquerque, NM, US, 1992. ACM Press.

[CC95] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. 7th FPCA*, pages 170–181, La Jolla, CA, US, 25–28 June 1995. ACM Press.

[CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *27th POPL*, pages 12–25, Boston, MA, US, Jan. 2000. ACM Press.

[CC02a] P. Cousot and R. Cousot. Modular static program analysis, invited paper. In R.N. Horspool, editor, *Proc. 11th Int. Conf. CC '2002*, pages 159–178, Grenoble, FR, 6–14 Apr. 2002. LNCS 2304, Springer.

[CC02b] P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *29th POPL*, pages 178–190, Portland, OR, US, Jan. 2002. ACM Press.

[CC03] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290(1):531–544, Jan. 2003.

[CC04] P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *31st POPL*, pages 173–185, Venice, IT, 14–16 Jan. 2004. ACM Press.

[CCF+] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE Static Analyzer. http://www.astree.ens.fr/.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.

[Cou78] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, FR, 21 Mar. 1978.

[Cou97] P. Cousot. Types as abstract interpretations, invited paper. In *24th POPL*, pages 316–331, Paris, FR, Jan. 1997. ACM Press.

[Cou00] P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 1–25. Springer, 26–29 Jul. 2000.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

[Cou03] P. Cousot. Verification by abstract interpretation, invited chapter. In N. Dershowitz, editor, *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, pages 243–268. LNCS 2772, Springer, Taormina, IT, 29 June – 4 Jul. 2003.

[Dan07] V. Danos. Abstract views on biological signaling. In *Mathematical Foundations of Programming Semantics, 23rd Annual Conf. (MFPS XXIII)*, 2007.

[DS07] D. Delmas and J. Souyris. Astrée: from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 437–451. Springer, 22–24 Aug. 2007.

[Fer04] J. Feret. Static analysis of digital filters. In D. Schmidt, editor, *Proc. 30th ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 33–48. Springer, Mar. 27 – Apr. 4, 2004.

[Fer05] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proc. 6th Int. Conf. VMCAI 2005*, pages 42–58, Paris, FR, 17–19 Jan. 2005. LNCS 3385, Springer.

[FHL+01] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In T.A. Henzinger and C.M. Kirsch, editors, *Proc. 1st Int. Work. EMSOFT '2001*, volume 2211 of *LNCS*, pages 469–485. Springer, 2001.

[Flo67] R.W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Proc. Symposium in Applied Mathematics*, volume 19, pages 19–32. AMS, 1967.

[GM04] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *31st POPL*, pages 186–197, Venice, IT, 2004. ACM Press.

[GMP02] É. Goubault, M. Martel, and S. Putot. Asserting the precision of floating-point computations: a simple abstract interpreter. In D. Le Métayer, editor, *Proc. 11th ESOP '2002*, Grenoble, FR, LNCS 2305, pages 209–212. Springer, 8–12 Apr. 2002.

[GR06] R. Giacobazzi and F. Ranzato. Incompleteness of states w.r.t traces in model checking. *Inform. and Comput.*, 204(3):376–407, Mar. 2006.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[JP06] Ph. Jorrand and S. Perdrix. Towards a quantum calculus. In *Proc. 4th Int. Work. on Quantum Programming Languages, ENTCS*, 2006.

[Min01] A. Miné. A new numerical abstract domain based on difference-bound matrices. In 0. Danvy and A. Filinski, editors, *Proc. 2nd Symp. PADO '2001*, Århus, DK, 21–23 May 2001, LNCS 2053, pages 155–172. Springer, 2001.

[Min04a] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. Schmidt, editor, *Proc. 30th ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 3–17. Springer, Mar. 27 – Apr. 4, 2004.

[Min04b] A. Miné. *Weakly Relational Numerical Abstract Domains.* Thèse de doctorat en informatique, École polytechnique, Palaiseau, FR, 6 Dec. 2004.

[Min06a] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *Proc. LCTES '2006*, pages 54–63. ACM Press, June 2006.

[Min06b] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[MR05] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In M. Sagiv, editor, *Proc. 14th ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2—-10, 2005.

[PCJD07] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. Semantics-based approach to malware detection. In *34th POPL*, pages 238–252, Nice, France, 17–19 Jan. 2007. ACM Press.

[Per06] S. Perdrix. *Modèles formels du calcul quantique : ressources, machines abstraites et calcul par mesure.* PhD thesis, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, 2006.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, Providence, RI, Nov. 1977.

[RSD00] F. Randimbivololona, J. Souyris, and A. Deutsch. Improving avionics software verification cost-effectiveness: Abstract interpretation based technology contribution. In *Proceedings DASIA 2000 – DAta Systems In Aerospace*, Montreal, CA. ESA Publications, 22–26 May 2000.

[RT04] F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. 30th ESOP '04*, volume 2986 of *LNCS*, pages 18–32, Barcelona, ES, Mar. 29 – Apr. 2 2004. Springer.

[Sou04] J. Souyris. Industrial experience of abstract interpretation-based static analyzers. In P. Jacquart, editor, *Building the Information Society*, chapter 4, pages 393–400. Kluwer Acad. Pub., 2004.

[TSH+03] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. Abstract interpretation-based timing validation of hard real-time avionics software. In *Proc. Int. Conf. DSN 2003*, San Francisco, CA, US, pages 625–634. IEEE Comp. Soc. Press, 22–25 June 2003.