PolySpace TECHNOLOGIES

Abstract Interpretation

Applied to Static Run-Time Error Detection (SigAda 2002)

Dr. Chris Hote
PolySpace Technologies

Slides prepared by Dr. Alain Deutsch
(CTO and co-founder of PolySpace Technologies)
http://www.polyspace.com

hote@polyspace.com, deutsch@polyspace.com

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

Talk overview

- What is abstract interpretation (AI)?
- How does it work? Can we trust it?
- How AI helps in software development?
- Are there industrial tools implementing AI?
- Are there pragmatic industrial users of AI?

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

Data-Flow Analysis

- **Definition.** Data-flow analysis is a branch of computer science aiming at statically computing program properties
- Principle:
  - Translate program to equations over lattice
  - Solve equations by fixed-point iterations
- Widely used in modern compilers for code optimizations
- Pioneered by Kildall in 1973

- G. Kildall, A unified approach to global program optimization, *ACM Symposium On Principles of Programming Languages*, 194-206, 1973.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

Abstract Interpretation

- **Definition.** Abstract interpretation extends data-flow analysis by providing an additional theoretical framework for:
  - Mathematically justifying data-flow analyzers
  - Designing new data-flow analyses
  - Handling particular infinite sets of properties

- M. Sintzoff, Calculating properties of programs by valuations on specific models, p*roc. ACM Conf. on Proving Assertions about Programs*, Sigplan Notices, 7(1), 203-207, 1972.
- B. Wegbreit, Property extraction in well-founded property sets, *IEEE Transactions on Software Engineering*, 1(3) :270-285, 1975.
- P. & R. Cousot, Systematic design of program analysis frameworks, *proc. Principles of Programming Languages*, ACM Press, 1979.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Operational Semantics

Example of a simple flowchart language

- Simple flowchart programming language consisting of:
  - 32bits integer variables and integers
  - arithmetic operations
  - assignments
  - conditional and loops
- States are pairs consisting of:
  - an integer representing the current flowchart instruction to be executed
  - a vector of integers in an *n*-dimensional state
  - *n* is the number of variables in program *P*

- J.Van Leeuwen. *Handbook of theoretical computer science*. The MIT Press,1990.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Strongest global invariants

Definition

- *SGI(k)* is the set of all possible states that are at program point *k* and reachable in program *P*
- For the flowchart language it is a set of points in an *n* dimensional space
- Run-time errors triggered when *SGI(k)* intersects a forbidden zone

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Strongest global invariants
As fixed-points

- *SGI(k)* as the result of formal proof methods
- *SGI(k)* as the least fixed-point of a monotonic operator on the lattice of set of states
- *SGI(k)* as the solution of a system of equations whose unknowns are sets of states

- R. Floyd, Assigning meaning to programs. In *Mathematical Aspects of Computer Science*, Proc. of Symposia on Applied Mathematics, American Mathematical Society, 19-32, Providence, 1967.
- D. Park, Fixpoint induction and proofs of program properties, in *Machine Intelligence,* Edinburgh Univ. Press, 5 : 59-78, 1969.
- E. Clarke, Program invariants as fixedpoints, *Computing,* 21 : 273-294, 1979.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Strongest global invariants

The Floyd/Park/Clarke method

- Translate program P to a system:

  $X_1 = F_1(X_1,\ldots,X_m)$

  $X_2 = F_2(X_1,\ldots,X_m)$

  ...

  $X_m = F_m(X_1,\ldots,X_m)$

- Compute the least solution $(V_1, \ldots, V_m)$
  - *using Kleene ascending sequence:*
    - $X_{i,0} = \phi$
    - $X_{i,k+1} = F_i(X_{1,k},\ldots,X_{m,k})$
- Let $SGI(p)=X_p$

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Strongest global invariants

Example computation

Program

```
0   k=ioread_i32();
1   I=2;
2   J=K+5;
3   while (I<10) {
4       I=I+1;
5       J=J+3;
6   }
7
8   … /(I-J)
```

Equations

$X_0=\{(0,0,k)|\ \exists k\in[-2^{31},2^{31}-1]\}$

$X_1=\{(2,j,k)|\ \exists(i,j,k)\in X_0\}$

$X_2=\{(i,k+5,k)|\exists(i,j,k)\in X_1\}$

$X_3=X_2 \cup X_6$

$X_4=\{(i+1,j,k)|\exists(i,j,k)\in X_3,\ i<10\}$

$X_5=\{(i,j+3,k)|\exists(i,j,k)\in X_4\}$

$X_6=X_5$

$X_7=\{(i,j,k)|\exists(i,j,k)\in X_3,\ i\geq 10\}$

$X_8=\{(i,j,k)|\exists(i,j,k)\in X_7,\ i-j\neq 0\}$

$X_{error}=\{(i,j,k)|\exists(i,j,k)\in X_7,\ i-j=0\}$

Solution (least fixed-point)

$X_0=\{(0,0,k)|\ \exists\ k\in[-2^{31},2^{31}-1]\}$

$X_1=\{(2,0,k)|\ \exists\ k\in[-2^{31},2^{31}-1]\}$

$X_2=\{(2,k+5,k)|\ \exists\ k\in[-2^{31},2^{31}-1]\}$

$X_3=\{(i,j,k)|\ \exists\ k\in[-2^{31},2^{31}-1], i\in[2,10],\ j=k+3i-1\}$

$X_4=\{(i,j,k)|\ \exists\ k\in[-2^{31},2^{31}-1], i\in[3,10],\ j=k+3i-4\}$

$X_5=\{(i,j,k)|\ \exists\ k\in[-2^{31},2^{31}-1], i\in[3,10],\ j=k+3i-1\}$

$X_6=X_5$

$X_7=\{(10,j,k)|\ \exists\ k\in[-2^{31},2^{31}-1],\ j=k+29\}$

$X_8=\{(10,j,k)|\ \exists\ k\in[-2^{31},2^{31}-1],\ j=k+29,\ j\neq 10\}$

$X_{error}=\{(10,10,-19)\}$

Is there a potential divide-by-zero?

Zero divide will occur at point 8 when K = -19

Note that constant -19 does not appear in the source

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Strongest global invariants

Intractability

- For general purpose languages, *SGI(k)* is non-computable:
  - halting problem (deciding if a program stops) is reducible to checking that *SGI(k)*=ϕ
  - halting problem is undecidable
  - thus computing *SGI(k)*=ϕ is undecidable

- A. Turing. Computability and λ-definability. *J. Symbolic Logic*, 2 :153-163, 1937.
- C. Hoare and D. Allison. Incomputability, *ACM Computing Surveys*, 4(3),1972.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Abstract Interpretation

Approximate Semantics

- Replace the system of exact equations by its image by a closure operator ρ:
  - monotonic: $x \subseteq y \Rightarrow \rho(x) \subseteq \rho(y)$
  - extensive: $x \subseteq \rho(x)$
  - idempotent: $\rho(\rho(x)) = \rho(x)$
- Solve this approximate system in the abstract lattice $\rho(L)$
- Solution necessarily a superset of solution of exact system

- B. Wegbreit, Property extraction in well-founded property sets, *IEEE Transactions on Software Engineering*, 1(3) :270-285, 1975.
- P. & R. Cousot, Systematic Design of Program Analysis Frameworks, *proc. Principles of Programming Languages*, ACM Press, 1979.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

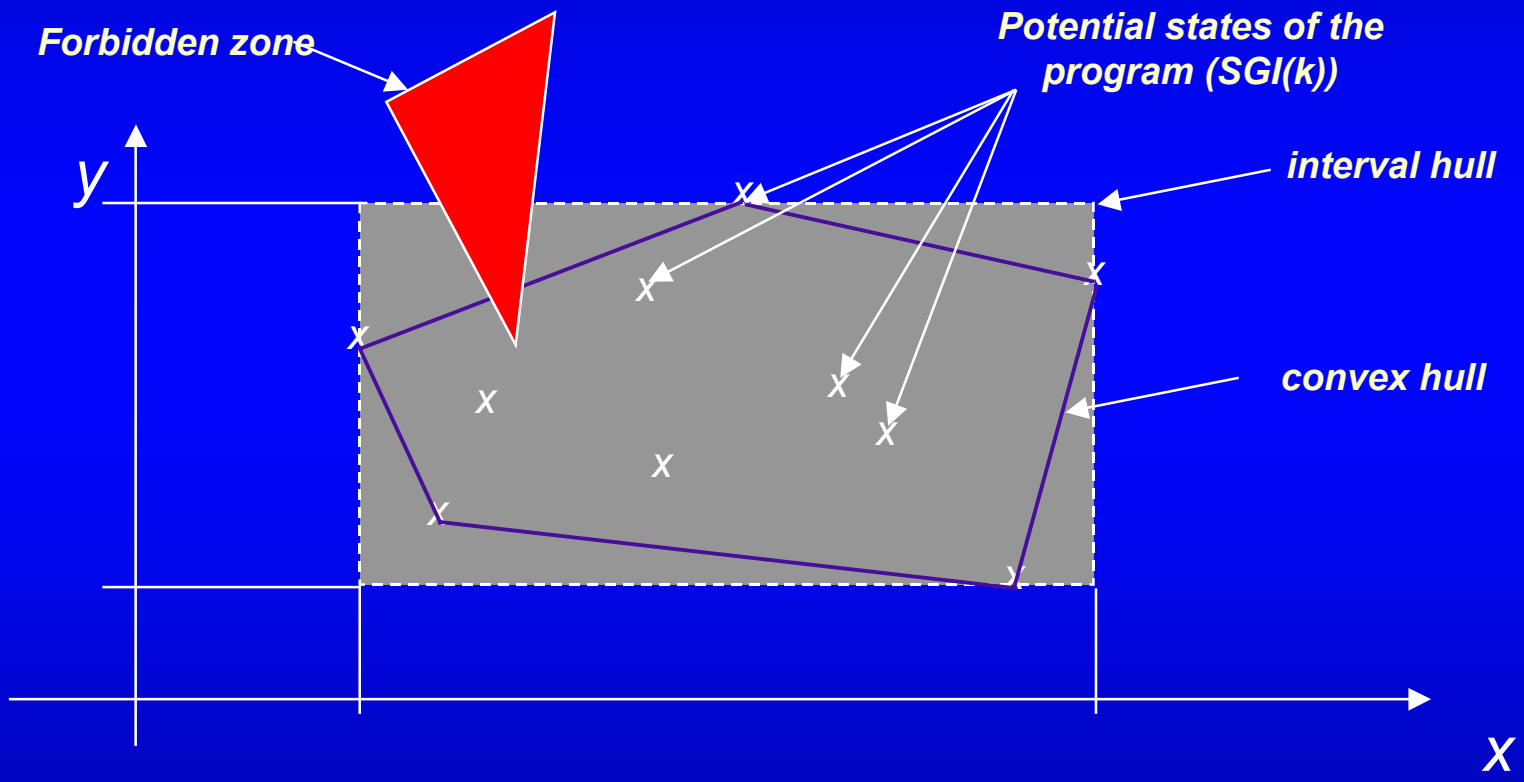
Non-numerical lattices

- Real programming language set new challenges:
  - Functions/subprograms
  - Pointers
  - Data structures (arrays, records, ...)
  - Dynamic allocation
  - Tasking
- Corresponding abstract lattices must be defined
- Example: lattice of unitary-prefix monomial relations to represent complex pointer aliasing patterns such as {(*(*(X+i)+4), *(Y+j)) | i≥2j+1}

- J.H. Chow, W.L. Harrison, Compile-Time Analysis of Parallel Programs that Share Memory. in *Proc. Principles of Programming Languages*, ACM Press, Albuqurque, 130-141, 1992.
- A. Deutsch, Interprocedural May-Alias Analysis for Pointers: Beyond k-limiting, in *Proc. Programming Language Design and Implementation*, ACM Press, Orlando, 1994.
- S. Eilenberg. *Automata, Languages, and Machines*. Academic Press, New York, 1974.

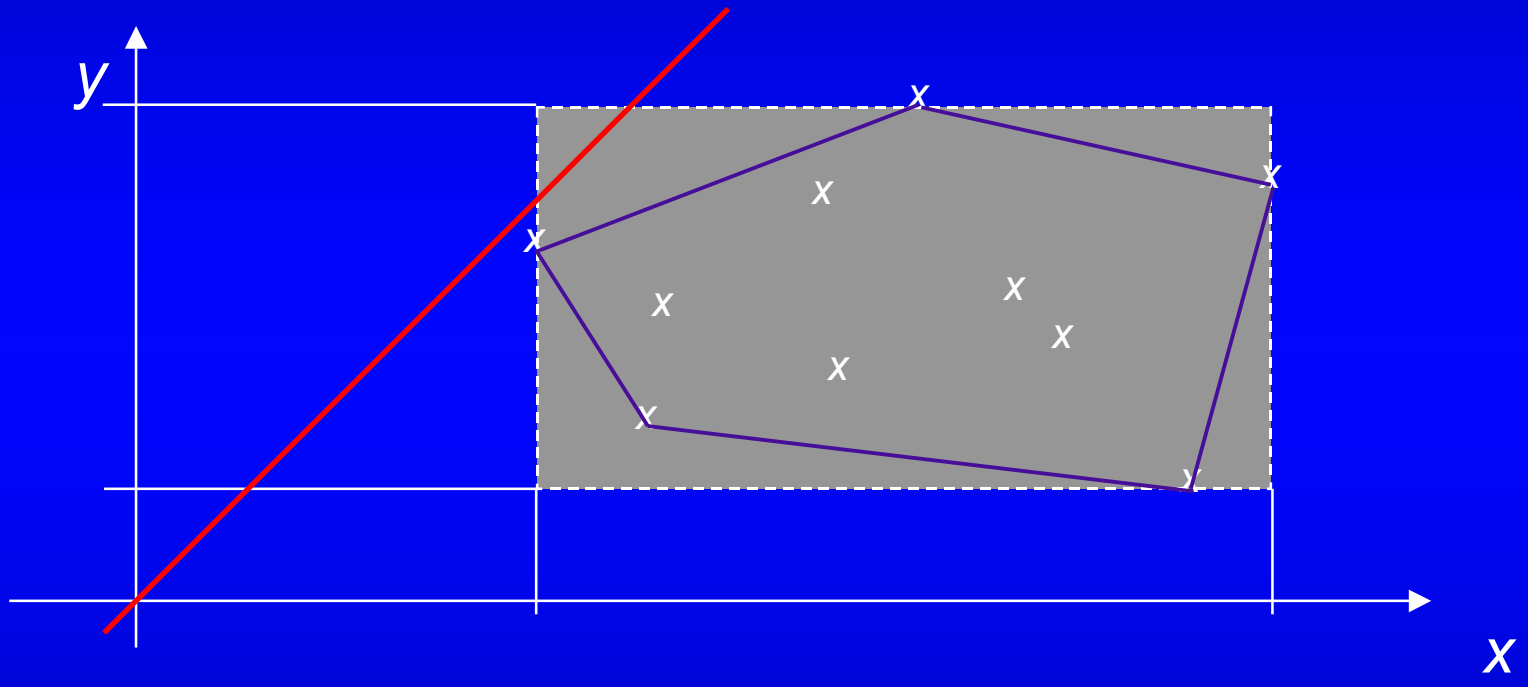
This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Abstract Semantics



This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

No Error

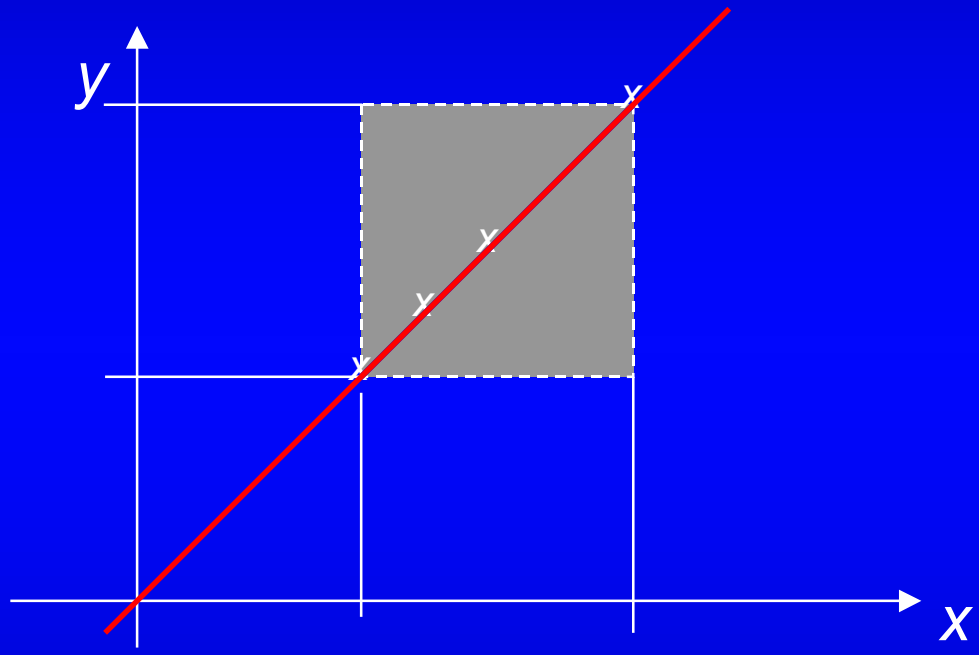


Check : `a = x /(x-y);`

The failure state is outside the state space of the program.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Certain Error

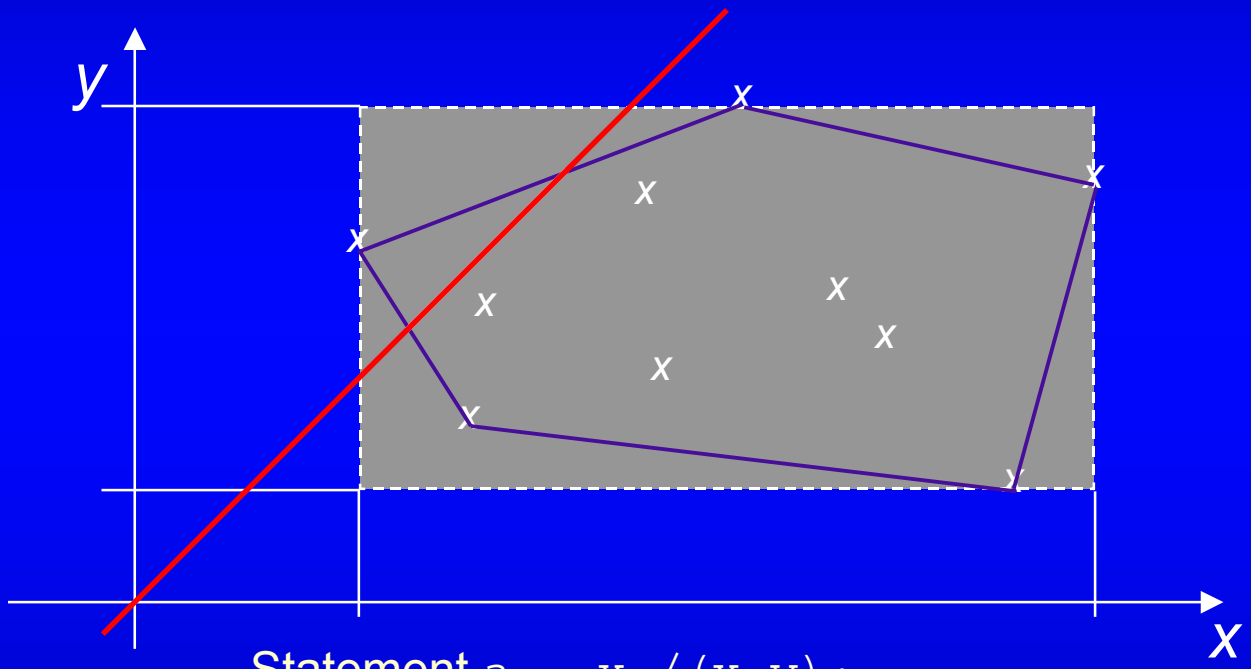


Check : `a = x /(x-y);`

The state space of the program is completely included in the failure state.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Potential Error



Statement `a = x /(x-y);`

Check the correctness condition x≠y

The intersection between the failure state and the state space of the program is not null.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Key properties

- Exhaustive Diagnostic:
  - A real error will never be signaled as green
  - An instruction always correct will never be signaled as red
  - Exhaustive analysis of run-time errors achieved by examining only orange and red checks
- No need to give test cases as input
- Diagnostics given hold for any future execution

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Why being interested in AI solution?

The NIST study

May'02 NIST Report (http://www.nist.gov/director/prog-ofc/report02-3.pdf):
Software failures cost up to $60bn to the US economy
(30%+ is supported by software manufacturers mainly spent in bug tracking before or after releasing products)

According to the NIST report (Executive Summary), an improved software testing infrastructure should:

- *Remove more bugs before software is released;*
- *Detect bugs earlier in the software development process;*
- *Locate the source of bugs faster and with more precision.*

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

How AI helps in verification & validation?

- Key properties ensure that AI can shorten and/or replace:
  - debugging, by finding run-time errors automatically
  - robustness testing by pinpointing exhaustively sources of run-time errors
  - code reviews and documentation, by extracting control and data flow information

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Are there industrial tools based on AI?

- Yes, PolySpace Verifier for Ada and ANSI C are industrial tools commercially available since 1999 (Ada) and 2000 (C)
- They address two essential needs:
  - **Static verification**: statically predict and precisely locate specific classes of run-time errors and sources of non determinism;
  - **Semantic browsing**: statically compute data and control flow to ease program understanding, verification or qualification.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

Technology

- But wait. The idea is from the 70's. If it is such a great idea, why hasn't it been done before ?
  1. Computers were not fast enough.
  2. Precise and scalable analyses were not available: many published methods are either too imprecise or too costly in our context.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace TECHNOLOGIES

PolySpace Verifier

Detected run-time errors (possibly causing non determinism, incorrect results or processor halt):

- De-referencing through null pointer
- Out-of-bounds pointers
- Out-of-bounds array access
- Read access to non-initialized data
- Access conflicts on shared data (multithreaded applications and/or interrupt routines)
- Invalid arithmetic operations: division by zero, …
- Overflow / underflow on integers and floating point numbers
- Unreachable (dead) code

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

PolySpace Verifier

Color-Coded Source code

Green
safe

Red
fatal error

Grey
dead

Orange
warning

```
62        static void Pointer_Arithmetic ()
63        {
64          int tab[100];
65          int i, *p = tab;
66
67          for(i = 0; i < 100; i++, p++)
68            *p = 0;
69
70          if(random int() == 0)
71            *p = 5; /* Out of bounds */
72            ++i;    /* Unreachable (runtime error on previous line) */
73            }
74
75          i = random int();
76          if (random int()) *(p-i) = 10;
77
78          if (0<i && i<=100)
79          { p = p - i;
80            *p = 5;      /* Safe pointer access */
81          }
82        }
```


This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Are there pragmatic industry users yet?

- Yes. PolySpace Verifier is now in use in several industry sectors:
  - Space industry: Ariane 5 flight program
  - Avionics: half-million lines application
  - Railways: safety systems
  - Chemical plants: safety control system
  - Automotive: 200,000 loc diesel engine control

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

1997: First Industrial Application Of Abstract Interpretation For Verification

- with CNES, Aerospatiale
- ARIANE Flight Programs A502,...,A517
  - Mission critical :
    - Navigation
    - Guidance
    - pilot ...
  - 5 interacting parallel tasks
  - Over 70000 LOC of Ada

- *ARIANE 5 - The Software Reliability Verification Process*, Ph. Lacan, J.N. Monfort & L.V.Q. Ribal - Aerospatiale, France ; A. Deutsch & G. Gonthier - INRIA, France; Proceedings (European Space Agency SP-422) DASIA 98 - Data Systems In Aerospace, May 1998.

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Industrial Applications
Avionics

- End-user of the avionics industry
- Flight Management System (FMS) of about 500,000 lines (Ada)
- End-user "estimates the cost savings in the final phase of the project at between $150,000 and $250,000" as consequence of several serious errors uncovered by the PolySpace Verifier tool, including data races

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Industrial Applications

Chemical Industry: Triconex/Invensys

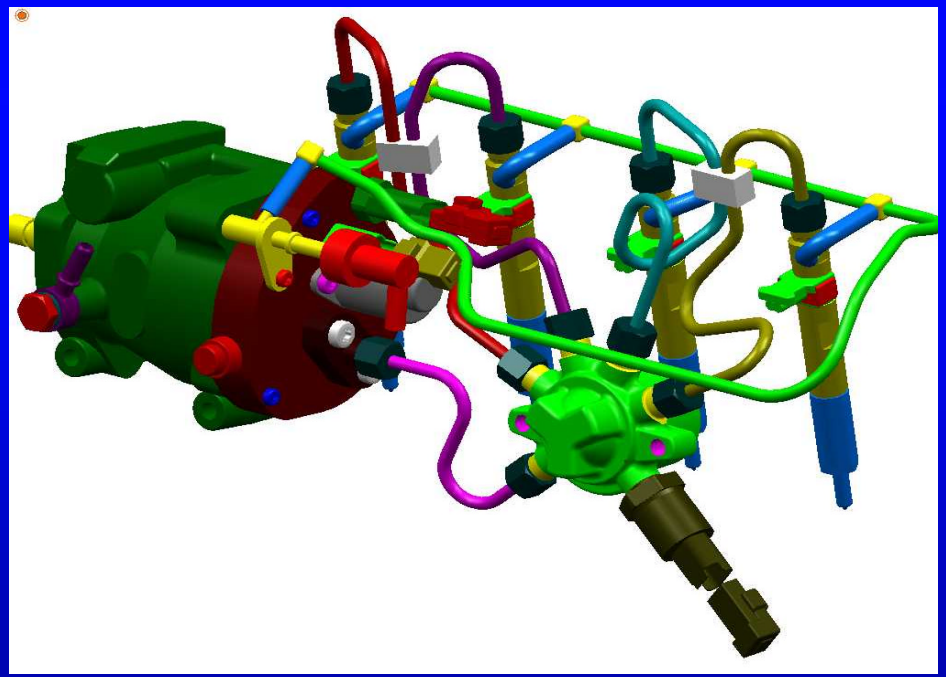
- Fault-tolerant controller software in safety-critical units in petrochemical and chemical plants
- level 3 HW redundancy
- 70,000 lines of C
- 140,000 lines of Ada
- 10,000 man-hours of testing saved
- 6-12 calendar months saved

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Industrial Applications

Automotive

- Major international automotive supplier
- 200,000 LOC diesel engine control
- Serious errors found on a sample of 32 modules of a validated application
- Problems previously not found despite 100% unit test coverage with automated test tools



This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Run-time errors
Testing

Detecting a run-time error requires:

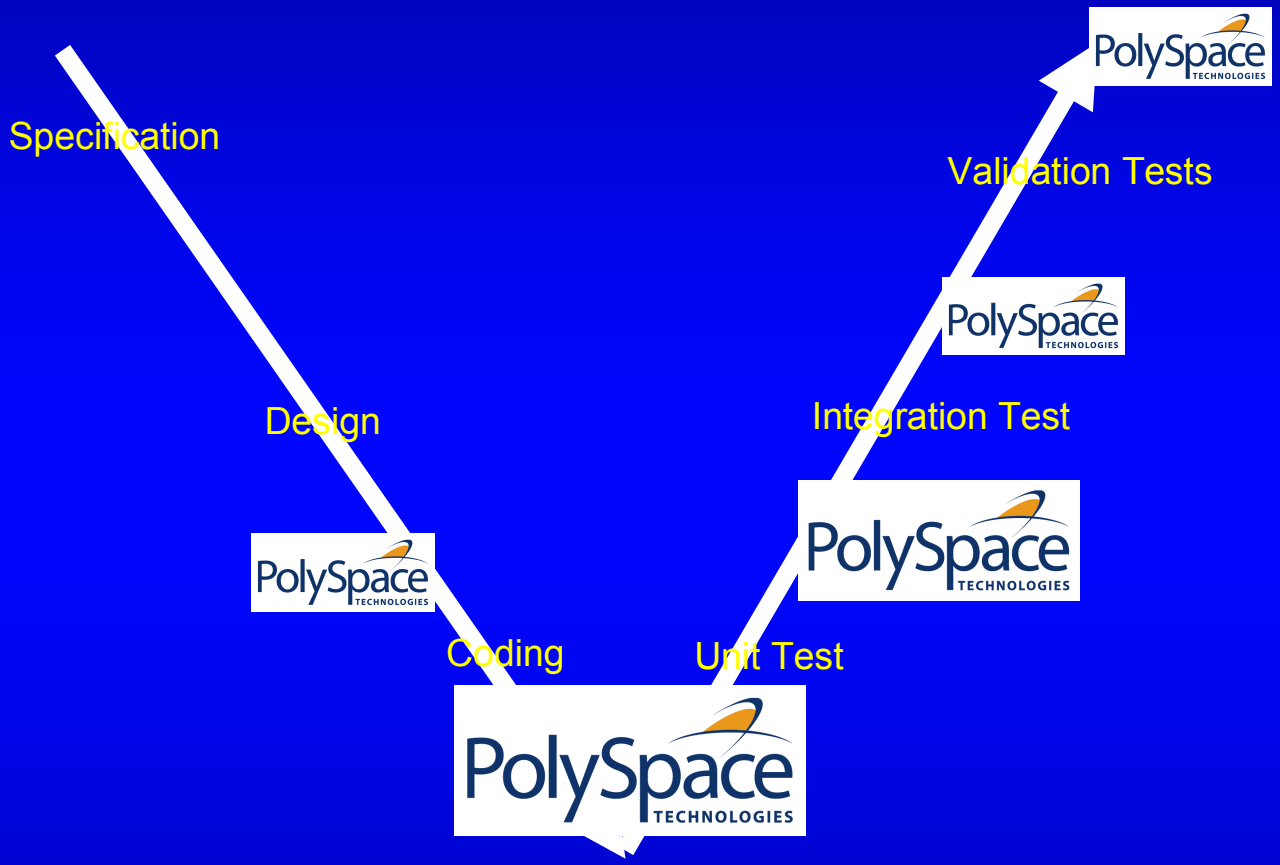
1. Executing the right statement;
2. with the right combination of values;
   as mere execution of the statement may not be enough
3. Detect the anomaly if it occurs,
   as triggering the error may not be enough.
4. Trace back the error in the code to fix it.

⇒ Even achieving 100% statement coverage during tests may not be enough to detect all run-time errors.

⇒ Abstract Interpretation covers these three points and thus exhaustively detects run-time errors

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Our recommendation



Detect errors earlier and avoid debugging and before they raise constraint error and cause test campaign disruption and a great deal of lab hours losses

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Our recommendation

- DURING CODING PHASE AND BEFORE UNIT TESTING:
  - Use PolySpace Verifier:
    - → For the earliest run-time error detection ever
    - → For verifying compliance with I/O range specifications (pragma assert based)
- BEFORE INTEGRATION TESTING
  - Use PolySpace Verifier:
    - → For detecting run-time errors before they cause test campaign disruption, test lab hours losses and annoying debugging activity
- AS CRITERIA FOR FINAL ACCEPTANCE
  - Use PolySpace Verifier:
    - → Make sure testing and validation thoroughly performed (Independent, non-intrusive, repeatable and semantic based V&V tool applicable to outsourced code too).

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Some PolySpace Customers

- **Automotive** (Delphi, TRW, Renault, Nissan, Valeo, Denso, Magnetti-Marelli, Autoliv,…);
- **Railways** (Westinghouse, RATP, SNCF, Faiveley, CSEE Transport, Alstom, Dimetronics, Technicatome, …);
- **Consumer Electronics/Telco** (General Electric, Neopost, NTT DoCoMo, France Telecom, …);
- **Energy, Chemical Industry** (Invensys, IRSN, EDF, …);
- **Defense** (Matra BAe Dynamics, EADS LV, EADS Missiles, Sagem,…);
- **Avionics** (Airbus, Diehl Avionics, Flight Visions, Smiths Industries, Alenia, BAe Systems, Thales Avionics, Honeywell, Snecma, …);
- **Space** (NASA, Astrium, EADS LV, EADS Sodern, Chinese Space Agency, CNES, Alcatel Space, …).

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002

Static Verification by Abstract Interpretation

- Has solid mathematical foundations
  - but AI can be used without theory background
- Has reached industrial maturity
- Complements coding rule checkers and testing
- Is a radical breakthrough in SW engineering enabling:
  - **The earliest detection of run-time errors**

    → shortens V&V cycle
  - **Automation of code inspections**

    → improves repeatability, delay control & quality
  - **Verification of codes with high number of calibrations**
  - **Exhaustive detection of errors without writing test cases**

    → Strongly improves application reliability

    → Robustness testing can be alleviated

This document is the property of PolySpace Technologies SA and shall not be communicated to third parties and/or reproduced without prior written agreement. Its contents shall not be disclosed.
© PolySpace Technologies SA – 2002