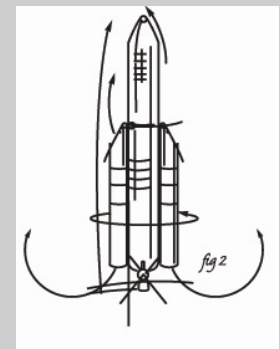
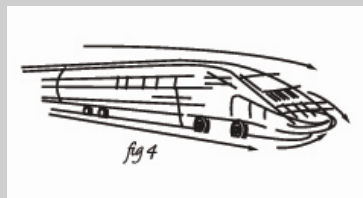


PolySpace Technologies

Abstract Semantic-based static analysis

***An alternative approach
to statically detect runtime errors
and accelerate debugging***

***Chris.Hote@PolySpace.com
AeroSpace Testing EXPO Booth #8020***



Presentation Content

- *Cost of software failures*
- *Abstract Semantics in a nutshell*
- *Market applicability*

Cost of S/W failures

**Software failures cost up to \$60bn yearly to US economy
(30%+ is software manufacturers addressing bug tracking)**

Software testing infrastructure could drastically reduce costs
by:

- ☞ Detecting bugs earlier in the software development process;
- ☞ Locating the source of bugs faster and with more precision;
- ☞ Removing more bugs before software is released.

Source: May'02 NIST report @ www.nist.gov/director/prog-ofc/report02-3.pdf

A study conducted at Berkeley and IBM Watson found that 30-40% of software defects addressed during a four-year maintenance phase on large IBM codes are due to runtime errors.

Note: Other defects relate to specification and design.

M. Sullivan and R. Chillarege, Software defects and their impact on system availability, proc. 21th International Symposium On Fault-Tolerant Computing (FTCS-21), Montreal, 1991, 2-9, IEEE Press.

Run-Time Errors

- *A.K.A. latent faults because they are so difficult and costly to find (need to debug)*
- *May cause non-determinisms, incorrect results, processor halt... with unknown consequences on system reliability and availability*

- *Read access to non-initialized data*
- *Out-of-bounds array access*
- *Concurrent access on shared data*
- *Dereferencing through null or out-of-bounds pointers*

- *Incorrect computation*
 - *Integer Overflow/Underflow*
 - *Division by zero*
 - *Square root of negative value*
- *Illegal type conversion*
- *Unreachable code*
- *And more...*

How runtime errors impact development process?

- Phase I: An anomaly surfaces during tests



Bug Bash by Hans Bjordahl



Copyright 2005 Hans Bjordahl



<http://www.bugbash.net/>

How runtime errors impact development process?

- *Phase II*

- *Need to report anomaly*
- *Need to reproduce anomaly*
- *Need to debug and fix anomaly*
- *Need to document changes made on code*
- *Need to analyze impact of code change*

- *Phase III*

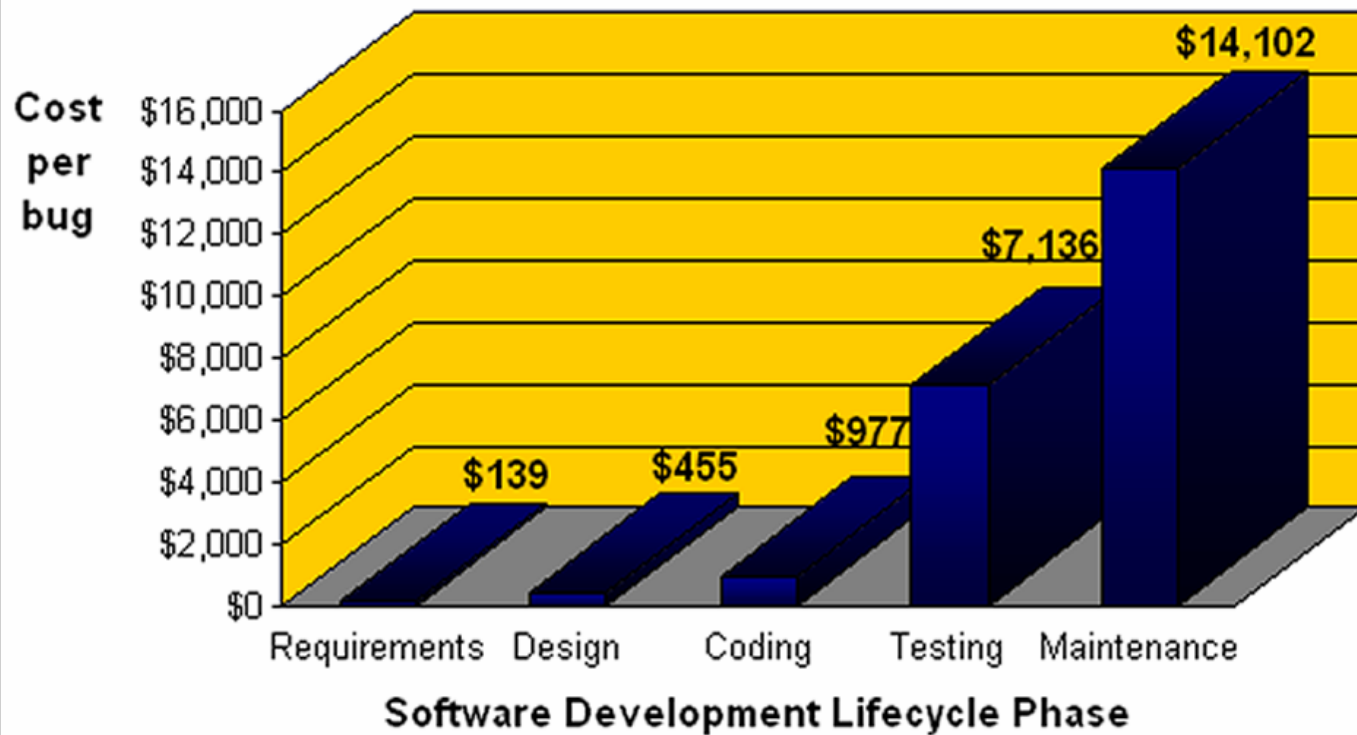
- *Define and run non-regression tests*
- *Continue tests*
- *Back to phase I if new anomalies manifest.*

Which costs involved?

Cost of late detection of errors

Every detection of S/W error at code level leads to \$\$\$ savings!!

Source: B. Boehms and V. Basilli, "Software Defect Reduction Top 10 List", IEEE Computer



Conventional Bug Detection Techniques

Static Approach

- *Code review is effective although time-consuming and not reproducible.*
- *Existing static analyzers don't tell which code is correct and suffer from high false-positive rate.*
- *Well-adapted for coding-rules checking.*

```
int tab[100];
int i, *p = tab;

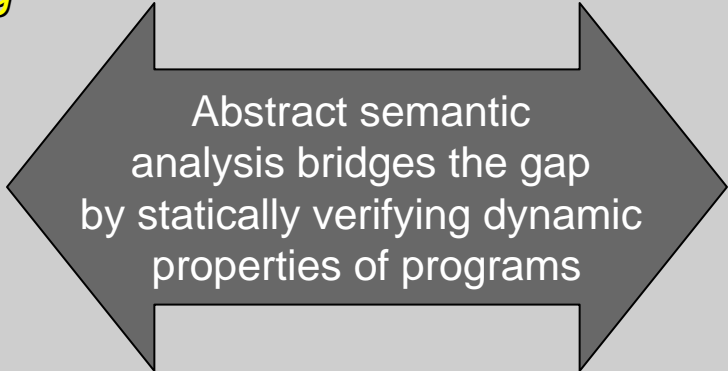
for(i = 0; i < 100; i++, p++)
    *p = 0;

if(random_int() == 0)
    *p = 5;
i = random_int();
if (random_int()) *(p-i) = 10;

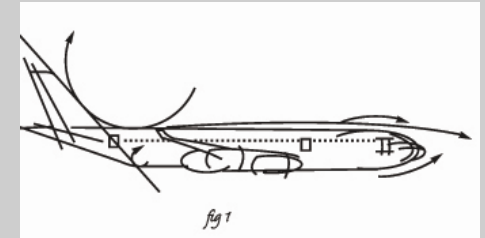
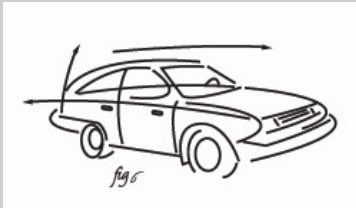
if (0<i && i<=100)
{ p = p - i;
  *p = 5;
}
```

Dynamic Approach

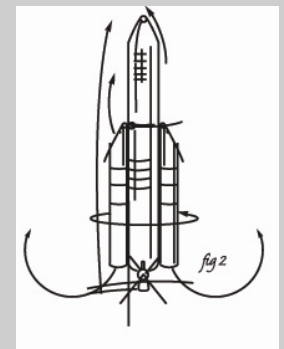
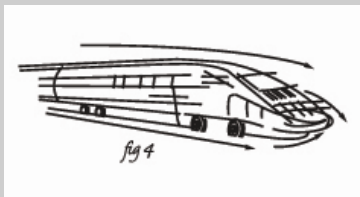
- *Lead to extra debugging costs when errors are found*
- *Well-adapted and necessary to validate functional and real-time behaviors provided no runtime errors alter the results.*



Abstract semantic analysis bridges the gap by statically verifying dynamic properties of programs



Abstract Semantic Analysis in a Nutshell



Abstract Semantics Proposal

Underlying technology

- ***Rely on a very wide base of mathematical theorems that provide the rules to statically analyze the dynamic of complex systems (pioneered by Kildall, 1973 and Wegbreit 1975)***

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

B. Wegbreit, Property extraction in well-founded property sets, IEEE Transactions on Software Engineering, 1(3) :270-285, 1975.

- ***Instead of enumerating each state of a program, represents program states in more general form (an abstraction) and provides the rules to manipulate them***
- ***Performs an analysis of all software constructs and automatically provides an individual diagnostic for each one before program is run, tested or shipped.
First industrial experiment performed late 1996 (A501).***

Abstract Semantics Proposal

Geometrical Analogy

Bridging the gap between static and dynamic analyses

Verifying: $x = x / (x-y);$

Potential Run-Time Errors

- Are x & y initialised?
- Could a division by 0 occur?
- Could there be an underflow or overflow on '-', '/' or '=' ?

The following slide focuses on the check for a division by 0

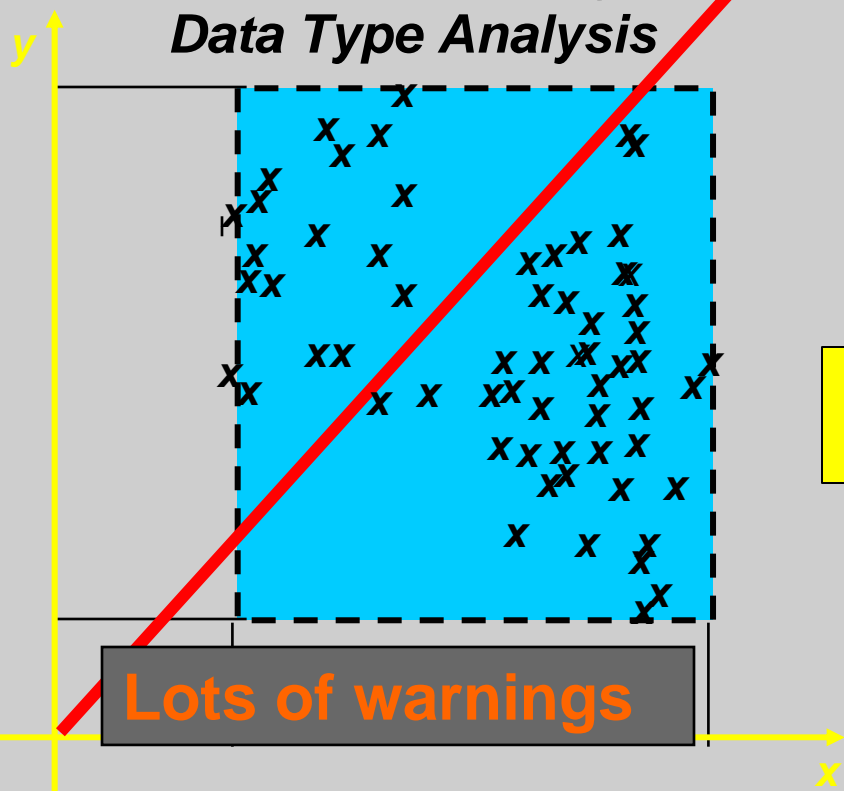
Abstract Semantics Proposal

Geometrical Analogy

Bridging the gap between static and dynamic analyses

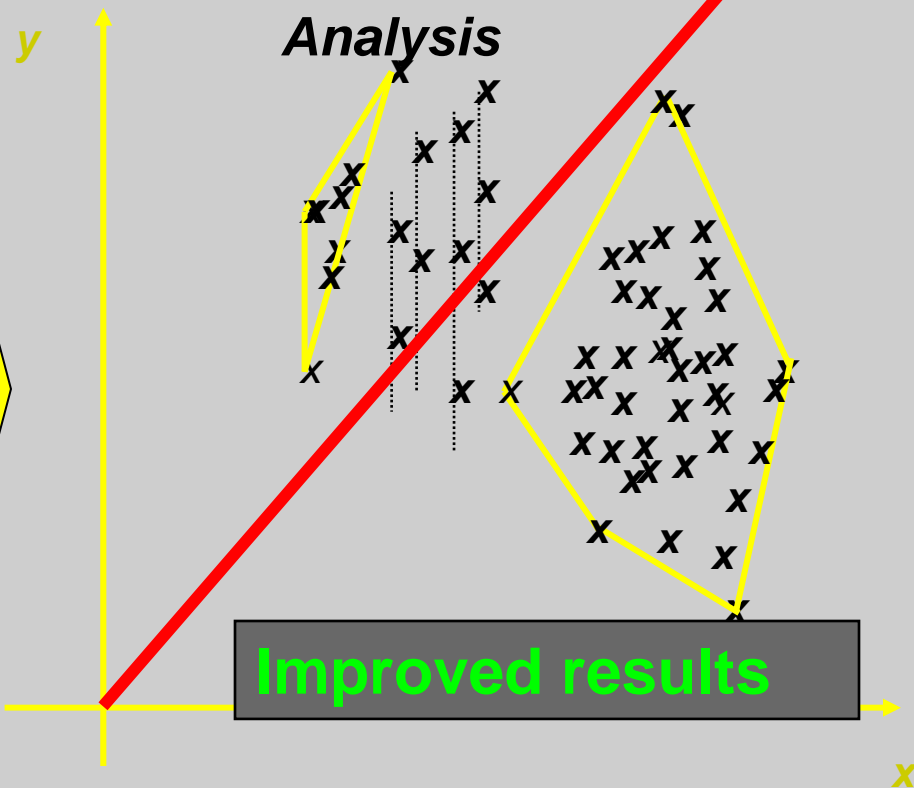
Verifying: $x = x \wedge (x-y)$;

Dynamic Testing,
Data Type Analysis



Lots of warnings

Abstract Semantic
Analysis

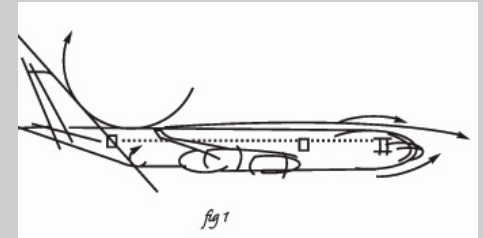


Improved results

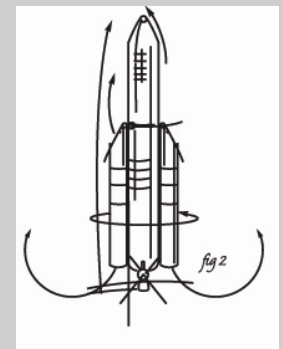
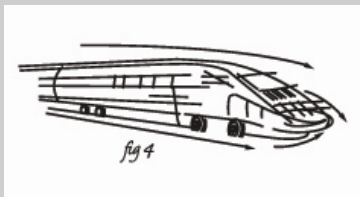
Abstract Semantics Proposal

Main features

- **Semantically based:**
Errors are directly detected not thru their consequences as it happens during testing (debugging is greatly accelerated)
- **Exhaustive:**
All possible input parameter combinations are scrutinized and each code section gets a diagnostic (far beyond what conventional analyzers can offer)
- **Non intrusive-Non invasive: Source code only**
 - **No test cases to write,**
 - **No code instrumentation needed**
 - **No execution of code needed**



Market Applicability



Runtime errors detected at compile time

Green
safe

Red
bug

Grey
dead

Orange
warning

```

70     static void Pointer_Arithmetic ()
71     {
72         int tab[100];
73         int i, *p = tab;
74
75         for(i = 0; i < 100; i++, p++)
76             *p = 0;
77
78         if(get_bus_status() > 0)
79         {
80             if(get_oil_pressure() > 0)
81                 *p = 5; /* Out of bounds */
82             else
83                 i++;
84         }
85
86         i = random_int();
87         if (random_int()) *(p-i) = 10;
88
89         if (0<i && i<=100)
90         { p = p - i;
91           *p = 5;      /* Safe pointer access */
92         }
93     }

```

Business proposal:

1

Find the bugs earlier before they cause troubles later on during tests

2

Find bugs that would not have been found, that test cases could not reveal

- Non-initialized data
- Memory corruption

3

Find bugs that would not have been found because of missing test cases

- Overflow

Abstract Semantics Proposal

Industrial application

- *Non-intrusive and scalable: applicable as soon as during coding phase (one file, one module,...)*
- *Automatic, repeatable (CPU-based) and quantitative results (reproducible quality guidelines)*
- *Cost-effective:*
Who can afford fixing syntax errors today without compiler log file?
The solution exists now for runtime errors.

Static Analysis and Industry Standards

- *Runtime errors may cause major issues in the embedded systems industries as they are difficult to reproduce and may cause substantial damages to S/W development productivity and/or corporate branding*
- *Thanks to the improvement of static analysis techniques, more industrial standards adopt those techniques as a viable alternative to white-box tests and code inspection*

Overall Conclusions

- New static analysis techniques based on abstract semantics unearth significant possibilities to reduce S/W development costs providing earlier detection of bugs ever achieved.
- New static analysis techniques based on abstract semantics substantially improve software quality at lower costs providing exhaustive diagnostic on all code sections of programs.
- Major market opportunities for embedded system industry where runtime errors are most severe in terms of consequences during operation and in terms of testing costs.
E.g.:
MISRA-C:2004 is now requiring the detection of runtime errors for embedded software.

- Have a demo at booth #8020
 - Live code analysis: Bring your code!
 - Integration with code generators and design tools

- Visit our partners
 - DO-178B expertise center: HighRely #8020
 - Compiler, IDE, testing tools: #8020 and #8025