Program Analysis

Dr. Florian Martin
AbsInt
Introduction

• For Questions:
  • ask immediately
  • mail florian.martin@absint.com
  • mail esd07-tutor@gigasun.cs.uni-sb.de
Program Analysis (PA)

• Classical compiler technique
• Prove properties about programs to generate more efficient code
• Example: Loop Invariant Code Motion

```
for (i=0; i<10; i++) {
    z = r*r;
    k = k*i;
}
```

```
z = r*r;
for (i=0; i<10; i++) {
    k = k*i;
}
```
PA in Embedded Systems

PA to check the non functional correctness of the program

- Pointer errors
- Overflow
- Division by zero
- Precision analyzer
- Stack overflow
- Timing analysis
Ariane 5
Ariane 5

- June 4, 1996
- Rocket with cargo valued $500 million
- Error in the inertial reference system
- 64 bit floating point number relating to the horizontal velocity was converted to a 16 bit signed integer
- The number was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus the conversion failed.
PolySpace Verifier


**Typical Run-Time Errors Detected**

- Overflows and underflows
- Division by zero and other arithmetic errors
- Out-of-bounds array access
- Illegally dereferenced pointers
- Read-only access to noninitialized data
- Dangerous type conversions
- Dead code
- Access to null this pointer (C++)
- Dynamic errors related to object programming and inheritance (C++)
- Errors related to exception handling (C++)
- Noninitialized class members (C++)
PolySpace Verifier

• **Green**: Proven reliable under all operating conditions
  **Red**: Proven faulty each time the operation is executed
  **Grey**: Proven unreachable (may indicate a functional issue)
  **Orange**: Unproven code section (a run-time error might occur under certain operating conditions)
Patriot

- 25/02/91: a Patriot missile misses a Scud in Dharan and crashes on an American building: 28 deads.

- Cause:
  - the missile program had been running for 100 hours, incrementing an integer every 0.1 second
  - but 0.1 not representable in a finite number of digits in base 2
  - $\frac{1}{10} = 0.00011001100110011001100 \ldots$

  Truncation error $\sim 0.000000095$ (decimal)

  Drift, on 100 hours $\sim 0.34s$

  Location error on the scud $\sim 500m$
Fluctuat

Static analyzer for C Programs with floating-point computations

Can compute

• Bounds of floating-point values

• **Bounds on the discrepancy error between the real and floating-point computations**

• If possible, the main source of this error
Altona Signal Box (=Stellwerk)

1995 the old signal box in Hamburg Altona was replaced by a 486 real time system. The machine crashed several times and a reboot took 10 min.
The whole station was closed. This had an impact on many trains in Germany.
After two days the bug was found: a stack overflow
StackAnalyzer

Stack usage of a single function
- local
- global

Call history from entry Point to function
StackAnalyzer

Maximum system stack usage

Category 1 ISR

Task context offset

Maximum User stack usage

(C16x/ST10)
Hard Real-Time Systems

- Controllers in planes, cars, plants, … are expected to finish their tasks within reliable time bounds.
- Schedulability analysis must be performed
- Hence, it is essential that an upper bound on the execution times of all tasks is known
- Commonly called the Worst-Case Execution Time (WCET)
aiT: Timing Details

Verst Case Execution Time: 508 cycles = 0.984 ms
### Analysis reports

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Entry</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>7</td>
<td></td>
<td>2914 add</td>
<td></td>
</tr>
<tr>
<td>MainNoPrime</td>
<td>4</td>
<td>main</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>PrimeT</td>
<td>6</td>
<td></td>
<td>2713 max</td>
<td></td>
</tr>
<tr>
<td>Operation: add</td>
<td></td>
<td></td>
<td>893</td>
<td></td>
</tr>
<tr>
<td>PrimeN</td>
<td>1</td>
<td>prime</td>
<td>2713 With all subroutines</td>
<td></td>
</tr>
<tr>
<td>MainNoPrime</td>
<td>4</td>
<td>main</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>PrimeN</td>
<td>1</td>
<td>prime</td>
<td>2713 With all subroutines</td>
<td></td>
</tr>
<tr>
<td>PrimeT</td>
<td>6</td>
<td></td>
<td>2713 max</td>
<td></td>
</tr>
<tr>
<td>Operation: add</td>
<td></td>
<td></td>
<td>893</td>
<td></td>
</tr>
<tr>
<td>PrimeW</td>
<td>2</td>
<td>prime</td>
<td>776 Without even and divides</td>
<td></td>
</tr>
<tr>
<td>Swap</td>
<td>3</td>
<td>swap</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>SwapI</td>
<td>5</td>
<td></td>
<td>117 add</td>
<td></td>
</tr>
<tr>
<td>Swap</td>
<td>3</td>
<td>swap</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td></td>
<td></td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Pathfinder (Priority Inversion)
Pathfinder (Priority Inversion)

Picture taken from

http://www.netrino.com/Embedded-Systems/How-To/RTOS-Priority-Inversion

Further reading:

• http://en.wikipedia.org/wiki/Priority_inversion
SymTA/S by

- Worst-Case Schedules
- End-to-End Timings
- Bottleneck Detection
- System Optimization
Software Failures

• Collection of Software Bugs at
  http://www5.in.tum.de/~huckle/bugse.html
Additional Reading

• Nielson, Nielson, Hankin:
  Principles of Program Analysis
  Springer

• PAG/WWW
  http://www.program-analysis.com