AstréeA
From Research To Industry

Dr.-Ing. Stephan Wilhelm, AbsInt GmbH

Workshop on Static Analysis of Concurrent Software
Edinburgh, 2016
AbsInt Angewandte Informatik GmbH

- Provides advanced development tools for embedded systems and tools for validation, verification, and certification of safety-critical software.
- Founded in February 1998 by six researchers of Saarland University, Germany, from the group of programming languages and compiler construction of Prof. Dr. Dr. h.c. mult. R. Wilhelm.
- Privately held by the founders.
- Headquarters in Saarbrücken, Germany.
- Our customers come from all industry sectors where safety-relevant software is used:
  - Aerospace & defense
  - Automotive
  - Nuclear & wind energy
  - Railway control
  - Medical devices
Astrée for C

- Sound static analysis based on abstract interpretation to prove absence of runtime errors.
- Astrée detects:
  - Array index out of bounds
  - Int/float division by 0
  - Invalid pointer dereferences
  - Arithmetic overflows and wrap-arounds
  - Floating point overflows and invalid operations (Inf and NaN)
  - Uninitialized variables
  - **Data races, inconsistent locking**
  - + Floating-point rounding errors taken into account
  - + User-defined assertions, unreachable code, non-terminating loops
  - + Supports MISRA, CERT, CWE
ASTRÉÉ(A) Timeline

Subjects of this talk

2001
ASTRÉÉ
ENS Paris

2009
ASTRÉÉA
ENS Paris

Astrée for C
AbsInt

2016
Practical Use
Various industries

15.10
AbsInt

...
PRODUCT INTEGRATION
The Average Industrial User

- Is not an expert C programmer or computer scientist!
  - Often has a background in testing and doesn’t know the software under analysis.
  - Separate teams for development and testing/validation/verification.
  - Model based development (=> C code is generated).

- Expects:
  - a simple, automatic setup procedure,
  - that the analysis runs fast and scales to complex systems,
  - that results are reliable, precise and easy to understand.

A static analysis tool must save time and money!
Consequences for Integrating AstréeA

1. **Simplification** of the setup procedure.

2. **User friendlier** display of new results like data races.

3. **Tuning** of performance and precision to typical user codes.
Concurrent Analysis Setup

- AstréeA was built around abstract OS models.
- OS models are shipped with the product and integrated with the preprocessing phase.
- OS models have been extended to report invalid usages of OS services.
- Simplified setup for OSEK by evaluation of OIL files.
Data Races vs. “Classic” RTEs

○ “Classic” run-time error:
  ▪ Detected in a single code location in multiple analysis contexts.
  ▪ Aggregating analyzer findings per code location results in a relatively small number of places that the user has to check.

○ Data race:
  ▪ Detected in many code locations.
  ▪ Which of these code locations access the same variable?
  ▪ Which parallel processes are involved?
Displaying Data Races

- Aggregate findings about data races per memory location.

- Show involved functions in the call tree.
RESULTS AND FEEDBACK
Example Automotive Projects

- Development versions of 2 projects with known issues:
  - Automotive 1 (Auto_1)
    - Partial OSEK project, configured by .OIL file
    - 3 application components,
      - 1 handwritten application component (control)
      - 2 application components generated by TargetLink (logic and control)
    - 2 tasks
  - Automotive 2 (Auto_2)
    - Complete AUTOSAR 3.2. project, including entire ECU code and AUTOSAR stack except NvM and CAN. Configured by .OIL file.
    - Application code mostly generated by TargetLink with some handwritten parts (e.g. CDD)
    - 11 tasks, 13 ISRs, 2 counters and 9 alarms
### Initial Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Auto_1</th>
<th>Auto_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC (preprocessed, without blank/comment lines)</td>
<td>177.576</td>
<td>2.574.481</td>
</tr>
<tr>
<td>Selectivity (number of lines proven correct)</td>
<td>99.5%</td>
<td>99.9%</td>
</tr>
<tr>
<td>Potential Run-time errors</td>
<td>774</td>
<td>702</td>
</tr>
<tr>
<td>Number of Shared variables</td>
<td>851</td>
<td>1874</td>
</tr>
<tr>
<td>Variables with data races (number/percentage)</td>
<td>6 (0.7%)</td>
<td>1481 (79%)</td>
</tr>
<tr>
<td>Data flow anomalies (infinite loops)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Reached code</td>
<td>78%</td>
<td>64%</td>
</tr>
<tr>
<td>Max RAM</td>
<td>2.6 GB</td>
<td>28.4 GB</td>
</tr>
<tr>
<td>Analysis time</td>
<td>30 m</td>
<td>1d 10h 8m</td>
</tr>
</tbody>
</table>

- All 6 findings about data races in Auto_1 were justified.
- Lots of false alarms in Auto_2.
General User Feedback

- Analysis works very well for smaller and medium sized codes.

- Higher analysis times on large codes (compared to sequential analysis) can be problematic.

- Users also asked for
  - Deadlock detection (new in next release 16.10).
  - OS model for POSIX threads (not yet available in the product).
  - Precision improvements for software running on OSEK/AUTOSAR (see next section).
CHALLENGES AND IMPROVEMENTS
OSEK / AUTOSAR

- Priority Ceiling Protocol (PCP).
  - Tasks dynamically change their priorities.
  - Original implementation was sound but imprecise.
  - Precise priorities are available with new release 16.10.

- Implicit Global Locking.
  - Use of [Suspend|Resume][All|OS]Interrupts functions as main synchronization primitives instead of explicit mutex locks.

    ```c
    SuspendAllInterrupts();
    /* write to shared variables */
    ResumeAllInterrupts();
    ```

- Original implementation assumed impossible interactions.
Initializations in Concurrent Phase

- Additional initializations precede periodic tasks.
- Problem: Exclude impossible data races between initialization and periodic phase.
Current Situation

- Example: Auto_2

<table>
<thead>
<tr>
<th>Project</th>
<th>original</th>
<th>improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC (preprocessed, without blank/comment lines)</td>
<td>2,574,481</td>
<td>2,574,481</td>
</tr>
<tr>
<td>Variables with data races (number/percentage)</td>
<td>1,481 (79%)</td>
<td>205 (10.9%)</td>
</tr>
<tr>
<td>Reached code</td>
<td>64%</td>
<td>72%</td>
</tr>
<tr>
<td>Max RAM</td>
<td>28.4 GB</td>
<td>23.3 GB</td>
</tr>
<tr>
<td>Analysis time</td>
<td>1d 10h 8m</td>
<td>14h 24m</td>
</tr>
</tbody>
</table>

- The improved analysis uses
  - More precise handling of dynamic priorities.
  - More precise handling of [Suspend|Resume][All|OS]Interrupts.
  - Separation of initialization and periodic phase.  [Development feature]
  - Improvements for analyzing AUTOSAR libraries.
Lock-Free Implementations

- Example:

  Process 1
  1. append data to buffer
  2. increase `max_valid` buffer index

  Process 2
  read `buffer` entries up to `max_valid` index

- No locking is required if all accesses to `max_valid` and `buffer` entries are atomic operations.
- Notion of atomic memory accesses has been added to the analyzer.
  - Specified per type in the ABI settings.
- Option: no data race alarms for unprotected accesses to volatile shared variables if all accesses are atomic.
The Future: Big Multicore Systems

- Example automotive setup:
  - An **application** is a collection of tasks.
  - Applications are **mapped to cores**.
    - Usually one application per core.
  - Each core has its own scheduler and OS.
  - Shared memory between cores.
  - Applications are **synchronized by spinlocks**.
  - A task on core 1 can “access” application on core 2.
    - Event set in task on core 1, received by application on core 2.
  - Task chaining between applications on different cores using events or spinlocks.
Conclusion

- The majority of safety critical software is concurrent.

- Concurrent software is nowadays successfully analyzed by industrial users of Astrée.

- We are continuously working on improving precision and performance in relevant use cases.

- We expect the automatic sound analysis of concurrent software to become more and more common in the relevant industries.
Thank you!

email: info@absint.com
http://www.absint.com