SAFE RIVER
Safety & Security Forge

# Computer Aided Extrinsic Robustness Verification

**Christèle Faure**

**Principal scientist**

**Christele.faure@safe-river.com**

SAFE RIVER
Safety & Security Forge

# Industrial Problem

**Given the *specification of input values*,**

**is it possible to *verify* that the source code of a program**

**is *robust* with respect to *erroneous inputs and memory alterations*?**

SAFE RIVER
Safety & Security Forge

# Software Robustness

- **“Art of making software behave reasonably in exceptional situations”**
- **Robustness failures lead to software false executions**
- **Come from**
  - Software bugs : Intrinsic
  - Environment of execution problems : Extrinsic
    - Sensor problems
    - Memory alterations

SAFE RIVER
Safety & Security Forge

# Enforcement of Intrinsic Robustness

- **Implementation: Protects against false executions of dangerous operations**
- **Example: protect against division by zero**

| | | |
|---|---|---|
| 1 | Assert (d != 0); | If (d != 0) /* error test*/ |
| 2 | e = n/d; | { e = n/d;} |
| 3 | | else { /* error handler*/ }; |

- **Intrinsic robustness enforcement**
  - A test protects against false executions of the dangerous operation
  - and branches to
    - The dangerous operation
    - The error handler

SAFE RIVER
Safety & Security Forge

# Automatic Verification of Intrinsic Robustness

- **Dangerous events: Runtime errors (RTE)**
- **Dangerous operations lead to (not completely specified in the language norm)**
  - Undefined behavior
  - Unspecified behavior
  - Implementation defined behavior
- **Verify the absence of RTE**
  - Static analyzers based on abstract interpretation: sound , complete
  - Numerical lattices:
    - Non relational: Interval, congruency …
    - Relational: Convex polyhedrons, …
- **Existing tools**
  - Astrée: ENS + INRIA + Absint
  - PolySpace: The MathWorks (PolySpace Technologies)
  - Frama-C: CEA List
  - Code Hawk: Kestrel Technology (C Global Surveyor NASA)

SAFE RIVER
Safety & Security Forge

# Extrinsic Robustness

- **Dangerous events**
  - Un-intentional erroneous input values (≠ security)
  - Memory alterations
- **Extrinsic robustness enforcement**
  - Do not trust input values
  - Check the value w. r. t. domain before consumption

SAFE RIVER
Safety & Security Forge

# Extrinsic robustness enforcement

- **Global input: Phase_id ∈ [0..MAX_PHASE]**

```
    {...
11  scanf(%n, Phase_id);
12  if ((Phase_id <0) || (Phase_id >= MAX_PHASE))
13  { /* handle the phase identification error */ };
14  /* else nothing to do */
15  switch (Phase_id) {
16          case 3 : ...;
            case 2 : ;}
    ...}
```

SAFE RIVER
Safety & Security Forge

# Robustness enforcement rule

- **Do not trust input values**
  - Impossible to implement in practice
  - Too much extra calculation
- **Practical enforcement rule**
  - Put a robustness check (target input, correctness domain, location)
  - For each non pointer input
  - Before value consumption

| ***Target input*** | ***Correctness domain*** | ***Location*** |
|---|---|---|
| **Global input** | **From the specification** | **After acquisition** |
| **Input parameter** | **To be computed** | **After function start** |

SAFE RIVER
Safety & Security Forge

# Robustness verification

- **No automatic tool**
- **Verification of the coherency between**
  - Actual enforcement check
  - Expected enforcement check
- **Verification (for each target input)**

| *Check* | *Correctness property* | *Complexity* |
|---|---|---|
| **Input** | **Check all inputs** | |
| **Location** | **Protection of all consumptions** | |
| **Domain** | **Coherency w. r. t. global input domains** | |

SAFE RIVER
Safety & Security Forge

# Robustness verification

- **Verification (for each target input)**

| *Check* | *Required computation* | *Mode* |
|---|---|---|
| **Input** | **Identification of** | |
| | ➢Global inputs | ➢ Automated |
| | ➢Input parameters | ➢ Manual |
| **Location** | **Identification of** | |
| | ➢Production sites: lower bound | ➢ Manual |
| | ➢Consumption sites: upper bound | ➢ Manual |
| **Domain** | **Computation of** | |
| | ➢Propagated input domain (⇔ expected) | ➢ Automated |
| | ➢Coherency between actual and expected domains | ➢ Automated |

SAFE RIVER
Safety & Security Forge

# Automatic Computations

- **PolySpace TMW**
- **Propagation of value domains**
  - From global inputs (instrumentation by assert)
  - Function parameters
  - Extracted using Inspection Point (instrumentation by IPT)
- **Domain coherency**
  - Actual domain
  - Propagated domain
  - Coherency verification
    - Protective assert never fail (green)
    - Error handler never executed (grey)

SAFE RIVER
Safety & Security Forge

# Conclusion

- **Intrinsic robustness ≠ extrinsic robustness**
- **General method**
  - Enforcement by coding rule
  - Verification
    - No automatic tool
    - Mostly manual
    - Conclusive if domain coherency automatically verified
- **Could be automated**
  - Automatic generation of checks (program instrumentation)
  - Automatic verification of checks (program verification)

SAFE RIVER
Safety & Security Forge

# SafeRiver industrial projects

- **Conception of critical systems**
  - THALES/RSS: train tracking
  - CSWT Ltd: tram-bus of DOUAI
- **Verification of embedded equipments**
  - DELPHI : low cost platform for car cabin equipment
  - AREVA T&D : control box for indoor switches (middle voltage)
- **Information system security**
  - THALES Communications : application of formal methods to cryptographic equipment
  - AIRBUS : Aircraft Information System Security