Proof-by-Instance for Embedded Network Design
From Prototype to Tool Roadmap

Marc Boyer\textsuperscript{2}, Loïc Fejoz \textsuperscript{1}, Stephan Merz \textsuperscript{3}

\textsuperscript{1} RealTime at Work, Nancy, France \textsuperscript{2} ONERA – The French aerospace Lab, \textsuperscript{3} Inria & LORIA, Nancy, France

Congress on Embedded Real-Time Software and Systems (ERTSs 20014)

\textsc{retour sur innovation}
Outline

Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion
Software qualification/certification

- **Process based quality:**
  - Per domain standards: DO 178, ECSS-Q-ST-80...
  - specification, tests, documentation
  - generation/transformation tools
  - verification tools
  - high cost

- **Proof by instance / certificate-based confidence**
  - product based quality
  - provides arguments to verify computation
  - for off-line computations
Proof by instance

Idea: for each run, provide a proof of correctness
- requires some automatic proof environment (Isabelle, Coq)
- requires modelling of the context (≈ specification)
- proof/certificate generation while computing result
- transfers confidence from tool to proof environment

Benefits:
- Checking a solution is simpler than proving full correctness of a program
- Computing software is used as a black box:
  - Open choice of the programming environment
  - Software can evolve without re-qualification

Not exactly a verification tool:
- based on formal proof checker (verifier)
- help the verification tool to makes the proof
Network Calculus Theory
Network Topology (flows and servers)
Tool Specification
Network Calculus Tool
Network Calculus in Isabelle
Isabelle/HOL Tool
Computation traces
Bounds (delay and memory)
OK/KO
Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion
Encoding Objects

\[ R \rightarrow S \rightarrow R' \]

\( R(t) \): amount of data send up to time \( t \)

- **Flow**
  \[ R : \mathbb{R} \rightarrow \mathbb{R}^+, \ x \leq 0 \implies R(x) = 0, \ x < y \implies R(x) \leq R(y) \]

  **typedef** \( ndf = \{ f :: \text{ereal} \Rightarrow \text{ereal} . \ (\forall r \leq 0. \ f r = 0) \land \text{mono } f \} \)

- **Server**, input/output relation such that \( R \xrightarrow{S} R' \implies R' \leq R \)

  **typedef**
  \[ server = \{ s :: (ndf \times ndf) \text{ set.} . \ (\forall \text{in. } \exists \text{out. } (\text{in}, \text{out}) \in s) \land (\forall (\text{in}, \text{out}) \in s. \ \text{out} \leq \text{in}) \} \]
Encoding contracts

- Real behaviour \((R, R')\) unknown at design \(\implies\) use of contracts
- Consider \((\min, +)\) convolution
  \[
  (f \ast g)(t) = \inf_{0 \leq s \leq t} \{f(t - s) + g(s)\}
  \]
- Traffic contract: arrival curve
  \[
  R \preceq \alpha \iff R \leq R \ast \alpha \quad (1)
  \]
- Service contract: service curve
  \[
  S \succeq \beta \iff \left( R \xrightarrow{S} R' \implies R' \geq R \ast \beta \right) \quad (2)
  \]
Network calculus theorem: if a flow $R$ with arrival curve $\alpha$ goes through a server $S$ of service curve $\beta$, its delay is not greater than $h(\alpha, \beta)$.

**Theorem $d$-$h$-bound:**
- assumes $in \preceq \alpha$ and $S \succeq \beta$
- shows worst-delay-server in $S \leq h$-$dev$ $\alpha \beta$
Outline

Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion
Computing and checking

- Initial set of hypotheses
- Successive application of theorems:
  - choose an adequate theorem
  - instantiate variables
  - check hypotheses
  - compute operations (sums, convolutions...)
  - generate new facts
Computing and checking

- Initial set of hypotheses
- Successive application of theorems:
  - chose an adequate consider one theorem
  - instantiate variables
  - check hypotheses
  - compute check operations (sums, convolutions...)
  - generate store new facts
Running prototype

- Generates bounds and proofs
- Able to handle a realistic industrial network:
  - 8 switches
  - 5000 flows
- Experiment results:
  - Bound accuracy: 2 times greater (i.e. worse) than state-of-the-art RTaW-PEGASE tool
  - Computation time:
    - computing and generating traces: a few minutes
    - check traces: 8 hours
- Development effort:
  - $\approx \frac{1}{4}$ for 7KLoc of Java
  - $\approx \frac{3}{4}$ for 3KLoc of Isabelle theories
  $\Rightarrow$ overhead 2x-3x for formal confidence
Outline

Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion
Extend Isabelle theories

- More theorems statements into Isabelle:
  - more accurate bounds
  - handle more scheduling policies
- With or without proofs:
  - Without:
    - confidence comes from paper-pencil published proofs
    - still prove that tool respects specification
  - With:
    - increase confidence
    - contribution to Isabelle theories
Network calculus handles “functions”
How to represent a function?
How to compute operations on functions (sum, convolutions)?
Simpler to check result than prove algorithm correctness
Current tool directly generates Isabelle proofs

Plan to define intermediate format:

- independence wrt the checker engine
- more compact, more efficient
- generate human-readable proofs (certification authorities)
Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion
Conclusion

Proof by instance
- a way to avoid long, costly and boring development process
- decouple computation and certification
- well suited for tools based on formal methods

Implementation
- Prototype developed
- Proof-of-concept validated
- Road-map for industrial tool
- Integration into RTaW-PEGASE