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

Marc Boyer[2], Loïc Fejoz [1], Stephan Merz [3]

[1] RealTime at Work, Nancy, France [2]ONERA – The French aerospace Lab, [3]Inria & LORIA, Nancy, France

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

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 ($\approx$ specification)
- proof/certificate generation while computing result
- transfers confidence from tool to proof environment

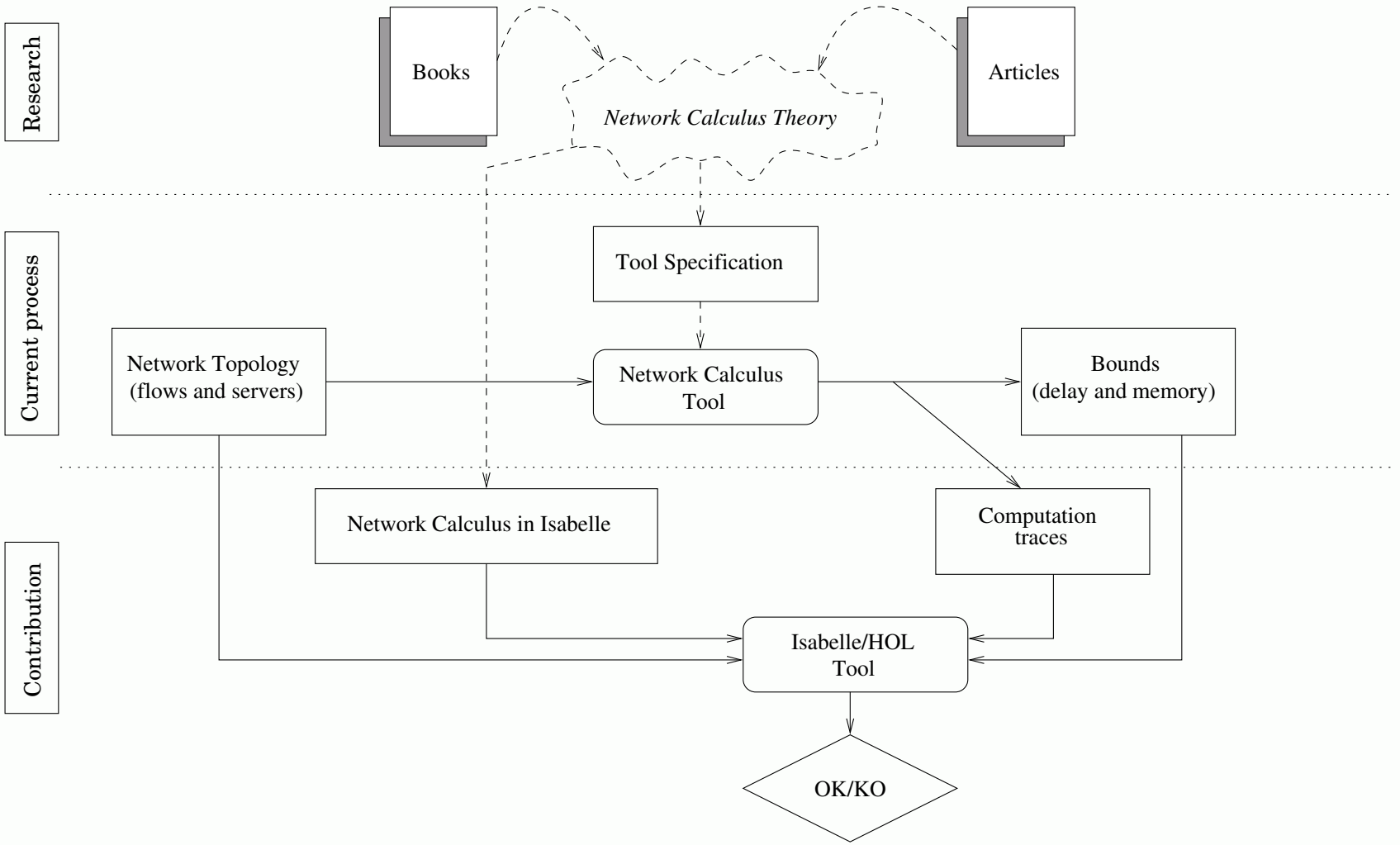
Benefits:

- 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

The full picture



Outline

Certificate-based confidence / Proof by instance

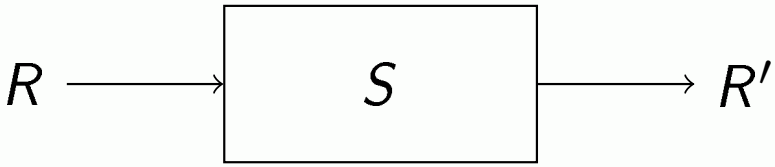
Encoding in Isabelle

Generating and checking traces

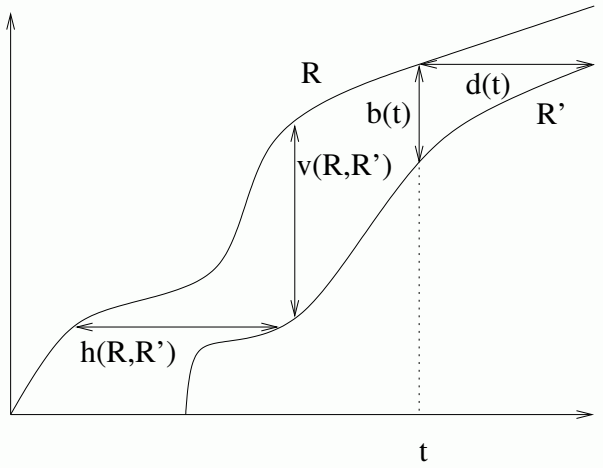
From Prototype to Tool Roadmap

Conclusion

Encoding Objects



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



- Flow

  $R : \mathbb{R} \to \mathbb{R}^+,\ x \leq 0 \implies R(x) = 0,\ x < y \implies R(x) \leq R(y)$

  typedef $ndf = \{\, f :: ereal \Rightarrow ereal \;.\; (\forall r \leq 0.\ f\ r = 0) \wedge mono\ f \,\}$

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

  typedef

  $server = \{\, s :: (ndf \times ndf)\ set. \quad (\forall in.\ \exists out.\ (in, out) \in s)$
  $\wedge \quad (\forall (in, out) \in s.\ out \leq in) \,\}$

ONERA THE FRENCH AEROSPACE LAB

Encoding contracts

- Real behaviour (R, R') unknown at design $\Longrightarrow$ use of contracts
- Consider $(\min, +)$ convolution

$$(f * g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$$

- Traffic contract: arrival curve

$$R \preceq \alpha \overset{\text{def}}{\Longleftrightarrow} R \leq R * \alpha \tag{1}$$

- Service contract: service curve

$$S \succeq \beta \overset{\text{def}}{\Longleftrightarrow} \left(R \overset{S}{\rightarrow} R' \Longrightarrow R' \geq R * \beta \right) \tag{2}$$

Encoding results

- Network calculus theorem: if a flow R with arrival curve α goes through a server S of service curve β, 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* α β

Outline

Certificate-based confidence / Proof by instance

Encoding in Isabelle

Generating and checking traces

From Prototype to Tool Roadmap

Conclusion

ONERA
THE FRENCH AEROSPACE LAB

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

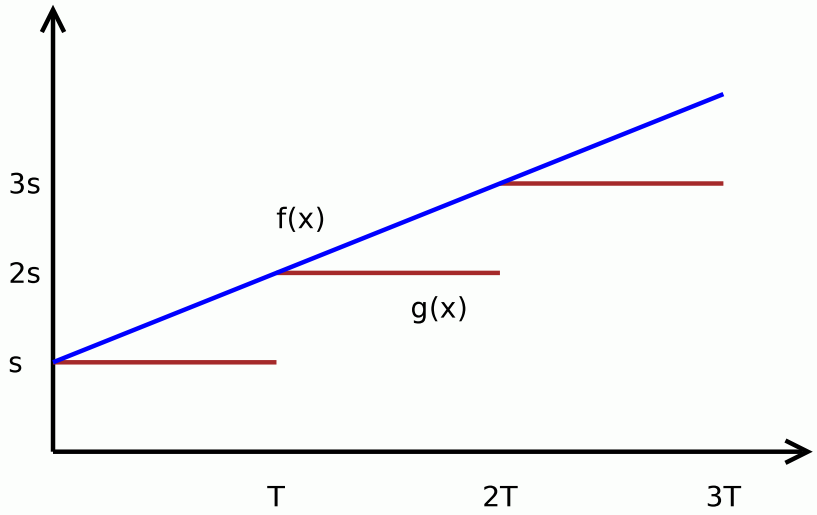
From Prototype to Tool Roadmap

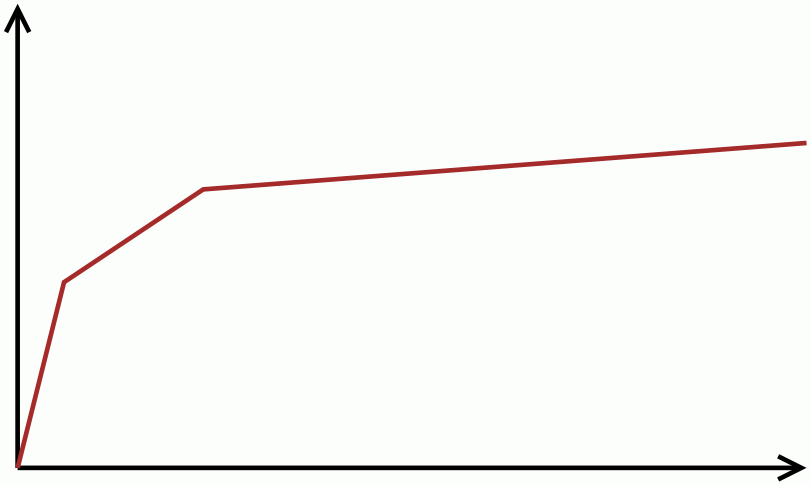
Conclusion

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

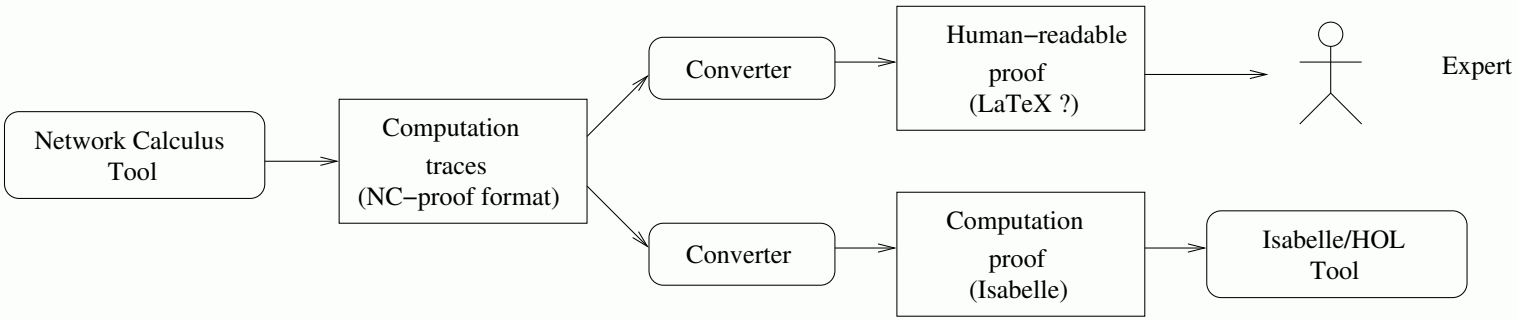
Classes of functions





- 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

Intermediate proof format



- 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)

Outline

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