Rely-Guarantee "thinking" for Real-Time Scheduling

Cliff Jones: Newcastle University
(joint with Alan Burns: University of York)

WG2.3
Athens
2025-05-21

Today's plan

- (FMSD paper; FM-Milano; general model under review)
- Real-Time Scheduling (RTS)
- challenges of formally describing a *Scheduler*
- quick reminder(?) of rely/guarantee idea
- two-minute guide to "time bands"
- how these ideas help formalise RTS
- open issues
- conclusions + related work

Real-Time Scheduling (RTS) — Alan Burns

- tasks define classes of *Job*s
  TaskInfo type defines resource demands, ...
- a *Planning* phase chooses scheduling "discipline"
  checks "schedulability"
- *Scheduler* must follow selected scheduling discipline

$$\mathit{Time} \;\|\; \mathit{Planning}\;;\; \{\mathit{Scheduler} \;\|\; \mathit{Job}_1 \;\|\; \mathit{Job}_2 \;\|\; \cdots \;\|\; \mathit{Job}_k\}$$

- aim: specification of *Scheduler*
- "mixed criticality" facilitates fault-tolerance wrt
  jobs overrunning, arriving too early, etc.

Rely/Guarantee (R/G)

- basic idea specify interference:

$$\overbrace{\sigma_0}^{pre}\ \cdots\ \overbrace{\sigma_i\ \sigma_{i+1}}^{rely}\ \cdots\ \underbrace{\underbrace{\sigma_j\ \sigma_{j+1}}_{guar}\ \cdots\ \sigma_f}$$

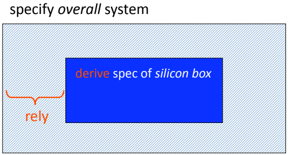
$$\underbrace{\sigma_0 \cdots \sigma_f}_{post}$$

$$\boxed{\parallel\text{-}I_c}\ \frac{\{P_1, R \vee G_2\}\ S_1\ \{G_1, Q_1\} \quad \{P_2, R \vee G_1\}\ S_2\ \{G_2, Q_2\}}{\{P_1 \wedge P_2, R\}\ S_1 \parallel S_2\ \{G_1 \vee G_2, Q_1 \wedge Q_2\}}$$

- "top down" design/record from abstract specification
  - "compositionality" is crucial [dRdBH+01]
  - compare Owicki/Gries [Jon24]
- RGSep [Vaf07] SAGL
- relations give only restricted expressiveness
  but have proved useful — RTS extra challenges

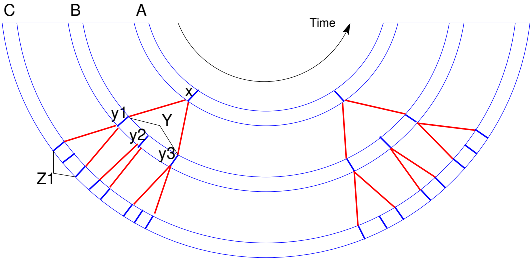
"Relying on" the environment

- R/G conceived as a (top-down) decomposition rule
- later applied to rely on non-developed components
  - e.g. physical components
  - can even "deduce the spec of control system" [BHJ20]



- of course, don't blandly "rely on"
  customer/deployer has to agree the assumptions
- "R/G thinking" $\approx$ "record assumptions"
- layered R/G for fault-tolerance
  - optimistic rely + guarantee ideal behaviour
  - weaker rely + less desirable guarantee

“Time bands” briefly! — see Burns/Hayes [BH10]



clearer specification at multiple bands (see [BHJ20])
but not refinements: implementation must satisfy all bands
“Granularity” G; only need today is “precision” ρ

Data: abstraction, reification + data type invariants (DTIs)

- data abstraction/reification in development methods
  more important than operation decomposition?
- most specifications: same collection of base types
- predicate restriction = DTI
  - useful (especially for future proofing)
  - DTIs as “meta pre/post conditions”
- R/G can become long (difficult to understand)
  - DTI as meta rely/guarantee conditions
  - reduces length/complexity of R/G conditions
- “dynamic invariants”?

Use in tackling RTS

- $Time \parallel (Planning; RunTime)$
- $Planning$
  - select discipline e.g. FCFS, EDF, FP
  - check schedulability: “Response Time Analysis”
- $RunTime = Scheduler \parallel Job_1 \parallel \cdots \parallel Job_k$
  - R/G of $Scheduler/Job$ relate to resources (time)
- $Scheduler$ design assumes Jobs will not exceed resource (WCET, arrival)
  guarantee that their Jobs will be given resource ($TaskInfo$)
- for Fault Tolerance (F-T):
  - strong assumptions require ideal behaviour
  - weaker assumptions require hi-crit serviced

Time itself

- schedules relate to *Time* in the external world
  but the *Scheduler* can only use internal $t : ClockValue$
- so our overall spec based on:
  $\Sigma = Time \to State$
  - *State* changed by "operations" — *Time* marches on!
  - "time band" idea links $ClockValue/Time$
    DTI + notion of precision ρ
  - rely on *Clock* accuracy

Time itself

$$\Sigma = Time \to State$$

where

$$inv\text{-}\Sigma(\sigma) \quad \underline{\triangle} \quad \mathcal{T}(\sigma) \wedge \mathcal{E}(\sigma)$$

$$\begin{array}{l} \mathcal{T}(\sigma) \quad \underline{\triangle} \\ \qquad (\forall \alpha \in Time \cdot \sigma(\alpha).t =_\rho \alpha) \wedge \\ \qquad (\forall \alpha_1, \alpha_2 \in Time \cdot \alpha_1 < \alpha_2 \;\Rightarrow\; \sigma(\alpha_1).t \leq \sigma(\alpha_2).t) \end{array}$$

$$\begin{array}{l} \mathcal{E}(\sigma) \quad \underline{\triangle} \\ \quad \forall \alpha_1, \alpha_2 \in Time \cdot \\ \qquad \forall j \in (\mathbf{dom}\, \sigma(\alpha_1).used \cap \mathbf{dom}\, \sigma(\alpha_2).used) \cdot \\ \qquad\quad ((\forall \alpha \mid \alpha_1 \leq \alpha \leq \alpha_2 \cdot \sigma(\alpha).run = j) \;\Rightarrow\; \sigma(\alpha_2).used(j) - \sigma(\alpha_1).used(j) =_\rho \alpha_2 - \alpha_1) \wedge \\ \qquad\quad ((\forall \alpha \mid \alpha_1 \leq \alpha \leq \alpha_2 \cdot \sigma(\alpha).run \neq j) \;\Rightarrow\; \sigma(\alpha_2).used(j) = \sigma(\alpha_1).used(j)) \end{array}$$

Forcing progress

- $\mathit{Scheduler}$ operations: $\mathit{Release}, \mathit{Overrun}, \mathit{Mode\text{-}up}$
- trigger action on Job release
  $\mathit{inv\text{-}State}$: $\forall j \in \mathit{JobId} \cdot t \leq \mathit{deadline}_j$ forces progress!
  slight simplification: deadlines can change
- $\mathit{Scheduler}$ only preserves $\mathit{inv\text{-}State}$ if it acts!
  appropriate JobId in run
  $\mathit{Scheduler}$ gets rid of a Job by giving it resource
- remember: spec $\neq$ implementation

Schedulability: Response Time Analysis

- FCFS/EDF/FP
- EDF is “optimal” for single core
- will actual WCET etc. “fit”
  including in degraded modes
- critical instant:
  consider all jobs arriving at time zero
- see MPI research: [BVB+22, MBB22] (Coq proofs)

Open issues

- marry with response time analysis
  overall specification is tricky!
- multi-core
- the $Planning/\cdots$ split has hints of ML trainingi/deployment
- “dynamic invariants” for concurrency
  - “dynamic invariants” in design (cf. loop invariants)

Conclusions

- R/G helps formalise RTS specification
  - general model: applied to various scheduling disciplines
- interesting extensions
  - $Time/ClockValue$
  - "liveness" (or progress)
  - invariant + a clock concept for termination!
    vs. (limited) use of TL
  - shades of Rick Hehner here?
- future work
  - link to "response time analysis"
  - mechanise proofs (cf. [BVB+22])
- subtext: formalism pays of more in design than *post facto*

References

A. Burns and I.J. Hayes.
A timeband framework for modelling real-time systems.
Real-Time Systems Journal, 45(1–2):106–142, June 2010.

Alan Burns, Ian J. Hayes, and Cliff B. Jones.
Deriving specifications of control programs for cyber physical systems.
The Computer Journal, 63(5):774–790, 2020.

Jonathan P. Bowen, Qin Li, and Qiwen Xu, editors.
Theories of Programming and Formal Methods, number 14080 in lncs. springer, 2023.

Kimaya Bedarkar, Mariam Vardishvili, Sergey Bozhko, Marco Maida, and Björn B Brandenburg.
From intuition to Coq: A case study in verified response-time analysis of FIFO scheduling.
In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 197–210. IEEE, 2022.

Willem-Paul de Roever, Frank de Boer, Ulrich Hanneman, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers.
Concurrency Verification: Introduction to Compositional and Noncompositional Methods.
Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

Cliff B. Jones.
Three early formal approaches to the verification of concurrent programs.
Minds and Machines, 34:73–92, 2024.

Marco Maida, Sergey Bozhko, and Björn B Brandenburg.
Foundational response-time analysis as explainable evidence of timeliness.
In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

Viktor Vafeiadis.
Modular fine-grained concurrency verification.
PhD thesis, University of Cambridge, 2007.