Prosa: A Case for Readable Mechanized Schedulability Analysis

Felipe Cerqueira, Felix Stutz, Björn B. Brandenburg
Prosa

Open-source foundation for formally proven schedulability analysis

Readable Formal Specification

Jobs
Multiprocessor
Response time
...

Mechanized Proofs

Schedulability Tests
Response-time Bounds
...

2
This Talk
This Talk

**Mechanized proofs** provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.
Mechanized proofs provide an opportunity to avoid the correctness pitfalls in real-time scheduling.

By focusing on readability and by maintaining the established proof culture, mechanized proofs can reach the community at large.
This Talk

**Mechanized proofs** provide an opportunity to avoid the **correctness pitfalls in real-time scheduling**.

By focusing on **readability** and by **maintaining the established proof culture**, mechanized proofs can reach the community at large.

Thanks to **mature proof assistants and libraries**, Prosa allows mechanizing **recent and complex schedulability analyses in reasonable time**.
Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa
Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa
What do we mean by mechanized?
RTS theory has been built with pen-and-paper proofs
RTS theory has been built with pen-and-paper proofs

Abstract view of the system

Pen-and-paper proofs

Semi-formal specification

Schedulability analysis

\[ \sum \frac{e_i}{T_i} \leq 1 \]
What is **mechanized schedulability analysis**?

- **Abstract view of the system**
- **Pen-and-paper proofs**
- **Semi-formal specification**
- **Schedulability analysis**

\[ \sum \frac{c_i}{T_i} \leq 1 \]
What is **mechanized schedulability analysis**?

- Abstract view of the system
- Pen-and-paper proofs
- Formal specification

Formal specification: $\sum \frac{e_i}{T_i} \leq 1$

We switch to a formal specification.
What is **mechanized schedulability analysis**?

**Abstract view of the system**

**Mechanized proofs**

**Formal specification**

**Schedulability analysis**

We prove theorems using a **proof assistant**

\[ \sum \frac{c_i}{T_i} \leq 1 \]
What is **mechanized schedulability analysis**?

Abstract view of the system

Mechanized proofs

Formal specification

Mechanized Schedulability analysis

The resulting schedulability analysis is **formally verified**
Why mechanized proofs?
Why mechanized proofs?

- Guaranteed Correctness
  - Trustworthy Extensions
  - Safe Composition
RTS have become more complex

Source: G. Buttazzo (Keynote @ RTSS’14)
RTS have become more complex

We need complex models to support real-world requirements

Source: G. Buttazzo (Keynote @ RTSS’14)
This complexity comes with a price

The original analysis for CAN had a bug that remained undetected from 1994 to 2006 [1].

Bugs are no longer an exception

Proofs have become so complicated that they often contain bugs.
Analysis for safety-critical systems?

How to ensure that schedulability analysis is actually correct?
Analysis for safety-critical systems?

How to ensure that schedulability analysis is actually correct?

Mechanized proofs

Opportunity: correctness is inherently guaranteed.
Why mechanized proofs?

Guaranteed Correctness

Trustworthy Extensions

Safe Composition
Analyses sometimes need refining

In most analyses, practical details are assumed to be negligible.
Analyses sometimes need refining

But when deploying actual systems, we might need to **refine the analysis.**
Analyses sometimes need refining

We call these extensions (i.e., same results + tweaks) **neighboring proofs**.

But when deploying actual systems, we might need to **refine the analysis**.
Example: incorporating release jitter

<table>
<thead>
<tr>
<th></th>
<th>Basic RTA</th>
<th>RTA with Jitter</th>
</tr>
</thead>
</table>
| **Uniprocessor** | $R_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lfloor \frac{R_i}{T_j} \right\rfloor e_j$ | $r_i \leftarrow e_i + \sum_{\tau_j \in hp_i} \left\lfloor \frac{r_i + J_j}{T_j} \right\rfloor e_j$

$$R_i = J_i + e_i + r_i$$

It has been known for more than 20 years how to incorporate release jitter into uniprocessor RTA [3].

---

Example: incorporating release jitter

<table>
<thead>
<tr>
<th></th>
<th>Basic RTA</th>
<th>RTA with Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Uniprocessor</strong></td>
<td>$R_i \leftarrow e_i + \sum_{\tau_j \in h(p_i)} \left\lfloor \frac{R_i}{T_j} \right\rfloor e_j$</td>
<td>$r_i \leftarrow e_i + \sum_{\tau_j \in h(p_i)} \left\lfloor \frac{r_i + J_j}{T_j} \right\rfloor e_j$</td>
</tr>
<tr>
<td></td>
<td>$R_i = J_i + e_i + r_i$</td>
<td></td>
</tr>
<tr>
<td><strong>Multiprocessor</strong></td>
<td>$R_i \leftarrow e_i + \frac{1}{m} \cdot \sum_{\tau_j \in h(p_i)} \left\lfloor \frac{I_j(R_i)}{T_j} \right\rfloor r_j$</td>
<td>???</td>
</tr>
</tbody>
</table>

But this result has not been proven for multiprocessor RTA.
Can we do the same for multiprocessors?

<table>
<thead>
<tr>
<th></th>
<th>Basic RTA</th>
<th>RTA with Jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Uniprocessor</strong></td>
<td>$R_i \leftarrow e_i + \sum_{\tau_j \in hpi} \left\lfloor \frac{R_i}{T_j} \right\rfloor e_j$</td>
<td>$r_i \leftarrow e_i + \sum_{\tau_j \in hpi} \left\lfloor \frac{r_i + J_j}{T_j} \right\rfloor e_j$</td>
</tr>
<tr>
<td><strong>Multiprocessor</strong></td>
<td>$R_i \leftarrow e_i + \frac{1}{m} \cdot \sum_{\tau_j \in hpi} \left\lfloor \frac{I_j(R_i)}{T_j} \right\rfloor r_j$</td>
<td>$R_i = J_i + e_i + r_i$</td>
</tr>
</tbody>
</table>

Just sum up the max jitter?
The answer is that we don’t know

**Different** system models have **different** assumptions. What if changing the model **breaks some existing proof**?
Recent case: self-suspending tasks

Misuse of release jitter in the analysis caused bugs in 12 papers related to self-suspensions!

Excerpt from [1]:

- Incorrect quantification of jitter for dynamic self-suspending task systems, which was used in [3, 4, 37, 58]. This misconception was unfortunately adopted in [12, 14, 28, 36, 40, 73, 74, 76] to analyze the worst-case response time for partitioned multiprocessor real-time locking protocols.

How to derive safe extensions?
How to derive safe extensions?

**Mechanized proofs**

Opportunity: neighboring proofs are conducted **systematically**.

We just need to **refine the analysis** and let the proof assistant recheck the proofs.

If there is a bug, { **it will always be detected.**

**we know exactly what to fix.**}
Why mechanized proofs?

Guaranteed Correctness

Trustworthy Extensions

Safe Composition
Sometimes we have to combine different analyses.

Even if each analysis is individually correct, they should not be combined if assumptions mismatch.

Example:

- **Suspension-oblivious schedulability analysis**
  - **Suspension-oblivious blocking bound**: compatible

- **Suspension-aware schedulability analysis**
  - **Suspension-aware blocking bound**: incompatible!
How to avoid mismatching assumptions?
How to avoid mismatching assumptions?

We just need to avoid stating contradictory assumptions. But this can also be mechanically verified!

Opportunity: mismatching assumptions are automatically caught by the proof assistant.
No more correctness pitfalls

**Guaranteed Correctness**

- Trustworthy Extensions
- Safe Composition

**Mechanized proofs** provide an opportunity to avoid the correctness pitfalls in real-time scheduling.
Outline of the Talk

- Why mechanized proofs?
- Challenges & Principles
- A Taste of Prosa
Outline of the Talk

- Why mechanized proofs?
- Challenges & Principles
- A Taste of Prosa
Verification has many challenges
Verification has many challenges

“Formal specifications are complex and full of symbols.”
Verification has many challenges

“Formal specifications are complex and full of symbols.”

“It might take many decades to verify all we know about real-time scheduling.”
Verification has many challenges

“Formal specifications are complex and full of symbols.”

“It might take many decades to verify all we know about real-time scheduling.”

“Knowledge about formal methods tends to be restricted to few research groups.”
Verification has many challenges

“Formal specifications are complex and full of symbols.”

“It might take many decades to verify all we know about real-time scheduling.”

“Knowledge about formal methods tends to be restricted to few research groups.”

But there’s an opportunity to change...
Principles & Goals of Prosa

1. Readability is crucial
2. Some proofs are more important than others
3. We should maintain the proof culture
4. Community involvement
Principle 1: Readability is crucial

The specification should be accessible to researchers with no previous experience with formal methods.
Principle 1: Readability is crucial

The specification should be accessible to researchers with no previous experience with formal methods.

We favor:

Many lemmas, short proofs (few dozen lines)
Long, verbose names and few cryptic symbols
Heavy use of documentation
Complex notation harms readability

Duration Calculus [Yuhua and Chaochen, 1994]

Furthermore, if there exists a ready task which is not running, then no processor should be idle. So let,

\[
SCH_m \equiv \left( \neg \Diamond \left( \text{true}; \ Run(S) \right) \land \left( \bigwedge_{i \in S} \bigvee_{j \in \alpha - S} \left( \text{Urgt}(j, i) \land [p_j.rdy] \land \text{Run}(S) \right) \right) \land \Box \left( \text{Run}(S) \land \left[ \bigwedge_{i \in \alpha - S} p_i.rdy \right] \rightarrow \#S = m \right) \right)
\]
**Complex notation harms readability**

*Duration Calculus [Yuhua and Chaochen, 1994]*

Furthermore, if there exists a ready task which is not running, then no processor should be idle. So let,

\[
SCH_m \equiv \left( \neg \Diamond \left( \text{true}; \text{Run}(S) \wedge \left( \forall i \in S \forall j \in \alpha - S (Urgt(j,i); [p_j.rdy] \wedge \text{Run}(S)) \right) \right) \right)
\wedge \Box (\text{Run}(S) \wedge \left[ \forall i \in \alpha - S p_i.rdy \right] \Rightarrow \#S = m)
\]

*Prosa*

(* A scheduler is work-conserving iff all processors are busy (non-idle) whenever a job is backlogged. *)

**Definition** `work_conserving` :=

\[\forall j \forall t,\]

`backlogged job_cost sched j t →`\n
\[\forall \text{cpu}, \exists j_{other},\]

`scheduled_on sched j_{other} cpu t.`
Definition work_conserving ::= 
\( \forall j \forall t, \) 
\( \text{backlogged \ job\_cost \ sched \ j \ t} \rightarrow \) 
\( \forall \text{cpu}, \exists \ j\_other, \) 
\( \text{scheduled\_on \ sched \ j\_other \ cpu \ t}. \)
A scheduler is work-conserving iff...

\[
\text{Definition work\_conserving} := \forall j \forall t, \\
\text{backlogged job\_cost sched j t } \rightarrow \\
\forall \text{ cpu, } \exists \text{ j\_other, } \\
\text{scheduled\_on sched j\_other cpu t}.
\]
A scheduler is work-conserving iff...

...for every job $j$ and time $t$...

Definition $work\_conserving :=$
$\forall j \forall t,$
  $\text{backlogged job\_cost sched j t} \rightarrow$
  $\forall \text{cpu}, \exists \text{ j\_other},$
  $\text{scheduled\_on sched j\_other cpu t}.$
A scheduler is work-conserving iff...

...for every job \( j \) and time \( t \)...

...if job \( j \) is backlogged at time \( t \), ...
A scheduler is work-conserving iff...

...for every job $j$ and time $t$...

...if job $j$ is backlogged at time $t$, ...

...then every processor $cpu$ has a job $j_{\text{other}}$...
A scheduler is work-conserving iff...

...for every job $j$ and time $t$...

...if job $j$ is backlogged at time $t$, ...

...then every processor $cpu$ has a job $j\text{\_other}$...

...that is scheduled on $cpu$ at time $t$. 

**Definition** work\_conserving :=
\[ \forall j \forall t, \]
\[ \text{backlogged \ job\_cost \ sched \ j \ t} \rightarrow \]
\[ \forall cpu, \exists j\_other, \]
\[ \text{scheduled\_on \ sched \ j\_other \ cpu \ t}. \]
Principle 2: Some proofs are more important than others

To make progress, we should focus on practical results.
Principle 2: Some proofs are more important than others

To make progress, we should focus on practical results.

We should formalize recent analyses and move towards multiprocessor scheduling.

Critical results should be proven first. E.g., proving analysis safety is more important than termination, time complexity or optimality.
Principle 3: Maintain the proof culture

To ensure accessibility, we should reuse the established proof style of the real-time systems community.
Principle 3: Maintain the proof culture

To ensure accessibility, we should reuse the established proof style of the real-time systems community.

We avoid complex logics (e.g., temporal operators) and advanced constructs from the proof assistant (e.g., records, canonical structures, etc.).

We favor instead first-order logic, lists, functions, basic arithmetic.
Unusual notation discourages adoption

**EDF Optimality in PPTL [Zhang, 2014]**

**Lemma 6.** If $P_i$ overflows at $t = kT_i$, there is no idle time unit in $[(k - 1)T_i, kT_i]$

That is,

$$\text{Sch} \supset \left( \bigcap_{t}^{kT_i} (\forall i \leq C_i) \rightarrow \bigcirc \left( \bigvee_{j=1}^{m} r_j = 1 \right) \right) \quad (k - 1)T_i < t < kT_i.$$ 

**Prosa — Definition of Instantaneous Service**

Definition `service_at (t: time) := \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.`
LaTeX-like operators improve readability

**Instantaneous Service**

Definition $service\_at\ (t: \ time) := \sum_\{cpu < num\_cpus \mid scheduled\_on\ j\ cpu\ t\} 1.$
LaTeX-like operators improve readability

Instantaneous Service

Definition \( \text{service\_at} (t: \text{time}) := \sum_{(\text{cpu} < \text{num\_cpus} \mid \text{scheduled\_on} \ j \ \text{cpu} \ t)} 1. \)

Sum over each processor...
**Instantaneous Service**

Definition `service_at (t: time) := \sum_(cpu < num_cpus | scheduled_on j cpu t) 1.`

Sum over each processor...

...where job j is scheduled...
Definition $service_{at} \ (t: \ time) := \sum_{(cpu < num\_cpus \ | \ scheduled\_on \ j \ cpu \ t)} 1.$

Sum over each processor...

...where job j is scheduled...

...of 1 (i.e., a count).
Principle 4: Community involvement

Vision: shared repository of real-time scheduling concepts and proofs.
Principle 4: Community involvement

Vision: shared repository of real-time scheduling concepts and proofs.

We encourage participation by the community:

Specification accepted by the community + Mechanized Proofs = Non-disputable Results

Check out our website: prosa.mpi-sws.org
Mechanized proofs can reach the community at large

1. Readability is crucial
2. Some proofs are more important than others
3. We should maintain the proof culture
4. Community involvement

By focusing on **readability** and by **maintaining the established proof culture**, mechanized proofs can reach the community at large.
Outline of the Talk

Why mechanized proofs?

Challenges & Principles

A Taste of Prosa
Prosa is a collection of *definitions, assumptions and theorems*

**Definitions**

**Assumptions**

**Theorems**
Prosa covers many concepts from real-time scheduling

Definitions

Library schedule: instantaneous service, cumulative service, job is pending, job is complete...

Library interference: total interference, per-task interference...

Library platform: work conservation, priority enforcement...

Assumptions

Theorems
Assumptions can be easily checked
(~10–15 in each analysis)

Definitions

Assumptions

Hypothesis $H_{\text{completed\_jobs\_dont\_execute}}$:
\[
\text{completed\_jobs\_dont\_execute} \text{ job\_cost \ sched.}
\]

Hypothesis $H_{\text{enforces\_FP\_policy}}$:
\[
\text{enforces\_FP\_policy} \text{ job\_cost job\_task sched higher\_priority.}
\]

Hypothesis $H_{\text{work\_conserving}}$:
\[
\text{work\_conserving} \text{ job\_cost sched.}
\]

Hypothesis $H_{\text{sequential\_jobs}}$:
\[
\text{sequential\_jobs} \text{ sched.}
\]

Theorems

[...]

74
Assumptions can be easily checked (~10–15 in each analysis)

Definitions

Assumptions

Hypothesis $H_{\text{completed\_jobs\_dont\_execute}}$:
completed\_jobs\_dont\_execute job\_cost sched.

Theorems
Assumptions can be easily checked (~10–15 in each analysis)

**Definitions**

**Assumptions**

Hypothesis \( H_{\text{completed\_jobs\_dont\_execute}}: \)
\[
\text{completed\_jobs\_dont\_execute} \quad \text{job\_cost\_sched}.
\]

In any given schedule and for any given actual job execution costs, ...

**Theorems**
Assumptions can be easily checked (~10–15 in each analysis)

Definitions

Assumptions

**Hypothesis H\_completed\_jobs\_dont\_execute:**

\[ \text{completed\_jobs\_dont\_execute} \text{ job\_cost sched.} \]

In any given schedule and for any given actual job execution costs, ...

...jobs do not execute after completion.

Theorems
Assumptions can be easily checked (~10–15 in each analysis)

Definitions

Assumptions

Hypothesis \( H_{\text{completed_jobs_dont_execute}} \):
\[
\text{completed_jobs_dont_execute \ job_cost \ sched.}
\]

Definition \( \text{completed_jobs_dont_execute} := \)
\[
\forall j \forall t, \\
\text{service \ sched \ j \ t} \leq \text{job_cost \ j}.
\]

Theorems
Assumptions can be easily checked (~10–15 in each analysis)

**Definitions**

**Assumptions**

Hypothesis $H_{\text{completed_jobs_dont_execute}}$:

\[
\text{completed_jobs_dont_execute \ job_cost \ sched.}
\]

**Definition** $\text{completed_jobs_dont_execute} := \\
\forall j \forall t, \\
\text{service sched j t} \leq \text{job_cost} j.
\]

For every job $j$ at any time $t$,

**Theorems**
Assumptions can be easily checked (~10–15 in each analysis)

Definitions

Assumptions

Hypothesis $H_{\text{completed\_jobs\_dont\_execute}}:$

\[ \text{completed\_jobs\_dont\_execute \ job\_cost \ sched.} \]

Definition $\text{completed\_jobs\_dont\_execute} :=$

\[ \forall j \forall t, \quad \text{service sched j t \leq job cost j}. \]

For every job $j$ at any time $t$,

...the service received by $j$ is no larger than its cost.

Theorems
Theorems are proven in small steps using lemmas

Definitions
Assumptions
Theorems

Theorem workload_bounded_by_W :
workload_of tsk t1 (t1 + delta) ≤ workload_bound.
Theorems are proven in small steps using lemmas

Definitions

Assumptions

Theorems

We upper-bound the workload of a task...

**Theorem workload bounded by \( \bar{W} \):**

\[
\text{workload}_\text{of} \ tsk \ t1 \ (t1 + \delta) \leq \text{workload}_\text{bound}.
\]
Theorems are proven in small steps using lemmas

Definitions

Assumptions

Theorems

Lemma  \text{workload\_bound\_many\_periods\_in\_between} :
\text{job\_arrival} j_{\text{lst}} - \text{job\_arrival} j_{\text{fst}} t \geq \text{num\_mid\_jobs}.+1 
\times (\text{task\_period tsk}).

We upper-bound the workload of a task...

...based on the minimum distance between its first and last jobs in the interval.

Theorem  \text{workload\_bounded\_by\_W} :
\text{workload\_of tsk} t1 (t1 + \text{delta}) \leq \text{workload\_bound}.
Prosa covers many concepts and is well-documented

We use short, easy-to-understand definitions.

1 comment for every 2 lines of spec!
What we have proven so far
(in ~8 person months)
What we have proven so far

(in ~8 person months)

- Sporadic Task Model
  - Workload-based interference bounds for work-conserving and EDF schedulers
What we have proven so far

(in ~8 person months)

• **Sporadic Task Model**

  • *Workload-based interference bounds* for work-conserving and EDF schedulers
  
  • Definition and proofs of correctness and termination of Bertogna and Cirinei’s RTA for FP scheduling

    ➔ Same for Bertogna and Cirinei’s RTA for EDF scheduling
What we have proven so far
(in ~8 person months)

• Sporadic Task Model
  • Workload-based interference bounds for work-conserving and EDF schedulers
  • Definition and proofs of correctness and termination of Bertogna and Cirinei’s RTA for FP scheduling
    ➔ Same for Bertogna and Cirinei’s RTA for EDF scheduling
  • Implementation of a work-conserving scheduler to test for contradictory assumptions
What we have proven so far
(in ~8 person months)

• **Sporadic Task Model**
  - **Workload-based interference bounds** for work-conserving and EDF schedulers
  - Definition and proofs of correctness and termination of Bertogna and Cirinei’s RTA for FP scheduling
    - Same for Bertogna and Cirinei’s RTA for EDF scheduling
  - **Implementation of a work-conserving scheduler** to test for contradictory assumptions

• **Extensions**
  - Same definitions and proofs for workloads with release jitter
  - Same definitions and proofs for workloads with parallel jobs

**novel results**
What we have proven so far
(in ~8 person months)

• Sporadic Task Model
• Workload-based interference bounds for work-conserving and EDF schedulers
• Definition and proofs of correctness and termination of Bertogna and Cirinei’s RTA for FP scheduling
  ➡ Same for Bertogna and Cirinei’s RTA for EDF scheduling
• Implementation of a work-conserving scheduler to test for contradictory assumptions

Thanks to mature proof assistants and libraries, Prosa allows mechanizing recent and complex schedulability analyses in reasonable time.

• Extensions
  ➡ Same definitions and proofs for workloads with release jitter
  ➡ Same definitions and proofs for workloads with parallel jobs
Future Work

1. Correct recently refuted proofs
   a) APA scheduling
   b) Self-suspending tasks

2. Verify practical results
   a) Semi-partitioned scheduling (e.g. C=D)
   b) Blocking analysis
   c) Overhead accounting

3. Investigate how to integrate Prosa with analysis tools and scheduler implementations (done! see prosa.mpi-sws.org/apa)
Future Work

1. Correct recently refuted proofs
   a) APA scheduling  (done! see prosa.mpi-sws.org/apa)
   b) Self-suspending tasks
Future Work

1. Correct recently refuted proofs
   a) APA-scheduling  
      (done! see prosa.mpi-sws.org/apa)
   b) Self-suspending tasks

2. Verify practical results
   a) Semi-partitioned scheduling (e.g. C=D)
   b) Blocking analysis
   c) Overhead accounting
Future Work

1. Correct recently refuted proofs
   a) APA-scheduling   (done! see prosa.mpi-sws.org/apa)
   b) Self-suspending tasks

2. Verify practical results
   a) Semi-partitioned scheduling (e.g. C=D)
   b) Blocking analysis
   c) Overhead accounting

3. Investigate how to integrate Prosa with analysis tools and scheduler implementations
Disclaimer
Disclaimer

Not every proof has to be formalized.
Disclaimer

Not every proof has to be formalized.

Pen-and-paper proofs are still useful.
Not every proof has to be formalized.

Pen-and-paper proofs are still useful.

We aim for readable specifications, but writing formal proofs remains non-trivial.
Mechanized proofs provide an opportunity to avoid the correctness pitfalls in real-time scheduling.

By focusing on readability and by maintaining the established proof culture, mechanized proofs can reach the community at large.

Thanks to mature proof assistants and libraries, Prosa allows mechanizing recent and complex schedulability analyses in reasonable time.
Backup slides
Generality of discrete time

**Theorem 6** A sporadic arbitrary-deadline task system \( T \) is feasible with respect to continuous schedules iff it is feasible with respect to discrete schedules.

Results about dense time could still be formalized with Coq libraries for real numbers, e.g. Coquelicot.

Working with Real Numbers

Coquelicot:
A User-Friendly Library of Real Analysis for Coq

Formalization of limits, continuity, differentiability, Riemann integrals, series, etc.

More info at coquelicot.saclay.inria.fr
Library: Probability Theory

Total/conditional probability, Bayes' theorem, random variables and finite distributions

Lemma prob_decomp: forall A B,
\Pr_d[A] = \Pr_d[A \&: B] + \Pr_d[A \&: \sim B].

Related Work
Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Schedulability condition of RM [Schuzhen et al. 1999]
- Simplified proofs and review [Xu and Zhan 2008]
Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Schedulability condition of RM [Schuzhen et al. 1999]
- Simplified proofs and review [Xu and Zhan 2008]

+ Formalism reduces ambiguity
Formalisms for schedulability analysis

Based on the Duration Calculus (DC) interval logic

- Schedulability condition of RM [Schuzhen et al. 1999]
- Simplified proofs and review [Xu and Zhan 2008]

+ Formalism reduces ambiguity
- Complex logics and manual proofs
- Only uniprocessor scheduling
Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* [Wilding 1998]
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols [Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]
- Schedulability conditions based on **task phase** using *Coq* [De Rauglaudre 2012]
- Certified Computations of **Network Calculus** in Isabelle/HOL [Mabille et al. 2013]
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq [Zhang et al. 2014]
Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* [Wilding 1998]
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols [Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]
- Schedulability conditions based on **task phase** using *Coq* [De Rauglaudre 2012]
- Certified Computations of **Network Calculus** in Isabelle/HOL [Mabille et al. 2013]
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq [Zhang et al. 2014]

+ Mechanically-checked
- No results about multiprocessors
Earlier mechanized proofs

- Proof of **EDF optimality** using *Nqthm* [Wilding 1998]
- Analysis of the **Priority Ceiling** and **Priority Inheritance** Protocols [Zhang et al. 1999] [Dutertre 1999] [Dutertre and Stavridou 2000]
- Schedulability conditions based on **task phase** using *Coq* [De Rauglaudre 2012]
- Certified Computations of **Network Calculus** in Isabelle/HOL [Mabille et al. 2013]
- Implementation and proof of **EDF optimality** with Propositional Projection Temporal Logic (PPTL) in Coq [Zhang et al. 2014]

+ Mechanically-checked
- No results about multiprocessors
- Not widely adopted by our community
Model checking and timed automata

- Analysis of **uniprocessor** FP scheduling using UPPAAL
  [Fersman at al. 2006]

- Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks** using UPPAAL and NuSMV
  [Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]

- Analysis of **sporadic tasks** based on state exploration and automata reachability
  [Baker and Cirinei 2007] [Geeraerts et al. 2012] [Burmyakov et al. 2015] [Sun and Lipari 2015]
Model checking and timed automata

• Analysis of **uniprocessor** FP scheduling using UPPAAL
  [Fersman et al. 2006]

• Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks**
  using UPPAAL and NuSMV
  [Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]

• Analysis of **sporadic tasks** based on state exploration and
  [Burmyakov et al. 2015] [Sun and Lipari 2015]

+ Multiprocessor, exact schedulability analysis
Model checking and timed automata

- Analysis of **uniprocessor** FP scheduling using UPPAAL  
  [Fersman et al. 2006]

- Analysis of **multiprocessor** FP and EDF scheduling of **periodic tasks** using UPPAAL and NuSMV  
  [Guan et al. 2007] [Guan et al. 2008] [Cordovilla et al. 2011]

- Analysis of **sporadic tasks** based on state exploration and automata reachability  
  [Baker and Cirinei 2007] [Geeraerts et al. 2012]  
  [Burmyakov et al. 2015] [Sun and Lipari 2015]

+ Multiprocessor, exact schedulability analysis  
- State explosion (≤ 10 tasks or 4 processors)
Avoiding contradictory assumptions

1. **Implement a scheduler function S** using the proof assistant (take pending jobs, sort by priority, assign to CPUs, …).

2. **Prove that scheduler S satisfies every requirement** of the analysis (work-conserving, enforces priority, etc.) in an assumption-free context.

3. Since S is an actual algorithm, it is **impossible that two contradictory assumptions are satisfied by S.**