



Software Plattform Embedded Systems 2020

GEFÖRDERT VOM



Bundesministerium
für Bildung
und Forschung

D3.3.B – ZP-Subtask3.3.4 Operational Semantics

Philipp Reinkemeier, OFFIS

Friday 1st April, 2011

| | | |
|-------------------|---|---|
| Project name | SPES2020 | |
| Accountability | Philipp Reinkemeier, OFFIS | |
| QS-Accountability | No external reviewer committed, reviewed internally | |
| Created on | 24.01.2011 | |
| Last changed | Friday 1 st April, 2011 | |
| Disclosure | | confidential project open public |
| | ✓ | public |
| Status | | work in progress submitted finished |
| | ✓ | finished |

Additional product information

| | |
|------------------|---|
| Initial creation | Philipp Reinkemeier (PR) |
| Contributed by | Philipp Reinkemeier (PR), Stefan Henkler (SH) |

Changes

| No. | Change | | Changed Chapter(s) | Description | Author | Status |
|-----|------------|---------|--------------------|------------------------------|--------|------------------|
| | Date | Version | | | | |
| 1 | 24.01.2011 | v0.1 | – | Setup document template | PR | work in progress |
| 2 | 31.01.2011 | v0.2 | 1, 2 | Internal review | PR | submitted |
| 3 | 31.03.2011 | v0.3 | 2 | Attached ENVISION 2011 paper | PR | finished |

Contents

- 1 Introduction** **1**
- 1.1 Motivation and Background 1
- 1.2 Management Summary 2

- 2 Operational semantics** **3**

- Bibliography** **4**

- 3 Annex** **5**

1 Introduction

In this chapter a motivation and background information for this work in the scope of the SPES 2020 project will be given, as well as a management summary and an overview of the structure of this document.

1.1 Motivation and Background

In the SPES architecture modeling document [3] an architecture meta-model is proposed. The specification of that meta-model is delivered separately in [5]. The concept of *Heterogeneous Rich Components* (HRC) is the basic foundation of the meta-model. Each HRC may have rigorous interface specifications for multiple *aspects*, which includes functional as well as non-functional aspects like safety and real-time. The specification of an aspect can be done by means of *contracts*. A *contract* is a triple (A_s, A_w, G) , where A_s is the *strong assumption* of a contract, A_w is the *weak assumption* and G its *guarantee*. A *strong assumption* characterizes the allowed design contexts of a component. If a component is used in a legal context, i.e. its environment behaves like assumed, the guarantee characterizes properties of the component. The *weak assumptions* of a contract allow to further structure it according to cases. They can be used to refine the design context by precisizing the strong assumptions and guarantees.

In the SPES architecture modeling document the formal semantics of contracts is defined. Further a pattern-based specification language (RSL) is provided for expressing the assumptions (strong and weak) and guarantee. RSL encompasses patterns supporting multiple aspects, namely *functional behavior*, *real-time* and *safety*. The *realization* of an aspect of a component can be done with any modeling formalism as long as it has a trace-based semantics.

No matter what design process is utilized or which application domain is addressed, there is always a task involved dealing with verification or validation of requirements. That means for an envisioned architectural solution it has to be checked whether all requirements are satisfied. There are many approaches to deal with this development task. For instance code-reviews may unveil parts of a specification not covered by the code. Further applying formal verification by analytical methods or model-checking can prove correctness of a given realization. Another method is to validate systems by simulating them. A simulation is different from verification in that the dynamics of a system are investigated with regard to a set of stimuli in contrast to analyzing its state space.

In order to support formal verification and simulation techniques applied to systems developed accordingly to the concepts explained in [3], this deliverable provides an operational semantics of aspect specifications done by means of contracts. These semantics may then be used to e.g. implement simulation engines and link executable models of aspect realizations with their specifications.

1.2 Management Summary

Work package 3 of ZP3 covers analyses methods and techniques enabling the assessment of architectures with regard to specifications of a set of aspects. The addressed aspects are *functional behavior*, *real-time* and *safety*. How contract-based aspect specifications are integrated in the architecture meta-model is explained in [3]. That document also proposes a pattern-based specification language called RSL that allows to express the assumption and guarantee parts of contracts. The previously enumerated aspects are supported by RSL with dedicated patterns.

This deliverable merely summarizes the relevant parts of [3], which already constitute the operational semantics of RSL patterns. Thus, this document is a supplement to [4] providing operational semantics as described in ZP-Subtask 3.3.4.

2 Operational semantics

In the SPES architecture modeling document [3] the semantics of *contracts* are defined. Basically they characterize sets of traces, which may be defined by logical formulas or other notations such as automata. For the definition of those patterns of the RSL, which support the specification of the *aspects functional behavior* and *real-time*, we used timed automata (first defined in [1]), more precisely a variant called *Timed Safety Automata* (cf. [2]). Each RSL pattern and its defining automaton can be found in Section 6.1 *Syntax and Semantics of RSL* in the SPES Architecture Modeling document [3].

In order to be able to simulate RSL pattern specifications, one needs operational semantics of each RSL pattern. These semantics are directly given by the operational semantics of the timed automata defining each RSL pattern. The operational semantics of that timed automata is a timed transition system as defined in [2].

For flowing interpretations semantics have been defined for real-time and functional patterns. This semantics and a comparison with the *Timing Augmented Description Language* (TADL) has been published in ENVISION 2011 workshop and is attached to this deliverable.

Bibliography

- [1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [2] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer Berlin / Heidelberg, 2004.
- [3] Werner Damm, Hardi Hungar, Stefan Henkler, Ingo Stierand, Bernhard Josko, Philipp Reinkemeier, Andreas Baumgart, Matthias Büker, Tayfun Gezgin, Günter Ehmen, and Raphael Weber. SPES2020 Architecture Modeling. SPES 2020 Deliverable, The SPES 2020 Project, OFFIS, December 2010.
- [4] Eike Thaden. Extended Analysis based on the SPES Architecture Meta-Model. SPES 2020 Deliverable, The SPES 2020 Project, January 2011.
- [5] Raphael Weber, Eike Thaden, and Philipp Reinkemeier. Specification of the SPES Meta-Model. SPES 2020 Deliverable D3.2.B, The SPES 2020 Project, OFFIS, November 2010.

3 Annex

A pattern-based requirement specification language: Mapping automotive specific timing requirements*

Philipp Reinkemeier¹, Ingo Stierand², Philip Rehkop¹, and Stefan Henkler¹

¹philipp.reinkemeier,philip.rehkop,stefan.henkler@offis.de

²stierand@informatik.uni-oldenburg.de

Abstract: Today most requirement specifications are documents written in natural language. Natural language however is ambiguous. Thus computer-aided verification of system-models against such requirement specifications is generally impossible. In this paper we propose a textual requirement specification language (RSL), that is based on patterns, which have a formally defined semantics. RSL is able to express requirements from multiple aspects (e.g. real-time, safety, etc.) on a system. In order to apply RSL in the domain of automotive systems, it has to support timing requirements as the *Timing Augmented Description Language* (TADL). TADL is the coming standard for handling timing information in the automotive domain. In this paper, we present a mapping of TADL constraints to RSL patterns.

1 Introduction

Stating requirements is an essential task in developing products. They build an interface between different groups of people, i.e. customers, engineers, project managers. Typically requirements are stated as natural language text. The disadvantage is that ambiguities can cause a huge amount of requirement changes in all development phases. This leads to higher development costs and may also be unacceptable if the product is deployed in safety critical environments. To avoid ambiguities, often formal languages are used to state requirements with well defined meaning. It however does not only require some training to write requirements in such formal languages, it may also be hard to read them.

The pattern based *Requirement Specification Language* RSL [Par10] bridges the gap by providing a formal language with well defined semantics that is still readable like natural language. RSL patterns consist of static text elements and attributes being instantiated by the requirements engineer. Each pattern has well defined semantics exploitable in verification tools and exhibits a natural language style in order to facilitate consistent interpretation of the written system specification across all project participants. Moreover, the underlying semantics of RSL establishes formal verification in order to ensure correctness of systems.

*This work was supported by the German Research Council (DFG) in SPES 2020 as well as by the ARTEMIS Joint Undertaking CESAR.

Writing requirements shall however be possible in an intuitive way. RSL allows writing of natural sounding requirements while being expressive enough to formalize complex requirements. To gain a high degree of intuition, the language consist of only a few constructs that can be easily remembered, allowing the user to fully understand the RSL and to choose the right pattern that fits the intended system properties. RSL aims at providing expression of a large scale of requirement domains. It defines patterns for each of the following categories:

- *Functional* patterns express requirements such as relationship between events, handling of conditions and invariants, and optional intervals in which a requirement (or parts of it) is valid.
- *Probability* patterns are used to specify the probabilities of failures or hazards occurring in safety critical systems.
- *Safety* related patterns outline relationships between system components. These patterns enable the specification of single point of failures or other failure and hazard dependencies between different components.
- *Timing* patterns are used to describe real-time behavior of systems. This includes recurring activations, jitter and delay.
- *Architectural* patterns specify the existence of architectural elements like components, events and connections between components.

There are already well established requirements specification languages for various aspects and application domains. In order to use RSL as a universal specification language it must be able to cover expressive power of those more specific approaches. In the automotive domain for example, there are mainly two relevant frameworks to specify timing requirements, namely the Timing Augmented Description Language (TADL) [Par09] and the AUTOSAR Timing Extensions (ARTE) [GbR09]. In this paper, we will justify about expressive power of RSL in comparison to TADL.

Though not illustrated in this paper, there already exists a compiler for translating instances of the patterns into observer automata. So verification can be done with state of the art model-checking tools like UPPAAL [BDL04].

Related Work The idea of requirement specification languages featuring a natural language style is not new. For instance in [HJD04] *boilerplates* for requirement specification are proposed. While this approach is also based on templates of natural language, there is not necessarily defined a formal semantics for them. Approaches based on controlled natural languages like *Attempto Controlled English* (ACE), first described in [FS96], or *PENG* [Sch04] are interesting as they come very close to natural language style and can be translated into a first-order logic system. However quantitative specifications like time intervals are inevitably for real-time systems. That cannot be done using such approaches because they usually do not provide a typed system. Graphical specification languages also provide an intuitive way to define requirements. Popular representatives are Live Sequence Charts (LCS) [DH01] and Timing Diagrams [DL02]. The mentioned approaches and formal language style languages like RSL have in common compositional techniques to define automaton semantics important for observer-based verification. RSL itself is

based on the *Contract Specification Language* (CSL) [GBC⁺08] defined in the SPEEDS project. RSL has been extended and evolved in that it is more consistent and modular.

Section 2 introduces the common concepts of the RSL and defines its semantics as far as relevant for this paper. Section 3 revisits the semantics of TADL constraints and discusses their covering by respective RSL patterns. Section 4 concludes the paper.

2 The Requirement Specification Language

The notation of RSL patterns normally contain optional parts that are not necessarily instantiated. A functional pattern for example describing causality between two events looks like this: **whenever** *event*₁ **occurs** *event*₂ **occurs** [**during** *interval*]

Phrases in square brackets are optional, bold elements are static keywords of the pattern, and elements printed in italics represent attributes that have to be instantiated by the requirements engineer. The names of the attributes can be used later to be mapped to a given architecture, or they can be used to generate the architecture.

When specifying requirements there is no strict typing like *int* or *real* but there are categories that specify what is described by the different attributes. To not complicate the language too much and to be also expressive enough, the set of supported attributes has to be carefully chosen:

Events represent activities in the system. An event can be for example a signal, a user input by a pressed button, or a computational result. Events occur at distinct time instants and have no duration.

Conditions are logic and/or arithmetic expressions over variables and the status of events. Conditions can be used in two different ways. Firstly, they can be used as pseudo events to trigger an action if the condition becomes true. And secondly, they may represent the system state that has to hold for a specified time.

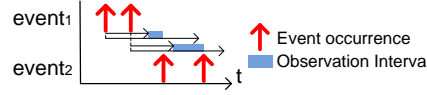
Intervals describe a continuous fragment of time whose boundaries are (relative) time measures, or events. They can be open “[x,y[“, closed “[x,y]” or a combination thereof.

Components refer to entities that are able to handle events or condition variables. Typically, components refer to an architecture.

RSL supports various combinations of elements. It allows for example filtering of events by intervals or conditions. That is, wherever an event occurs in a pattern, one may also use *event* **during** [*interval*], or *event* **under** (*condition*). Thus, one can e.g. specify

whenever *request* **during** [*activate*, *deactivate*] **occurs** *response* **occurs during** [*l*, *u*]
meaning that a response is constrained only if the *request* happened after an *activate* and before some *deactivate* event. Filtering of events can be applied recursively. So *activate* could again be filtered by some interval or condition.

In the following we define syntax and semantics of those RSL patterns that are used later to define the mapping to TADL-constraints. We define semantics in terms of *timed traces*


 Figure 1: Semantics of pattern $F1$

and *timed languages* [AD94]. A *timed trace* over Σ is a pair (σ, τ) , where $\sigma = \sigma_1\sigma_2\cdots$ is an infinite word over Σ , and $\tau = \tau_1\tau_2\cdots$ is a sequence of time instants $\tau_i \in \mathbb{R}$ and $\forall i : \tau_i \leq \tau_{i+1}$, at which the events σ_i occur. A *timed language* is a set of *timed traces* over Σ . The projection of (σ, τ) onto $\Sigma' \subseteq \Sigma$, written $(\sigma, \tau)|_{\Sigma'}$ is formed by deleting all $\sigma_i \notin \Sigma'$ along with their associated time value τ_i from (σ, τ) . Projection is naturally extended to languages over timed traces.

Pattern F1: **whenever** $event_1$ **occurs** $event_2$ **[does not] occur[s] [during interval]**.

The pattern defines that an $event_2$ shall (not) occur whenever an $event_1$ has been occurred in the system. An optional interval confines the instant in time for $event_2$. Thereby *interval* may be defined either by other events like $[startEvent, endEvent]$, or by *timed values* forming the boundaries of an interval like $[0ms, 5ms]$. Timed values can also be negative, thereby allowing "backward specification". For mapping TADL constraints only boundaries in terms of timed values are needed.

Throughout this paper we make use of the following notations. We assume a set of events Σ_{Sys} that can be produced by a given system. The events occurring in an RSL pattern p are denoted Σ_p . For example, for the F1 pattern instance

whenever s **occurs** r **occur during** $[l, u]$.

we get $\Sigma_{F1} = \{s, r\} \subseteq \Sigma_{Sys}$. We further assume l and u to be timed values $l, u \in \mathbb{R}$. For $0 \leq l \leq u$ the semantics of pattern $F1$ is defined by the timed language \mathcal{L}_{F1} over Σ_{F1} :

$$\mathcal{L}_{F1} = \{(\sigma, \tau) | \forall i \in \mathbb{N}, \exists j \in \mathbb{N} : \sigma_i = s \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}$$

The semantics is shown in Figure 1. Multiple activations of the pattern may run in parallel. The respective definition for $u \leq l < 0$ is symmetric.

The semantics of some RSL patterns can be modified by additional attributes. A pattern modified by the attribute **once** gets a strict interpretation of the action during the observed interval. The attribute **once** modifies \mathcal{L}_{F1} by replacing the quantified $\exists j \in \mathbb{N}$ with $\exists! j \in \mathbb{N}$. Note that the attribute **once** can only be used in combination with an interval as otherwise the pattern would not terminate. Thus, the part "during interval" of the pattern becomes required. The optional part **does not** of the pattern modifies \mathcal{L}_{F1} by replacing the quantified $\exists j \in \mathbb{N}$ by $\nexists j \in \mathbb{N}$ in the language definitions above.

Pattern R1: $event$ **occurs sporadic with minperiod** $period_1$ **[and maxperiod** $period_2$ **and jitter** $jitter$].

The pattern defines that $event$ shall occur sporadically, relatively to some reference event which occurs with a minimum inter-arrival time $period_1$. An optional *jitter* specifies the maximum amount of time each occurrence of $event$ can be delayed relatively to the

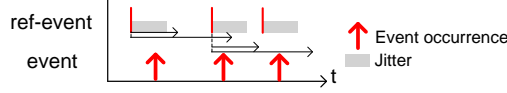
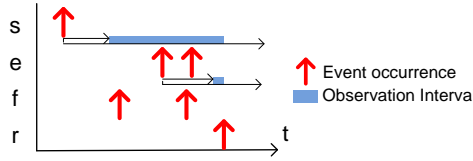

 Figure 2: Semantics of pattern $R1$


Figure 3: Semantics of sequences of events

occurrence of the reference event. If no jitter is defined it is assumed to be 0. The optional $period_2$ bounds the maximum inter-arrival time for subsequent occurrences of the reference event. For the $R1$ pattern instance

e occurs sporadic with minperiod T_1 and maxperiod T_2 and jitter J .

we have $\Sigma_{R1} = \{e\} \subseteq \Sigma_{Sys}$, and $T_1, T_2, J \in \mathbb{R}, 0 \leq T_1 \leq T_2, 0 \leq J$ are normalized time values. The language \mathcal{L}_{R1} of the pattern is defined as follows:

$$\mathcal{L}_{R1} = \{(\sigma, \tau) \mid \exists \tau', \forall i \in \mathbb{N} : (\tau'_i \leq \tau_i \leq \tau'_i + J) \wedge (T_1 \leq \tau'_{i+1} - \tau'_i \leq T_2)\}$$

where T_1, T_2 form the lower and upper bound for the distance between subsequent occurrences of the reference event described by the *time sequence* $\tau' = \tau'_1 \tau'_2 \dots$.

Sequences and Sets of Events The RSL language also provides specification of complex event occurrences by means of event sequences, event sets and logical operations on events. Event sequences are specified by an **and then** expression: (**event and then event [during interval]**)

It can be used wherever the syntax allows to specify an event. They can be nested allowing for specification of event sequences longer than two. An optional interval in the expression confines the minimum and maximum distance between the involved events. In order to capture the semantics of event sequences, we denote aT the sequence of symbols specified by an **and then**-expression, and aT_i denotes the i th event in the sequence. In the following we restrict to the case where only the last event may be bounded by an additional interval. For example, the semantics of the $F1$ pattern instantiation

whenever s occurs (e and then f and then r during $[l', u']$) occur during $[l, u]$.

is depicted in Figure 3. Here, $\Sigma_{F1} = \{s, e, f, r\} \subseteq \Sigma_{Sys}$ denotes the set of events of the pattern, $l, u, l', u' \in \mathbb{R}$, and $aT = efr$. The language $\mathcal{L}_{F1_{aT}}$ is defined as follows:

$$\mathcal{L}_{F1_{aT}} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists \varrho = \varrho_1 \dots \varrho_n, n = |aT| : \sigma_i = s \Rightarrow (i < \varrho_1) \wedge (\forall 1 \leq k \leq n : (\tau_i + l \leq \tau_{\varrho_k} \leq \tau_i + u) \wedge (\sigma_{\varrho_k} = aT_k) \wedge (\tau_{\varrho_1} + l' \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + u'))\}$$

Similarly to \mathcal{L}_{F1} the attribute **once** may be applied or the optional part **does not** may be used, which replaces \exists_Q by $\exists!_Q$, respectively \nexists_Q .

Additionally, RSL provides a shortcut for the specification of N subsequent occurrences of the same event. The syntax is: N **times** *event*.

Further it is sometimes useful to define sets of events where ordering does not matter. Arbitrary ordering of a set $\{a, b, c\}$ of events can thus be expressed as

(*a and then b and then c during* $[l', u']$) **OR** (*a and then c and then b during* $[l', u']$)
OR (*b and then a and then c during* $[l', u']$) **OR** ...

As an abbreviation RSL provides the construct **set** $\{event, \dots, event\}$ [**during** *interval*].

Satisfaction-relation While each RSL pattern instantiation defines a distinct language, we have to reason about satisfaction of such constraint by a given system. So we assume system behaviour also be defined in terms of languages over timed traces. That is, a system S is defined by a language \mathcal{L}_{Sys} over the alphabet Σ_{Sys} . S satisfies an RSL constraint p over an alphabet Σ_p if and only if $\mathcal{L}_{Sys|\Sigma_p} \subseteq \mathcal{L}_p$.

For multiple RSL-patterns $p_1 \dots p_n$, each of which forms a *timed language* \mathcal{L}_{p_i} over an alphabet Σ_{p_i} , the system obviously must satisfy all those constraints. This is equivalent to constructing the parallel composition of the constraints $\parallel_i p_i$ and then to check the system against it. The language of the parallel composition however is simply defined as follows:

$$\mathcal{L}_{\parallel} = \{(\sigma, \tau) \mid \forall i : (\sigma, \tau)_{|\Sigma_{p_i}} \in \mathcal{L}_{p_i}\}$$

If all alphabets Σ_{p_i} are the same, then \mathcal{L}_{\parallel} is simply the intersection of all \mathcal{L}_{p_i} .

3 Mapping of TADL Constraints

The TADL semantics [Par09] of constraints defines how assumptions about occurrences of *events* must relate to each other for a set of timing constraints to be satisfied. That is the semantics of a constraint is given by its satisfaction-relation. The semantics of an *event* is the same as in RSL. For the occurrence of an event e TADL defines a function $dynamics(e) = \langle t_1, t_2, \dots \rangle$, which associates that event with a list of strictly increasing time values. Thus it is a projection $(\sigma, \tau)_{|\{e\}}$ followed by a projection on the second component, where $\{e\} \subseteq \Sigma_{Sys}$. What remains is a strictly increasing *time sequence* $\tau = \tau_1 \tau_2 \dots$ denoting the times of occurrences for event e in an observation.

An event chain relates two sets of events, *stimulus* and *response*. There is an intuition behind an event chain. However in [Par09] the semantics are left undefined and it is argued, that arising questions about causality would involve the semantics of the structural model, which was not intended. That means the causality of events is NOT defined by such an event chain.

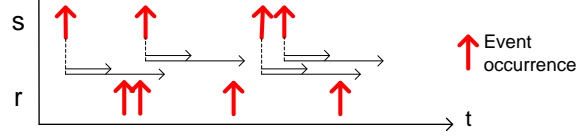


Figure 4: occurrences of s and r and the reference window for reaction constraint

3.1 Delay constraints

Delay constraints are applied on the *stimulus*- and *response*-sets of event chains and describe how the occurrences of events of that sets must relate. The parameters of each kind of delay constraint are given by a subset the following elements: A set S of events acting as stimuli, a set R of events acting as responses, a time offset l indicating the near edge of a time window, a time offset u indicating the far edge, and the size w of a sliding window.

Reaction constraint From [Par09]: A reaction time constraint c is satisfied for some given event behaviour if and only if for all events s in S , for all times t in $dynamics(s)$, for all events r in R , there exists at least one time v in $dynamics(r)$ such that $t+l \leq v \leq t+u$.

Using the previously defined interpretation of the function $dynamics(e)$, we can derive a timed language characterizing a reaction constraint. A reaction constraint forms a timed language \mathcal{L}_{react} over an alphabet $\Sigma_{react} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{react} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists j \in \mathbb{N} : (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}$$

Characterizing the timed language of each TADL constraint is straight forward. For the remaining constraints we will thus not cite the TADL semantics and directly give the timed language. The semantics of a reaction constraint \mathcal{L}_{react} can be expressed in RSL by a $F1$ pattern for each event $r \in \{r_1, \dots, r_n\} = R$. The lower and upper bound l and u , respectively, are the same for all these $F1$ patterns. Thus, $\forall r \in R$:

whenever s_1 OR \dots OR s_m occurs r occur during $[l, u]$.

Proposition 1 $\mathcal{L}_{react} = \parallel_{r \in R} \mathcal{L}_{F1_r}$

Output Synchronization constraint An output synchronization constraint is a reaction constraint extended by the additional parameter w . It forms a *timed language* \mathcal{L}_{output} over an alphabet $\Sigma_{output} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{output} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \exists x \in \mathbb{R}, \forall r \in R, \exists! j \in \mathbb{N} : (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r) \wedge (x \leq \tau_j \leq x + w)\}$$

The semantics of an output synchronization constraint \mathcal{L}_{output} can be expressed in RSL by a $F1$ pattern for each event $r \in \{r_1, \dots, r_n\} = R$. The lower and upper bound l and u , respectively, are the same for all these $F1$ patterns. Thus, $\forall r \in R$:

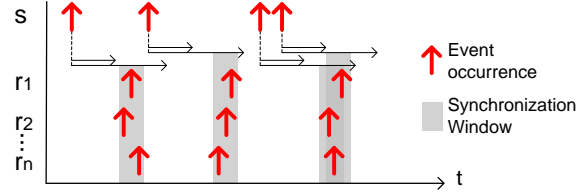
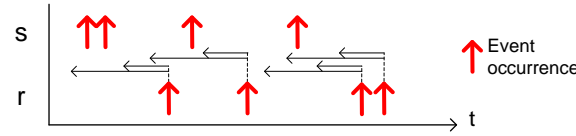


Figure 5: output synchronization constraint and the reference- and sliding-window


 Figure 6: occurrences of s and r and the reference window for age constraint

whenever s_1 OR \dots OR s_m occurs r occur during $[l, u]$, once.

Finally another $F1$ pattern of the following definition must be instantiated:

whenever s_1 OR \dots OR s_m occurs set $\{r_1, \dots, r_n\}$ during $[0, w]$ occur during $[l, u]$.

Proposition 2 $\mathcal{L}_{output} = (\parallel_{r \in R} \mathcal{L}_{F1_{once_r}}) \parallel \mathcal{L}_{F1_{set_w}}$

Age constraint An age constraint forms a *timed language* \mathcal{L}_{age} over an alphabet $\Sigma_{age} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{age} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists j \in \mathbb{N} : (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\}$$

The semantics of an age constraint \mathcal{L}_{age} can be expressed in RSL by a $F1$ pattern for each event $s \in \{s_1, \dots, s_n\} = S$. The lower and upper bound l and u , respectively, are the same for all these $F1$ patterns. Thus, $\forall s \in S$:

whenever r_1 OR \dots OR r_m occurs s occur during $[-u, -l]$.

Proposition 3 $\mathcal{L}_{age} = \parallel_{s \in S} \mathcal{L}_{F1_s}$

Input Synchronization constraint An input synchronization constraint is an age constraint extended by the additional parameter w . It forms a *timed language* \mathcal{L}_{input} over an alphabet $\Sigma_{input} = \{S \cup R\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{input} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \exists x \in \mathbb{R}, \forall s \in S, \exists j \in \mathbb{N} : (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s) \wedge (x \leq \tau_j \leq x + w)\}$$

The semantics of an input synchronization constraint \mathcal{L}_{input} can be expressed in RSL by a $F1$ pattern for each event $s \in \{s_1, \dots, s_n\} = S$. The lower and upper bound l and u , respectively, are the same for all these $F1$ patterns. Thus, $\forall s \in S$:

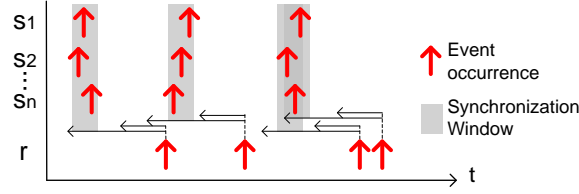


Figure 7: input synchronization constraint and the reference- and sliding-window

whenever r_1 OR ... OR r_m occurs s occur during $[-u, -l]$, once.

Finally another $F1$ pattern of the following definition must be instantiated:

whenever r_1 OR ... OR r_m occurs set $\{s_1, \dots, s_n\}$ during $[0, w]$ occur during $[-u, -l]$.

Proposition 4 $\mathcal{L}_{input} = (\|_{s \in S} \mathcal{L}_{F1_{once_s}}) \| \mathcal{L}_{F1_{set_w}}$

3.2 Repetition rate constraints

Repetition rate constraints place restrictions on the distribution of the occurrences of a single event, forming a so called event model. TADL includes four kinds of repetition rate constraints, namely: *periodic*, *sporadic*, *pattern* and *arbitrary*. An additional *generic repetition rate constraint* is defined, that constitutes a semantic foundation for those repetition rate constraints. However it can **NOT** be instantiated directly.

Generic repetition rate constraint A generic repetition rate constraint is parametrized by the following elements: The event e , whose occurrences are constrained, a lower bound l on the time-distance between occurrences of e , an upper bound u on the time-distance between occurrences of e , the deviation J from an ideal point in time, at which the event e is expected to occur and the actual time it occurs and a count SP indicating whether it is subsequent occurrences or occurrences farther apart that are constrained.

A generic repetition rate constraint forms a *timed language* $\mathcal{L}_{generic}$ over an alphabet $\Sigma_{generic} = \{e\} \subseteq \Sigma_{Sys}$, defined as

$$\mathcal{L}_{generic} = \{(\sigma, \tau) | \exists \tau', \forall i \in \mathbb{N} : (\tau'_i \leq \tau_i \leq \tau'_i + J) \wedge (l \leq \tau'_{i+SP} - \tau'_i \leq u)\}$$

where $\tau' = \tau'_1 \tau'_2 \dots$ is a *time sequence* of ideal times of occurrences for event e .

The semantics of the four kinds of repetition rate constraints are defined in terms of multiple *generic repetition rate constraints* and *reaction time constraints*. The definition of *periodic*, *sporadic* and *pattern* are based on generic repetition rate constraints with $SP = 1$. That semantics can be expressed in RSL by instantiating the following $R1$ pattern:

e occurs sporadic with minperiod l and maxperiod u and jitter J .

If we have a generic repetition rate constraint c , with $SP = 1$, $\mathcal{L}_c = \mathcal{L}_{R1}$ is given by the definitions of both languages.

The semantics of an *arbitrary repetition constraint* is based on generic repetition rate constraints with $SP \geq 1$ and $J = 0$. It can be expressed by the following $F1$ patterns:

whenever e occurs SP times e occur during $[0, u]$, once.
whenever e occurs SP times e does not occur during $[0, l]$.

Proposition 5 For a generic repetition rate constraint c with $\text{jitter}(c) = 0$: $\mathcal{L}_c = \mathcal{L}_{F1_{aT,once}} \parallel \mathcal{L}_{F1_{aT,neq}}$

4 Conclusion

We presented a pattern-based textual requirements specification language (RSL), that exhibits a natural language style. The RSL allows writing of natural sounding requirements while providing a well defined formal semantics. Thus RSL can narrow the gap from informal requirement specification documents to formal specifications and verification techniques. The language provides patterns for various aspects like safety, functional and timing requirements. With regard to timing aspects we showed that the expressiveness of RSL is powerful enough to cover all constraints of the Timing Augmented Description Language (TADL). This enables using RSL to express a relevant set of timing requirements as defined in a well established specification language, while maintaining convenience and intuitiveness of natural language.

References

- [AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comp. Sc.*, 126(2), 1994.
- [BDL04] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on Uppaal. In *LNCS, Formal Methods for the Design of Real-Time Systems*, pages 200–237. Springer, 2004.
- [DH01] Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, July 2001.
- [DL02] H. Dierks and M. Lettrari. Constructing Test Automata from Graphical Real-Time Requirements. In *Proc. FTRTFT '02*, 2002.
- [FS96] N. E. Fuchs and R. Schwitter. Attempto Controlled English (ACE). In *CLAW '96: First International Workshop on Controlled Language Applications*, pages 124–136, 1996.
- [GBC⁺08] V. Gafni, A. Benveniste, B. Caillaud, S. Graph, and B. Josko. Contract Specification Language (CSL). Technical report, SPEEDS Consortium, April 2008.
- [Gbr09] AUTOSAR GbR. *Specification of Timing Extensions*, r4.0 rev1 edition, November 2009.
- [HJD04] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2 edition, 2004.
- [Par09] Project TIMMO: TIMMO Partners. TADL: Timing Augmented Description Language version 2. Deliverable d6, The TIMMO Consortium, October 2009.
- [Par10] Project CESAR: CESAR Partners. *RSL Reference Manual*. CESAR Consortium, 1.1 edition, 2010. Not publically available yet.
- [Sch04] R. Schwitter. Representing Knowledge in Controlled Natural Language: A Case Study. In *Knowledge-Based Intelligent Information and Engineering Systems*, LNCS. 2004.

5 Proofs

5.1 Proof of Proposition 1

$$\begin{aligned}\mathcal{L}_{react} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\} \\ \Sigma_{react} &= \{S \cup R\}\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{F1_r} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists j \in \mathbb{N} : \\ &\quad ((\sigma_i = s_1) \vee \dots \vee (\sigma_i = s_m)) \Rightarrow \\ &\quad (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\} \\ \Sigma_{F1_r} &= \{S \cup r\}\end{aligned}$$

with $((\sigma_i = s_1) \vee \dots \vee (\sigma_i = s_m)) \Leftrightarrow s \in \{s_1, \dots, s_m\} : (\sigma_i = s)$ we get

$$\begin{aligned}\mathcal{L}_{F1_r} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}\end{aligned}$$

The parallel composition $\parallel_{r \in R} \mathcal{L}_{F1_r}$ yields

$$\begin{aligned}\Sigma_{\parallel} &= \bigcup_{r \in R} \Sigma_{F1_r} = \{S \cup R\} = \Sigma_{react} \\ \mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall r \in R : (\sigma, \tau)_{\{S \cup r\}} \in \mathcal{L}_{F1_r}\} \\ &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\} \\ &= \mathcal{L}_{react}\end{aligned}$$

□

5.2 Proof of Proposition 2

$$\begin{aligned}\mathcal{L}_{output} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \exists x \in \mathbb{R}, \forall r \in R, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r) \\ &\quad \wedge (x \leq \tau_j \leq x + w)\} \\ \Sigma_{output} &= \{S \cup R\}\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{F1_{once_r}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists! j \in \mathbb{N} : \\ &\quad ((\sigma_i = s_1) \vee \dots \vee (\sigma_i = s_m)) \Rightarrow \\ &\quad (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\} \\ \Sigma_{F1_{once_r}} &= \{S \cup r\}\end{aligned}$$

Analog to the proof in Section 5.1 we get

$$\begin{aligned}\Sigma_{\parallel_1} &= \bigcup_{r \in R} \Sigma_{F1_{once_r}} = \{S \cup R\} = \Sigma_{output} \\ \mathcal{L}_{\parallel_1} &= \{(\sigma, \tau) \mid \forall r \in R : (\sigma, \tau)_{\{S \cup r\}} \in \mathcal{L}_{F1_{once_r}}\} \\ &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)\}\end{aligned}$$

In Section 2 we say that the semantics of sets of events are derived from that of sequences of events. By *set* we denote the set of symbols specified by some **set**-expression. $\pi(set)$ is then the set of all possible sequences of the symbols contained in *set*, i.e. the permutations of the symbols. For the *F1* pattern instance used to map the output synchronization constraint we have $set = R = \{r_1, \dots, r_n\}$. Thus:

$$\begin{aligned}\mathcal{L}_{F1_{set_w}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \exists \varrho = \varrho_1 \dots \varrho_n, n = |R| : \\ &\quad (\sigma_i = s) \Rightarrow (\exists aT \in \pi(set) : (i < \varrho_1) \wedge (\forall k \in \{1, \dots, n\} : \\ &\quad (\tau_i + l \leq \tau_{\varrho_k} \leq \tau_i + u) \wedge (\sigma_{\varrho_k} = aT_k) \wedge \\ &\quad (\tau_{\varrho_1} \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + w)))\}\end{aligned}$$

For the construction of $\mathcal{L}_{\parallel_1} \parallel \mathcal{L}_{F1_{set_w}}$ we can simply intersect both languages as they are defined over the same alphabet $\{S \cup R\}$ and have

$$\begin{aligned}\mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \forall r \in R, \exists! j \in \mathbb{N}, \exists \varrho = \varrho_1 \dots \varrho_n, n = |R| : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r) \wedge \\ &\quad (\exists aT \in \pi(set) : (i < \varrho_1) \wedge (\forall k \in \{1, \dots, n\} : \\ &\quad (\tau_i + l \leq \tau_{\varrho_k} \leq \tau_i + u) \wedge (\sigma_{\varrho_k} = aT_k) \wedge \\ &\quad (\tau_{\varrho_1} \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + w)))\}\end{aligned}$$

If the term $(j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r)$ is true, then each symbol $r \in R$, resp. $r \in \text{set}$, occurs exactly once in the interval $[\tau_i + l, \tau_i + u]$. So there can only exist one ρ . It exists, if $(\tau_{\rho_1} \leq \tau_{\rho_n} \leq \tau_{\rho_1} + w)$, which requires the first and last symbol of the sequence to be confined to an interval of length w . So there is a point in time $x \in \mathbb{R}$, such that $\forall k \in \{1, \dots, n\} : x \leq \tau_{\rho_k} \leq x + w$. Thus:

$$\begin{aligned} \mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall s \in S, \exists x \in \mathbb{R}, \forall r \in R, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = s) \Rightarrow (j > i) \wedge (\tau_i + l \leq \tau_j \leq \tau_i + u) \wedge (\sigma_j = r) \\ &\quad \wedge (x \leq \tau_j \leq x + w)\} \\ &= \mathcal{L}_{\text{output}} \end{aligned}$$

□

5.3 Proof of Proposition 3

$$\begin{aligned} \mathcal{L}_{\text{age}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\} \\ \Sigma_{\text{age}} &= \{S \cup R\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{F1_s} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists j \in \mathbb{N} : \\ &\quad ((\sigma_i = r_1) \vee \dots \vee (\sigma_i = r_m)) \Rightarrow \\ &\quad (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\} \\ \Sigma_{F1_s} &= \{R \cup s\} \end{aligned}$$

with $((\sigma_i = r_1) \vee \dots \vee (\sigma_i = r_m)) \Leftrightarrow r \in \{r_1, \dots, r_m\} : (\sigma_i = r)$ we get

$$\begin{aligned} \mathcal{L}_{F1_s} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\} \end{aligned}$$

The parallel composition $\parallel_{s \in S} \mathcal{L}_{F1_s}$ yields

$$\begin{aligned} \Sigma_{\parallel} &= \bigcup_{s \in S} \Sigma_{F1_s} = \{S \cup R\} = \Sigma_{\text{age}} \\ \mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall s \in S : (\sigma, \tau) \upharpoonright_{\{R \cup s\}} \in \mathcal{L}_{F1_s}\} \\ &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\} \\ &= \mathcal{L}_{\text{age}} \end{aligned}$$

□

5.4 Proof of Proposition 4

$$\begin{aligned}\mathcal{L}_{input} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \exists x \in \mathbb{R}, \forall s \in S, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s) \\ &\quad \wedge (x \leq \tau_j \leq x + w)\} \\ \Sigma_{input} &= \{S \cup R\}\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{F1_{once_s}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists! j \in \mathbb{N} : \\ &\quad ((\sigma_i = r_1) \vee \dots \vee (\sigma_i = r_m)) \Rightarrow \\ &\quad (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\} \\ \Sigma_{F1_{once_s}} &= \{R \cup s\}\end{aligned}$$

Analog to the proof in Section 5.3 we get

$$\begin{aligned}\Sigma_{\parallel_1} &= \bigcup_{s \in S} \Sigma_{F1_{once_s}} = \{S \cup R\} = \Sigma_{age} \\ \mathcal{L}_{\parallel_1} &= \{(\sigma, \tau) \mid \forall s \in S : (\sigma, \tau)_{\{R \cup s\}} \in \mathcal{L}_{F1_{once_s}}\} \\ &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)\}\end{aligned}$$

In Section 2 we say that the semantics of sets of events are derived from that of sequences of events. By *set* we denote the set of symbols specified by some **set**-expression. $\pi(set)$ is then the set of all possible sequences of the symbols contained in *set*, i.e. the permutations of the symbols. For the *F1* pattern instance used to map the input synchronization constraint we have $set = S = \{s_1, \dots, s_n\}$. Thus:

$$\begin{aligned}\mathcal{L}_{F1_{set_w}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \exists \varrho = \varrho_1 \dots \varrho_n, n = |S| : \\ &\quad (\sigma_i = r) \Rightarrow (\exists aT \in \pi(set) : (i > \varrho_{SP}) \wedge (\forall k \in \{1, \dots, n\} : \\ &\quad (\tau_i - u \leq \tau_{\varrho_k} \leq \tau_i - l) \wedge (\sigma_{\varrho_k} = aT_k) \wedge \\ &\quad (\tau_{\varrho_1} \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + w)))\}\end{aligned}$$

For the construction of $\mathcal{L}_{\parallel_1} \parallel \mathcal{L}_{F1_{set_w}}$ we can simply intersect both languages as they are defined over the same alphabet $\{S \cup R\}$ and have

$$\begin{aligned}\mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \forall s \in S, \exists! j \in \mathbb{N}, \exists \varrho = \varrho_1 \dots \varrho_n, n = |S| : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s) \wedge \\ &\quad (\exists aT \in \pi(set) : (i > \varrho_{SP}) \wedge (\forall k \in \{1, \dots, n\} : \\ &\quad (\tau_i - u \leq \tau_{\varrho_k} \leq \tau_i - l) \wedge (\sigma_{\varrho_k} = aT_k) \wedge \\ &\quad (\tau_{\varrho_1} \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + w)))\}\end{aligned}$$

If the term $(j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s)$ is true, then each symbol $s \in S$, resp. $s \in \text{set}$, occurs exactly once in the interval $[\tau_i - u, \tau_i - l]$. So there can only exist one ϱ . It exists, if $(\tau_{\varrho_1} \leq \tau_{\varrho_n} \leq \tau_{\varrho_1} + w)$, which requires the first and last symbol of the sequence to be confined to an interval of length w . So there is a point in time $x \in \mathbb{R}$, such that $\forall k \in \{1, \dots, n\} : x \leq \tau_{\varrho_k} \leq x + w$. Thus:

$$\begin{aligned} \mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \forall r \in R, \exists x \in \mathbb{R}, \forall s \in S, \exists! j \in \mathbb{N} : \\ &\quad (\sigma_i = r) \Rightarrow (j < i) \wedge (\tau_i - u \leq \tau_j \leq \tau_i - l) \wedge (\sigma_j = s) \\ &\quad \wedge (x \leq \tau_j \leq x + w)\} \\ &= \mathcal{L}_{\text{input}} \end{aligned}$$

□

5.5 Proof of Proposition 5

From $J = 0$ it follows for $\mathcal{L}_{\text{generic}}$ that $\tau_i = \tau'_i$. Thus:

$$\mathcal{L}_{\text{generic}, J=0} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N} : (l \leq \tau_{i+SP} - \tau_i \leq u)\}$$

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{once}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists! \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : (\sigma_i = e) \Rightarrow \\ &\quad (i < \varrho_1) \wedge (\forall k \in \{1, \dots, SP\} : (\tau_i \leq \tau_{\varrho_k} \leq \tau_i + u) \wedge (\sigma_{\varrho_k} = aT_k))\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{neg}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \nexists \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : (\sigma_i = e) \Rightarrow \\ &\quad (i < \varrho_1) \wedge (\forall k \in \{1, \dots, SP\} : (\tau_i \leq \tau_{\varrho_k} < \tau_i + l) \wedge (\sigma_{\varrho_k} = aT_k))\} \end{aligned}$$

It holds that $\Sigma_{F1_{aT}, \text{once}} = \Sigma_{F1_{aT}, \text{neg}} = \{e\}$, which means the terms $(\sigma_i = e)$ and $(\sigma_{\varrho_k} = aT_k)$ are always true. Thus we simplify:

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{once}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists! \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : \\ &\quad (i < \varrho_1) \wedge (\forall k \in \{1, \dots, SP\} : (\tau_i \leq \tau_{\varrho_k} \leq \tau_i + u))\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{neg}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \nexists \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : \\ &\quad (i < \varrho_1) \wedge (\forall k \in \{1, \dots, SP\} : (\tau_i \leq \tau_{\varrho_k} < \tau_i + l))\} \end{aligned}$$

Due to ϱ being strictly increasing and monotonicity of τ , constraining the first and last element of ϱ to be confined to some interval implies the same for all other elements of ϱ . Further from $(i < \varrho_1)$ it follows that $(\tau_i \leq \tau_{\varrho_1})$. Thus we can simplify:

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{once}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \exists! \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : \\ &\quad (i < \varrho_1) \wedge (\tau_{\varrho_{SP}} \leq \tau_i + u)\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{F1_{aT}, \text{neg}} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N}, \nexists \varrho = \varrho_1 \dots \varrho_{SP}, SP = |aT| : \\ &\quad (i < \varrho_1) \wedge (\tau_{\varrho_{SP}} < \tau_i + l)\} \end{aligned}$$

For $\mathcal{L}_{F1_{aT},once}$: Because $\exists! \varrho, \tau$ is monotonic and ϱ strictly increasing, $\varrho_j = i + j$ holds. In particular $\varrho_1 = i + 1$ and $\varrho_{SP} = i + SP$. If $\varrho_{SP} > i + SP$ and $\tau_{\varrho_{SP}} \leq \tau_i + u$ would hold, then there would exist another sequence ϱ' , with $\varrho'_{SP} = i + SP$, which also satisfies $\tau_{\varrho_{SP}} \leq \tau_i + u$. Further $\varrho_{SP} < i + SP$ would be impossible as it requires $\varrho_1 < i + 1$, which contradicts $i < \varrho_1$. Thus we simplify:

$$\mathcal{L}_{F1_{aT},once} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N} : (\tau_{i+SP} \leq \tau_i + u)\}$$

For $\mathcal{L}_{F1_{aT},neg}$: Suppose there would exist some ϱ satisfying $(i < \varrho_1) \wedge (\tau_{\varrho_{SP}} < \tau_i + l)$. For that $\varrho, \varrho_{SP} \geq i + SP$ would have to hold because of $\varrho_1 > i$. What follows is $\tau_{i+SP} < \tau_i + l$. Because there does not exist such a ϱ it follows that $\tau_{i+SP} \geq \tau_i + l$. Hence:

$$\mathcal{L}_{F1_{aT},neg} = \{(\sigma, \tau) \mid \forall i \in \mathbb{N} : (\tau_{i+SP} \geq \tau_i + l)\}$$

So the parallel composition $\mathcal{L}_{F1_{aT},once} \parallel \mathcal{L}_{F1_{aT},neg}$ yields

$$\begin{aligned} \Sigma_{\parallel} &= \Sigma_{F1_{aT},once} \cup \Sigma_{F1_{aT},neg} = \Sigma_{generic_{J=0}} \\ \mathcal{L}_{\parallel} &= \{(\sigma, \tau) \mid \forall i \in \mathbb{N} : (l \leq \tau_{i+SP} - \tau_i \leq u)\} \\ &= \mathcal{L}_{generic_{J=0}} \end{aligned}$$

□