A Simulation Framework for the Validation of Event-B Specifications

Faqing Yang

UNIVERSITÉ DE LORRAINE

École Doctorale IAEM Lorraine
DFD en informatique

Un environnement de simulation pour la validation de spécifications B événementiel

THÈSE

présentée et soutenue publiquement le 29 Novembre 2013

pour l'obtention du

Doctorat de l'Université de Lorraine

(Spécialité : informatique)

par

Faqing YANG

Composition du jury

Rapporteurs :	
Michael LEUSCHEL	Professeur, Université de Düsseldorf, Allemagne
Catherine DUBOIS	Professeur, ENSIIE, Evry
Examinateurs :	
Pascale LE GALL	Professeur, Ecole Centrale, Paris
Stephan MERZ	Directeur de recherche, INRIA, Nancy
Directeurs de thèse :	
Jeanine SOUQUIÈRES	Professeur, Université de Lorraine, Nancy, LORIA
Jean-Pierre JACQUOT	Maître de conférences, Université de Lorraine, Nancy, LORIA

Loria Laboratoire lorrain de recherche en informatique et ses applications

UMR 7503
Laboratoire Lorrain de Recherche en Informatique et ses Applications

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Jeanine Souquières and my co-advisor Dr. Jean-Pierre Jacquot for the continuous support during my Ph.D. study and research, for their patience, motivation, enthusiasm, and immense knowledge. Without their guidance, I cannot imagine to successfully achieve my research goals. I would also like to thank their helps in my life, especially in a foreign country. I would always remember them as the best advisors and the mentors for the lifetime.

Furthermore, I would like to thank the rest of my thesis committee : Prof. Michael Leuschel, Prof. Catherine Dubois, Prof. Pascale Le Gall, and Dr. Stephan Merz, for their participation, insightful comments, and constructive criticism.

In addition, I am indebted to the colleagues from MAIA and TRIO teams at LORIA with special thanks to Dr. Alexis Scheuer for his development of the 2D platooning mathematical model.

I would like to especially dedicate this thesis to my family, for their love, support, patience, and understanding. They allowed me to spend most of the time on this thesis. Last, but certainly not least, I am thankful to my friends, for all the moral encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

1.1.1 Research Context

The classical approaches of developing systems are conducted through activities realized by humans. Unfortunately, humans often make mistakes and mistakes are the most common cause of potential safety-threatening situations. For instance, the first test flight of the Ariane 5 rocket [1] failed on 4 June 1996, because of a malfunction in the control software. A data conversion from a 64-bit floating point value to a 16-bit signed integer value was left unchecked by engineers; it produced an overflow which led to the destruction of the rocket.

The quality of systems can be improved by reducing human mistakes during the development process, e.g., by using formal methods for the whole system or some critical subsystems. Formal methods use mathematical logic to abstractly represent systems, to prove that the formal specifications are consistent with the requirements, to prove that the implementations meet the specification, and to generate code from the specifications.

1. http://en.wikipedia.org/wiki/Ariane_5

In this thesis, a system is said to be correct if it satisfies two conditions:

– *Verified: the system must be internally consistent and it must meet the initial specification, i.e., we are building it right. Verification activities are usually an internal process executed by the developers.*
– *Validated: the system must fulfill the intended purpose of its users, i.e., we are building the right thing. Validation activities require processes which involve people outside developers (stakeholders).*

Using formal methods, getting a verified specification is reasonably easy. However, a verified specification does not automatically result in a validated specification. Verification and validation are independent procedures. They should be used together to assure the correctness of a formal specification. Often, the verification activities are made through mathematical proofs and require significant resources (time, money and experience). This makes formal methods more adapted to the development of safety-critical systems where the cost of faults is very high. For instance, in the railway domain or in the aerospace domain, undetected errors may cause the loss of lives.

This thesis aims at the specification, verification and validation of safety-critical systems with formal methods, in particular, with Event-B. Our research work started from analyzing an existing Event-B specification which formalizes the longitudinal control by a platooning algorithm. The safety property which must hold in the platooning model is the absence of collisions between vehicles. Then we extended this formal specification into two dimensions by adding the lateral control.

1.1.2 Scientific Problem

The most important scientific issue uncovered by case studies used in our research work was:

> *The mathematical proof of a formal specification is not enough to ensure its correctness: verification does not entail validation.*

Proof-based and refinement-based development techniques guarantee that a development is internally consistent, in the formal sense. In particular, those techniques guarantee that all models written during the development meet the initial formal specification. However, non-functional requirements are often very hard to express within standard logics; many requirements are even outside such mathematical frameworks. Hence, they are not in the initial specification. Furthermore, obtaining clear and complete requirements at the early stage of the development is known to be a near-impossible task. Therefore the initial requirements document may be incomplete, ambiguous, inconsistent. As a result, the developers will have to "supply" missing requirements in order to take the necessary decisions during their work. So, the validation activities should be performed on the formal models as early as possible after they have been verified.

1.1.3 Technical Problems

Using Event-B to develop a system raises some technical issues:

– *the absence of practical tools to support validation activities,*
– *the lack of a guideline to integrate formal reasoning and semi-formal reasoning into a development process.*

1.1.4 Objectives

Our work has focused on specific objectives related to the extension of the Event-B usability:

1. The most important objective is about a framework which can simulate the Event-B model for its validation. This framework allows us to guarantee the semantic correctness of simulations. This framework is a complement to existing proving and animation tools that allows to associate validation activities with verifications activities along the chain of refinements.
2. A second objective is to help the verification for the absence of deadlocks in Event-B models.
3. The last objective, more methodological, is to think and enlarge the core notation of refinement used in Event-B. The state-of-the-art refinement emphasizes the verification activities. We complement it with other activities, such as requirements management, adaptation of the mathematical model, checking temporal properties and validation by animation or simulation.

1.2 Contributions

In this thesis, we aim at defining two important contributions.

Contribution I: assessment of Event-B usability

The start point of our research work is to assess the Event-B usability:

– a critical analysis of the original 1D platooning specification and the explanation of some anomalies in the behaviors,
– a Rodin plug-in (about 500 lines in Java) to automatically generate deadlock-freeness theorems,
– an assessment of the scalability of Event-B from different aspects (mathematics, structure, temporal properties and tools).

Contribution II: a JavaScript simulation framework for Event-B

Our key contribution is a simulation framework for Event-B, based on JavaScript. This framework generates simulators from Event-B models and provides a lightweight graphic execution environment. Simulators can be used to validate Event-B models at each refinement step. They help final users, domain experts and developers to better understand the mathematical models and the specifications.

This simulation framework includes:

– an integrated simulation environment for Event-B models, composed of two main elements:
  – a translator (about 2800 lines in Java): a Rodin plug-in which automatically generates simulators from Event-B models, and
  – a runtime environment and most notably a JavaScript library (about 2700 lines in JavaScript) which supports all Event-B mathematical notations,
– a semantics for the correctness of simulations based on the generation of proof obligations.

1.3 Case Studies

We used four case studies in our research work. These case studies allow us to experiment different strategies of translating Event-B models, implementing the simulator and parameterizing the simulations. At present, all refinements of these models can be simulated by our tool.

1D Platooning Model This model [Lanoix 2008] was developed to verify a platooning algorithm in one dimension with Event-B. It contains 10 components and about 600 lines of Event-B formal texts. It can be animated by the existing Event-B animators with some strategies. Hence, we use it as a reference.

2D Platooning Model This model [Yang 2011] is an extension of the first model in 2 dimensions. It contains 10 components and about 1800 lines of Event-B formal texts. It has an abstract carrier set and some uninterpreted functions which caused the failure to the existing Event-B animators. Hence, we use it as a test bed.

Transport-domain Model This model [Mashkoor 2009a] formalizes the transport domain in Event-B. It contains 23 components and about 1100 lines of Event-B formal texts. It uses many Event-B mathematical notations, e.g., abstract carrier sets, set comprehensions, non-deterministic assignments. We use this model to test the interface of our Event-B library and the symbolic executions.

MIDAS Model This model [Wright 2009b, Wright 2010] uses Event-B to construct instruction set architectures. It contains 104 components and about 21800 lines of Event-B formal texts. It has a complex structure of contexts, and the longest refinement chain from the abstract machine composed of forty machines. We use it to test the scalability of JeB for a large project.

1.4 Publications

The obtained results in this thesis have been published in the following papers:

International conferences

[1] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. Proving the Fidelity of Simulations of Event-B Models. In *The 15th IEEE International Symposium on High Assurance Systems Engineering (HASE)*, Miami, USA, forthcoming 2014.

[2] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. JeB: Safe Simulation of Event-B Models in JavaScript. In *The 20th Asia-Pacific Software Engineering Conference (APSEC)*, Bangkok, Thailand, 2013.

[3] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. The Case for Using Simulation to Validate Event-B Specifications. In *The 19th Asia-Pacific Software Engineering Conference (APSEC)*, Hong Kong, China, 2012.

[4] Faqing Yang and Jean-Pierre Jacquot. Scaling Up with Event-B: A Case Study. In Mihaela Bobaru, Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods (NFM)*, volume 6617 of *Lecture Notes in Computer Science*, pages 438–452. Springer Berlin / Heidelberg, 2011.

[5] Faqing Yang and Jean-Pierre Jacquot. An Event-B Plug-in for Creating Deadlock-Freeness Theorems. In *14th Brazilian Symposium on Formal Methods (SBMF)*, São Paulo, Brésil, 2011.

National conferences

[6] Faqing Yang and Jean-Pierre Jacquot. JeB : un environnement de simulation en JavaScript pour B événementiel. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Nancy, France, 2013.

[7] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. Traduction de B événementiel en C pour la validation par la simulation. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Grenoble, France, 2012.

[8] Faqing Yang and Jean-Pierre Jacquot. Prouvé ? Et après ? In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Poitiers, France, 2010.

Article abstract for workshops

[9] Faqing Yang and Jean-Pierre Jacquot. Generating Executable Simulations from Event-B Specifications. In *Rodin User and Developer Workshop*, Fontainebleau, France, 2012.

[10] Faqing Yang and Jean-Pierre Jacquot. An Event-B Plug-in for Creating Deadlock-Freeness Theorems. In *Rodin User and Developer Workshop*, Fontainebleau, France, 2012.

[11] Atif Mashkoor, Faqing Yang and Jean-Pierre Jacquot. Validation of formal specification: The case for animation. In *3rd Workshop on Security and Reliability (SecDay)*, Trier, Germany, 2011.

1.5 Thesis Outline

This thesis is organized as follows

– Chapter 2 describes some related work.

Part I: Assessment of Event-B Usability

– Chapter 3 makes a critical analysis of the 1D platooning specification.
– Chapter 4 describes a tool for generating deadlock-freeness theorems.
– Chapter 5 presents the scalability of Event-B from different aspects.

Part II: JavaScript Simulation Framework for Event-B

– Chapter 6 describes the JeB design philosophy.
– Chapter 7 gives the JeB translator and simulator implementation.
– Chapter 8 illustrates how to realize a simulation and some analysis from a point of view of the validation.
– Chapter 9 defines the notion of correctness of simulations and defines the proof obligations related to that property.

– Chapter 10 gives our conclusion and future work.

Appendices

– Appendix A shortly sums up this thesis in French.
– Appendix B gives the detail translation rules for Event-B formulas into JavaScript.
– Appendix C defines the JavaScript library API for Event-B formulas.
– Appendix D presents the original 1D platooning model in Event-B.
– Appendix E presents the 2D platooning model in Event-B.

Chapter 2

State of the Art

Contents

2.1 Introduction

In this thesis, we consider a software product as an information system. We address the process of developing software based information systems which are composed

not only of software subsystems, but also of hardware subsystems. How to integrate such subsystems and what is the realistic operational environment of a final system are important aspects which should be accounted for at the beginning of a project. The development process is decomposed into a few main activities. The different development methodologies, e.g., linear, iterative, or agile, are defined by different structures and practices of these activities.

Unlike classical development approaches, formal methods use the mathematical logic to rigorously reason about the correctness of a construction. They provide a strong assurance of the absence of bugs in the software. However, they are generally expensive in resources and thus, they are currently reserved for the development of safety-critical systems. Formal methods can be applied at various points during a development process, in particular, at the specification and the verification stages.

Event-B is a formal method for system-level modeling. It is based on the first order logic and set theory. It uses refinements to represent systems at different abstraction levels and uses mathematical proofs to verify the consistency between refinements. In this thesis, we use Event-B to formalize the platooning problem as the ground base for our research.

2.2 Development Process

Computer science and its application quickly evolved in the passed decades. When the first digital computers appeared in the early 1940s, the computer instructions were wired into the machine whose design was not flexible. The computer system architecture then evolved into "hardware" and "software", with the introduction of more and more abstractions to deal with more and more complex computations. Programming languages appeared in the 1950s; this was also a major step to drive the usage of more abstract notations in the software development. The key concept of modularity and information hiding [Parnas 1972] is used to handle the increasing complexity of software systems. In the 1980s, software engineering emphasized the system structure and its management with large complex specifications. In the mid 1990s, the concept of distributed computing gained greater influence as a way to design systems, while the Java programming language [Gosling 2005] was introduced with its own virtual machine; both ideas were another step toward higher abstractions. In 2001, the Agile Manifesto [Beck 2001] introduced the agile software development which is based on iterative and incremental development and encourages rapid and flexible response to change.

The evolution of developing software based systems focused on two aspects: the control of the development process and the control of the software quality. The former resolves the management problems by adopting some development processes, such as the waterfall model or the agile model. The latter resolves the quality problems, such as reliability and security, which can be considerably improved by applying formal methods.

A development process is a set of activities which lead to the production of a software product. However, there is no ideal approach [Brooks 1987] for different types of software products. Some fundamental activities, like requirements analysis, systems/architecture design, subsystem design, coding and testing, depicted in Figure 2.1,

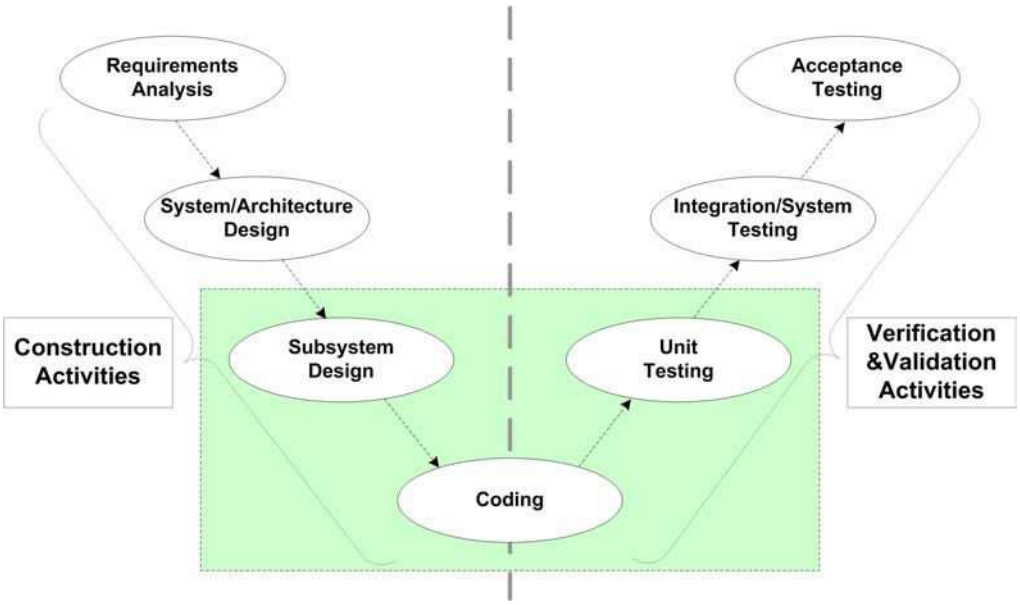

Figure 2.1: Main development activities

are common to all process models. In the following, we give a brief description for each activity by two main catalogs: construction activities, verification and validation activities.

2.2.1 Construction Activities

The aim of construction activities is to produce a system which conforms to the users' requirements. A high quality requirements document [Balzer 1981] is then a key to success. The classical approaches construct systems from the initial requirements document which becomes the reference against which the system is verified and validated in later development steps.

Requirements Analysis This activity collects the requirements of the proposed system by analyzing the needs of the user. It establishes what the expected system has to build, but it does not specify how the system will be designed. Usually, the needs are collected into a document, the "User Requirements Specification," which is the output of this phase.

The user requirements document is used to describe the system properties expected by the user, such as functions, interface, data security, and so on. The plan-driven process models, like the waterfall model, emphasize that the user reviews very carefully this document as it will serve as a guideline for following development activities. The user acceptance testing is carried out according to this document.

System/Architecture Design A system design analyzes and decides the business logic of the proposed system according to the user requirements document. It looks for feasible techniques to implement the user requirements. If some of requirements are not feasible, an alternative solution is proposed to the user and the requirements document is modified accordingly.

The architecture design specifies the architecture of the system and its software part in a high-level abstract formalism. It should consist of a list of subsystems, with precise descriptions of their functionality, their interface and relationships with other subsystems, the dependencies between sub-systems, etc.

Usually a system design document is generated at the end of this activity. It serves as a blueprint for the subsystem design and the coding activities. The integration/system testing is carried out according to this document.

Subsystem Design The subsystem design is a lower-level description of the system. The final system is broken up into smaller subsystems or units. Each of them is described in detail and then can be coded directly.

For each subsystem, a subsystem design document is generated, which contains detailed functional logics, such as database tables, interface details, dependency issues, complete inputs and outputs. The unit testing is carried out according to this document.

Coding This activity results in the executable code for each subsystem. It consists of writing, debugging and testing the source code of a subsystem according to its design document. These codes are often written in one or more programming languages. Some flaws in programs can be more or less easily found by debugging and testing, but certain complex business logics are difficult to be detected in this phase.

2.2.2 Verification and Validation Activities

The verification and validation [Adrion 1982, Andriole 1986] activities are independent procedures. They are used together for checking that (i) a product, service, or system meets its requirements and specifications, and (ii) a product, service, or system meets the intended purpose of the user. These activities are critical elements to achieve a certified software system [Maibaum 2007].

Informally, we express the validation by the question "Are you building the right thing?" and the verification by "Are you building it right?". The IEEE standard [IEEE 2004] give their descriptions as follows:

> The verification process provides objective evidence whether the software and its associated products and processes: (i) conform to requirements for all life cycle activities during each life cycle process, (ii) satisfy standards, practices, and conventions during life cycle processes, (iii) successfully

> complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities.
>
> The validation process provides evidence whether the software and its associated products and processes (i) satisfy system requirements allocated to software at the end of each life cycle activity, (ii) solve the right problem, (iii) satisfy intended use and user needs.

Unit Testing Unit testing aims at verifying the internal logic of the code by exploring every possible branch of the control flow of a subsystem. This testing is usually white-box while the code is directly checked for errors. Static analysis tools are used to facilitate this process without actually executing programs. Usually, the state space explosion prevents these tests to be exhaustive and induces high costs.

Integration/System Testing After the disjoint subsystems are assembled into a single system, the integration testing looks for faults in the interfaces and in the interaction between the integrated subsystems. This testing is usually conducted in a black-box spirit: the source code is not visible to the tester and not directly checked for errors. Once the integration testing is complete, the system testing will be executed to compare the system specifications against the actual system and to check if the integrated system meets the specified requirements.

Acceptance Testing This activity is used to determine whether a system satisfies the requirements specified in the user requirements document. The customer uses the defined acceptance criteria to determine whether to accept the system or not. Once the acceptance testing is complete and its results approved, the developed system will be deployed and delivered to the customer.

2.2.3 Other Activities

Other activities complement the life cycle of a system, such as the project definition and the maintenance. The project definition finds out the organization's objectives, the scope of the problem to resolve, other alternative solutions, the cost and benefits, etc. The maintenance occurs once the system is delivered and operational. The purpose of maintenance is to correct faults, to improve performance and to enhance the system functionality, etc. Usually it has the longest period in the system life cycle.

2.2.4 Some Process Models

A process model is an abstraction of a development process. Several models are widely used in the current software engineering practice. Each one has its advantages and disadvantages. The development team should adopt the most appropriate one for a

project, ideally, within the policies of the organization. Sometimes a combination of these models may be more suitable for a large project.

Waterfall Model The waterfall model [Royce 1970] is one of the first published models of the development process. This model divides a development in sequential phases. The developers follow these phases in order: requirements analysis, system design, implementation, integration, testing, deployment and maintenance.

This model discourages revisiting and revising any prior phase once it's complete. It can be adopted for a project when (1) requirements are very well known, clear and never changed; (2) product definition is stable; (3) the project is short.

V-Model The V-model [Forsberg 1995] is considered as an extension of the waterfall model. The development phases form a typical V shape unlike a linear way in the waterfall model. The V-Model emphasizes the relationships between a phase in the construction and its associated testing phase in the verification and validation. The horizontal axis represents project completeness and the vertical axis represents the abstraction level of the development.

This model greatly emphasizes testing activities, and in particular, the importance of early test planning. It has evolved over time to improve its flexibility and agility for different type of projects.

Spiral Model The spiral model [Boehm 1988] is based on the continuous refinement of the key products realized during requirements definition and analysis, system and software design, and implementation. At each iteration around the cycle, the products are extensions of earlier products. This model emphasizes iterative risk analysis and management. Starting from the center, each turn around the spiral goes through several tasks: 1) determine the objectives; 2) identify and resolve risks; 3) develop and test the product; 4) plan the next iteration.

It is reasonable to use this model in projects where the business goals are unstable but where the architecture must be strong enough to support high load and resist high stress.

Agile Software Development The Agile software development is a philosophy of developing software based on iterative and incremental developments [Larman 2003]. It is not a set of tools or a single methodology. The self-organizing teams and their interactions play an important role in the development process. It emphasizes on rapid iterations, frequent delivery of working software, close collaboration with customers and business experts, quick responses to the requirements change. The face to face communication is considered more effective than written documentation. The working software is considered more useful than presenting documents to clients.

However, agile development is more suitable for small teams. It is often recognized as inefficiency in large organizations and certain types of projects (e.g., mission-critical systems). Large-scale agile software development is still in the field of active research.

2.3 Formal Methods

Formal methods [Wing 1990, Clarke 1996] are mathematical techniques for specification, design and verification of software based systems. They use mathematical logic to reason rigorously about the software construction in order to obtain a reliable and robust system. They enable a good warranty regarding the lack of "bug" in the software. They ensure that the implementation of a software conforms to its specification. However, the application of formal methods is generally costly resources and is currently reserved for developing safety-critical systems. For example, in railway [Butler 2002] and aerospace [Ait Ameur 2010] systems, undetected errors may cause death.

A safety-critical system is that the failure or malfunction of a system can have dramatic consequences, such as death, serious injury, equipment damage and environmental harm. The failure rate is generally allowed to be less than one life per billion (10^9) hours of operations in safety-critical systems. The probabilistic risk assessment, failure mode and effects analysis (FMEA) [Stamatis 2003], fault tree analysis are usually used to evaluate risks and failure analysis.

To develop safety-critical systems, the classical approach is to let humans make the development while applying extra care. But humans easily make mistakes, and these mistakes are the most common cause of safety-threatening errors. Some serious disasters were caused only by a little human mistake. For example, the first test flight of the Ariane 5 rocket on 4 June 1996 failed by a mis-checking on a data conversion.

The quality of safety-critical systems can be improved by using formal methods to develop their critical subsystems [Rushby 1993]. Mathematical proofs ensure that the specifications meet the requirements. Sometimes the executable code can be generated from the formal specifications directly. In such cases, unit testing can be removed from the development process: the correction of the construction is guaranteed by mathematical proofs.

There is often some misunderstanding on the role and application of formal methods [Bowen 1995a, Bowen 1995b, Bowen 2006]. These ideas have overemphasized full formalization of a specification or design. Since a full formalization of a system is a difficult and expensive task, various lightweight formal methods [Jackson 2001, Barner 2005] have been proposed, which emphasize partial specifications and focused applications. Formal methods can be applied at various points through the development process, in particular, at the specification, the verification, and the code generation.

2.3.1 Formal Specification

Formal specifications [Lamsweerde 2000] use mathematical techniques to give a precise and unambiguous description of a system. They use rigorous and effective reasoning tools to describe a system, to analyze its behavior, and to verify interest properties. They help uncover problems and ambiguities in the system requirements. They can be used to guide further development activities.

2.3.2 Formal Verification

Once a formal specification has been developed, the specification should be verified [Bjesse 2005] to prove the correctness of systems. Two main techniques are used to do formal verification: model checking and theorem proving.

Model Checking Model checking is a technique for automatic verification of a given model of a system: whether this model satisfies its specification. It consists of a systematic and, ideally, exhaustive exploration of the space defined by a mathematical model. The specification to be verified is often formulated in terms of temporal logic, such as linear temporal logic or computational tree logic. The advantage of model checking is that it is mostly automatic and easily to product a counter-example; but it often does not scale up well to large systems, especially due to the state explosion problem. Several approaches are used to combat this problem, such as symbolic execution, abstract interpretation, partial order reduction.

Theorem Proving Theorem proving is a deductive method. Firstly a collection of mathematical proof obligations is generated from a system specification. The correctness of a system is then assured by discharging these proof obligations using either theorem provers or SMT solvers. This approach does not suffer the problem of state space explosion and it can be applied to large systems. This approach may require some experiences and skills to manually discharge plenty of proof obligations, in some cases, which is not an easy task.

2.3.3 Code Generation

A formal specification may be used as a guide for the implementation of the system. Refinement based formal methods, like B Method, support gradually add more details about data structures and algorithms to obtain a deterministic version. This deterministic refinement may be directly translate into source code in a target programming language, like [Bert 2003].

2.4 B Method

B method is successfully deployed in many industry projects. This thesis focus on the extension of Event-B usability.

2.4.1 Overview

B is a formal specification language invented by Jean-Raymond Abrial based on the first order logic and set theory. It allows accurately express system properties and prove these properties in a systematic way. Then we can take into account these properties to guide

further development activities. The correctness of construction is fully controlled by the discharge of many mathematical *proof obligations* (PO).

The B method is a mature construction approach based on the B Language. It uses refinements, proof obligation and is fully supported by tools. A development in B starts with writing an abstract model that includes all the requirements: the main data processed by the system is described, as well as the fundamental properties of this data.

The B model thus obtained is a specification of what the system should do. Then the B model is transformed by a succession of refinements into a concrete model that can be coded into a language such as C or Ada. Each refinement can be proven, and so, the development leads to a fault-free software. The B method has been successfully applied in several industrial projects, such as [Behm 1999, Gerhart 1994, Patin 2006].

The B method has two versions: classic B [Abrial 1996] and Event-B [Abrial 2010]. Both languages are based on first order logic and set theory.

Classic B is used in the development of computer software elements, it is supported by tools such as Atelier B [Cle 2009b] or B-Toolkit [B-Core(UK) Ltd 1996]. Atelier B is a commercial tool which includes the ability to translate the refinements into a standard programming language.

Event-B is an evolution of Classic B. It uses only an *event* notation [Abrial 1999] to describe the state transitions; it is supported by the Rodin platform [RODIN 2012]. Compared to Classic B, Event-B has the capacities to model a system.

2.4.2 Presentation of Event-B

Event-B [Abrial 2010] is a formal framework to specify complex systems. It can be analyzed along three axis:

- Description axis: systems are modeled as a state, i.e., a mapping from names to values; they are constrained by an invariant, i.e., a conjunction of predicates on the state, and a collection of events. Events are discrete modifications of the state.
- Semantics axis: it is based on the notion of correctness. A model is correct if it enjoys several demonstrable properties, mainly: well-typedness, existence of actual states and invariant preservation. These properties are expressed as POs that can be automatically generated.
- Development axis: Event-B embed a notion of formal refinement which allows specifiers to use a stepwise development strategy. The correctness of refinements is defined by POs which guarantee that the invariant of the previous machine is preserved by the refinement.

An Event-B model has two kind of components : contexts and machines. Contexts are used to describe the static elements of a system. Machines are used to specify the dynamic behavior of a system. A context may be extended to many other contexts; a context may also extend many other contexts. A machine may be refined to many other machines; but a machine may at most refine one machine in the Rodin platform.

Contexts Contexts describe the static properties of a model. In fact, a given model is parameterized and can be instantiated with associated contexts. A context contains sets, constants, axioms, and theorems. Sets are user-defined types, classified by carrier sets and enumerated sets. Each constant must declare its type in axiom section. The properties of sets and constants are described by axioms. Axioms are considered as hypotheses used in proofs, they are only required be well-defined. So it is possible to introduce a wrong axiom to a model. Axioms can be marked as theorems for derived properties. A theorem must be proved by the axioms written before this theorem; The proved theorems can be used later in proofs just like the other axioms.

Machines Machines describe the dynamic behavior of a model. A machine contains variables, invariants, events, theorems, and variants. Variables constitute the system state, whose values are determined by an initialization event and can be changed by events. Like constants, variables can be any mathematical objects defined in Event-B language, such as integers, sets, relations and functions, etc. Invariants specify the system behavioral properties with variables. They are predicates that should be held for every reachable state. Events are guarded substitutions to change the values of variables. Like axioms, any predicate defined in invariants or event guards can be marked as theorems which should be proved within the machine. A variant is a numeric expression or a finite set to support proofs for event termination.

Event An event represents a transition. It is a guarded substitution. Each event is composed of an arbitrary number of parameters, guards and actions. The guards defined the necessary conditions when an event is enabled to execute. The actions specify how to change the variables' values. They may be deterministic assignments or non-deterministic assignments. When an enabled event is executed, only the variables specified in the actions are changed their values, the other variables keep their old values.

Events have three kind of forms. Let v be a collections of state variables in a machine, x be a collections of parameters of an event, $G(x,v)$ be the guards of an event, $S(x,v)$ be the actions of an event, the general form for an event is:

ANY x **WHERE** $G(x, v)$ **THEN** $S(x, v)$ **END**

The second form is used when there is no parameters:

WHEN $G(v)$ **THEN** $S(v)$ **END**

The last form is used when there is neither guards nor parameters:

BEGIN $S(v)$ **END**

The initialization of the machine is a special event using the last form.

An intuitive operational interpretation of an Event-B model consists in checking if values can be assigned to the parameters to make the guard true for an event, and then to execute the substitution. More generally, execution is a procedure in four steps: (1) to pick (or compute) and assign values to the parameters, (2) to compute the set of enabled events,

(3) to choose one enabled event, and (4) to pick (or compute) and assign values to the substituted variables.

The validation of a model can then be done by observing the evolution of the state's values and the sequences of events fired during executions. Technically, assigning values to parameters, choosing an enabled event to fire and determining the substituted values introduce non-determinism in executions.

Refinement The progress towards implementation is made by following a gradual refinement process. A refinement is the transformation of an abstract model into a more concrete and elaborate model. New variables can be introduced in the state, either as addition or as reification of the abstract variables. This introduction is reflected in the substitutions of the events. A **WITH** clause expresses the link between the parameters of an abstract event, (possibly removed from the refined event) and their concretization.

New events may also be introduced. Except when specified as refining an abstract event, new events are assumed to refine the **SKIP** event. These new events should not prevent forever the old ones from being triggered. A **VARIANT** can be introduced to ensure this property. It consists of a numeric expression or a finite set which must decrease each time a new event is fired. Proof obligations ensure that the refined model is consistent, i.e., its **INVARIANT** is preserved, and the **VARIANT** is decreased by the new events. Furthermore, they ensure that the refinement is correct, i.e., the refined events do not contradict their abstract counterpart.

Semantics The mathematical semantics of Event-B is defined by a set of logical properties of the model, mostly concerned about the possibility to instantiate an actual state and about the preservation of invariants. The syntax of Event-B and the model structure allow tools to automatically generate *Proof Obligations* (POs). A model is *correct,* in the B sense, when all POs have been discharged.

The most important POs can be classified as follows: **WD** (well-definedness), **THM** (provable theorem), **INV** (invariant preservation), **GRD** (guard strengthening), **SIM** (action simulation), **FIS** (feasibility), **VAR** (decreasing of variant).

WD POs ensures that formal predicates, expressions and assignments are indeed well defined. They are mainly a form of type-checking or have special conditions for some particular formulas. Typing in B and Event-B is based on set membership.

THM POs ensures that a theorem declared in a context or a machine is indeed provable. In the Rodin platform, axioms, invariants and guards can be marked as theorems, so they can be proved and used in a proof. The validity of a theorem must be proven from the axioms, invariants or guards declared before this theorem. Theorems are important to simplify some proofs.

FIS and **INV** POs express the logical consistency of the model. The state modifications are expressed as events, which are substitutions. The guards of an event are predicates that identify a set of pre-states. The actions of an event identify a set of post-states. The *Before-After-Predicate* (BAP) of an event relates these pre-states and post-states.

INV proof obligations ensure that post state remains within the legal states (where the invariant holds). **FIS** proof obligations ensure that all the states satisfying the guard are related by the BAP to at least one post-state.

GRD and **SIM** POs express the consistency of refinements. The former set of POs ensure that a refined event cannot be fired when its counterpart in the abstract model could not. The latter set ensures that all substitutions performed in the abstract model are still performed in the refined model.

VAR POs ensures that an event will terminate to execute after a finite number of times. Events can be marked as: ordinary, convergent and anticipated. An ordinary event may occur arbitrary times without the variant constraint. A convergent event must decrease the variant. An anticipated event must not increase the variant.

In practice, many POs may be trivial; current tools discharge them automatically and silently. Users are presented only with the undischarged POs.

Note that POs cannot guarantee that the behavior of the machine is correct *in the user's sense:* POs are only about verification. Validation is required to check that the machine conforms to the users' actual requirements.

2.4.3 Rodin Platform

The Rodin platform [RODIN 2012] is an Eclipse-based integrated development environment for Event-B. It provides specifiers with an effective support for editing the specifications, refining the models, generating and discharging the POs. The platform is an open source and extendable with external plug-ins. The architecture of the Rodin platform is composed of three sets of tools: the Eclipse platform, the kernel plug-ins and the external plug-ins.

2.4.3.1 Eclipse Platform

The Eclipse platform provides the basic tools for constructing an integrated development environment. It is easily customized to support any particular development process. Based on this feature, Rodin reuses the most notable Eclipse facilities to support the Event-B modeling and proving processes.

2.4.3.2 Kernel Plug-ins

The kernel plug-ins provide the basic modeling and proving functionalities with Event-B. They are developed on top of the Eclipse platform. They are composed of a set of plug-ins for model storage, checking, proof manage and user interface.

Core This plug-in provides low-level APIs to operate Event-B models with a database manger. All elements related to Event-B models are stored into a Rodin database bases

on XML files. This plug-in also provide a incremental project builder to schedule the static checker, the proof obligation generator and the prover.

Static Checker This plug-in provides APIs to check Event-B models. The static checker is firstly called once an Event-B model is saved. The mathematical formulas are statically checked for being meaningful. Every formula is parsed to an abstract syntax tree (AST) representation. The proof obligations are only generated from a statically sound model.

Proof Obligation Generator This plug-in automatically generates the POs for the checked Event-B models without errors.

Prover This plug-in provides APIs to discharge the POs. It contains a Proof Manager and a set of proof engines.

Modelling UI This plug-in provides the graphical user interface to write and edit Event-B models. This interface contains fours areas for the project navigation, component editing, outline view, and message reminding.

Proof UI This plug-in provides the graphical user interface for displaying, managing, and discharging the interactive proofs.

2.4.3.3 External Plug-ins

The external plug-ins [1] are all the other plug-ins that can be used in the Rodin Platform. Some of them have been developed in the course of the Rodin Project (UML to B translator, ProB model checker, etc.) while others might be developed later on. Here we just present some useful external plug-ins. A set of published APIs allows any Rodin user to contribute specific plug-ins.

Edition Tools The *Camille Editor* [Bendisposto 2010] is a plain text editor for Event-B language. Is is similar to the classical programing language editors. It complements the standard structured editor of Rodin.

Verification Tools The *AtelierB Prover* [Cle 2009a] is a powerful prover; it is recommended to use it in place of the internal provers. *Isabelle for Rodin* [Schmalz 2011] exports POs as Isabelle/HOL theories so they can be discharge with Isabelle/HOL. The *SMT Plug-in* [Déharbe 2012] provides an interface for SMT solvers.

Validation Tools *ProB* [Leuschel 2003, Ligot 2007, Leuschel 2008] is an animator and model checker for the B Method. *Brama* [Servat 2007] and *AnimB* [Métayer 2012] are animators for the Rodin platform. The *UML-B-State-machine Animation* [Savicks 2009] provides an animation of UML-B State-machines by using ProB.

1. http://wiki.event-b.org/index.php/Rodin_Plug-ins

Code Generation Tools *B2C* [Wright 2009a] automatically translates Event-B models to C source code, while *EB2ALL* [Méry 2011] is a set of plug-ins to translate Event-B models to several programing language, such as C, C++, Java and C#. Both B2C and EB2ALL only support a subset Event-B notations without non-deterministic notations, therefore they are applied to the last refinement which is enough deterministic. *Tasking Event-B* [Edmunds 2012] supports multi-tasking Java, Ada and OpenMP C code generation from an extended tasking Event-B model. *EventB2JML* translates Event-B machines into JML (Java Modeling Language) specifications. *EventB2Dafny* translates Event-B proof-obligations into equivalent Dafny [Leino 2010] programs which can be discharged with Dafny proof environment. The *EHDL Plug-in* enables the automatic generation of VHDL [Ashenden 2002] code from Event-B models.

2.5 Animation of Event-B Models

Animation is a technique to *execute* specifications. Thus, we can play, experiment and observe the behavior of models. Several tools support this technique [Van 2004, Bendisposto 2008, Leuschel 2001, Schmid 2000, Siddiqi 1997].

2.5.1 Animation Difficulties

The Event-B animators are used to observe the behaviors of a given model. Brama, AnimB and ProB are animators integrated within the Rodin platform; B2EXPRESS [Ait-Sadoune 2009] is a standalone animator. An animator for Event-B should resolve two issues : execution and visualization.

An Event-B model is not an executable program. The capability of an Event-B animator depends on its level of support of Event-B mathematical notations. Furthermore, an Event-B animator has to address some challenges during the executions:
– to find values for the constants and sets which satisfy their properties,
– to find values for the parameters of a given event,
– to decide the set of enabled events and to schedule an event to fire.
Unfortunately, the automatic solution of the above problems are undecidable. Indeed, the set of possible values can be infinite in Event-B.

For the purpose of validation, an Event-B animator should also provide with several user-friendly visualization features. They help users to analyze and understand the behavior of Event-B models; e.g., the graphical representation of the system state and the interaction controls for users.

2.5.2 Event-B Animators

Brama This tool uses a Java library to parse and interpret Event-B formulas. It requires the version under 1.0 of Rodin; so it cannot be used with the latest versions 2.x. Brama contains the following principal modules: an animation engine (predicate solver), an

interface to visualize events and variables, and a communication module with Flash animation.

Brama uses an enumeration strategy to compute values and parameters: all carrier sets and constant functions need be instantiated by explicit set extensions. This strategy limits Brama's applicability to Event-B models which contain constants and sets with finite values.

The parameter value to a given event is randomly picked from the set of values which satisfy the event guard. If this set is empty, then the given event is not enabled. The user can also input a special parameter value through a user interface. The input value is verified by the event's guard.

To schedule the enabled events, Brama use an XML file to configure events' order of firing. With this file, Brama can run the model in an automatic mode.

Brama uses a Flash communication module to build the external graphical animation. The communication between the Flash interface and the Event-B model is based on the exchanges of some observed expressions and predicates which are configured in an XML file.

Brama partially supports the Event-B notations; many Event-B models are not directly animatable with Brama. They need some transformations [Mashkoor 2009b] before using Brama, for example:
– specify the finiteness of a quantified domain,
– generalize expressions involving complex iterations,
– explicitly provide the typing information of all sets used in an axiom,
– avoid dynamic function computation in substitutions,
– use in-line functions in machines to replace those functions defined in contexts.
Notice that the strategy of these transformations is to change a formal specification to adapt an animator tool because of its limitations. Despite these transformations, many specifications are still non-animatable by Brama.

AnimB AnimB is similar to Brama, but it supports the latest Rodin version 2.x. It uses the same Java library to parse and interpret Event-B formulas, so the limitations of AnimB are the same as those of Brama. Furthermore, AnimB does not check axioms and invariants. So, there is no indication that an observed behavior may not be allowed in the original model. This is main disadvantage of AnimB. Like Brama, it provides users with an interface toward Flash for building graphical visualizations.

ProB Based on Prolog, the ProB tool supports the automated consistency checking of B machines via model checking. Both exhaustive and non-exhaustive model checking are supported by ProB. The exhaustive model checking requires that the checked model only uses small finite sets and integer variables are restricted to small numeric ranges. Otherwise, the size of carrier sets and the range of integers are bounded in the ProB preferences. ProB is useful to find out a counter-example where a specification contains errors. If any invariant violation or any deadlock is not found by ProB, this does not

imply that the specification is correct. It should be understood that no error was found within the given checking conditions.

For any particular animation, two strategies can be used to find specific values for all constants and sets: (i) let ProB do this automatically, using the constraint-solving technique to find proper values that satisfy all axioms, (ii) use the context extension mechanism to explicitly set values to all constants and sets; this is useful when the automatic solution fails or when we want an animation on a particular scenario with specific data.

Like Brama, ProB will calculate a set of parameter values for a given event. The size of this set is configured in the ProB preference. Users can pick the first solution for parameters, a random solution for parameters, or open a dialog for choosing. Unlike Brama, ProB does not provide a user interface to input a specific parameter value.

To schedule the enabled events, users can automatically execute a certain number of steps or manually choose an enabled event. ProB does not provide an explicit scheduler for auto run mode.

The visualization with ProB uses an animation function written in B to link the model and some static images. These visualizations are rather simple and restricted. Also, writing the required animation function can still be a considerable challenge.

Based on the ProB plug-in for Rodin, the B-Motion Studio [Ladenberger 2009] allows to create visualizations as using animation functions in the standalone ProB. BMotion Studio has a graphical editor with a number of default controls and observers to facilitate the construction of visualizations. Users use controls, like labels, images or buttons, to construct a graphical representation of the model. Those controls are linked to the model's state by some observers which use Event-B predicates and expressions as gluing code. Unlike a single animation function in the standalone ProB, observers are more flexible to construct a complex animation with many small functions.

B2EXPRESS This tool translates Event-B models into data models expressed in the EXPRESS [Schenck 1994] data modeling language. The animation consists in instantiating the different entities of the obtained data models to describe the traces of events of an Event-B model. The interesting point is that B2EXPRESS offers two animation mode: (i) the guarded mode only allows users to trigger those events whose guard evaluates to true, (ii) the free mode may allow users trigger any event, even whose guard evaluates to false. The first mode can be used to the validation activities, while the second mode is useful to debug an Event-B models, e.g., to check invariants, deadlocks or unexpected behaviors. This tool is not publicly available and is not integrated into the Rodin platform.

2.5.3 Multi-level Animation of Refinement

For complex models, the refinement technique allows us to introduce details gradually in order to reduce the complexity at each refinement step. The proof obligations generated

with each refinements can guarantee the consistence between refinements, but sometimes it is difficult to analyze a refinement relationship. Brama, AnimB and ProB provide a multi-level animation facility to help detect refinements' errors.

Compared to Brama and AnimB, the articles [Hallerstede 2010, Hallerstede 2013] present an algorithm for the simultaneous multi-level animation of refinements according to witnesses. This algorithm allows ProB to detect more efficiently a variety of refinement errors in a systematical way. The witnesses in Event-B play an important role to realize this algorithm.

2.6 Platooning Models

Our research strategy makes a heavy use of case studies. We started our work with the analysis of an existing Event-B specification (see Appendix D) which formalizes a platooning algorithm for the longitudinal control. We extended this specification to the bi-dimensional space by adding the lateral control. Through these two case studies, we have identified the following problems:

– *A fully proved Event-B specification can be incorrect from the users' point of view because of missing validation activities, i.e., verified but not validated.*
– *The industrial application of Event-B lacks mature tools to support validation activities in the current state of the art.*
– *We need a practical process to integrate formal reasoning and semi-formal reasoning into the validation and verification activities.*

2.6.1 Platooning Problem

Research on urban mobility systems based on fleets of small electric vehicles [Baille 1999] stresses the importance of a new moving mode: platooning [Bom 2005]. A platoon is defined as a convoy of autonomous vehicles which follow exactly the same path and which are spaced at very close distance one from the other.

In this work, we consider platoons formed by a *leader* vehicle and *followers*. Leaders and followers have different control laws. We specify only the follower control law. Its aim is to keep as close as possible to the preceding vehicle while following a virtual ideal track without colliding. We use a model of vehicle where the control can be decomposed into longitudinal (i.e., speed and acceleration) and lateral (i.e., curve and wheel orientation) laws. We assume operating conditions such that the two controls can be computed independently.

There are numerous strategies to form and maintain platoons, characterized by their degree of centralization and the volume of communication. We specify a minimal strategy: no central control and no communication [Scheuer 2009] between vehicles other than perception, i.e., a vehicle can sense a few information from the preceding vehicle (distance, speed, etc.). The virtual track is set by the leader. The control is local to each vehicle, based on current state and perceptions. This strategy may not be one of

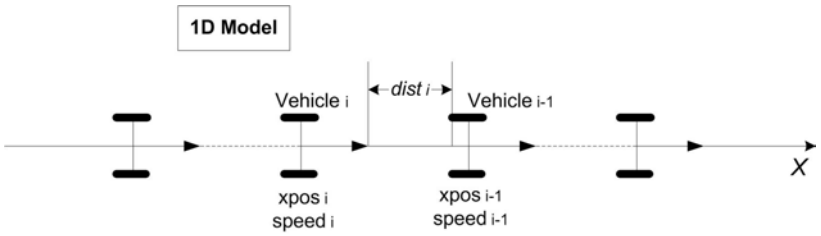

Figure 2.2: 1D platooning model

most efficient but it is very robust. In particular, it can be used as a fall-back in case of failure in a system using more sophisticated algorithms. Hence the need to guarantee its correctness.

Within this problem setting, platoons can be considered as situated multi-agent systems (MAS) which evolve following the Influence/Reaction model [Ferber 1996, Ferber 1999]. The development of the specification follows a stepwise refinement process based on this model: (i) driving systems perceive, (ii) decisions are taken, and (iii) physical vehicles move.

We aim at modeling a pragmatic strategy known as *Daviet-Parent algorithm* [Daviet 1996] in order to prove that implementations enjoy certain properties such as: (i) the model is sound bound-wise, (ii) no collision occurs between the vehicles, (iii) no unhooking occurs, and (iv) no oscillation occurs.

Presently, we focus on two essential safety properties: (i) soundness is maintained and (ii) no collision within a platoon occurs [2].

2.6.2 1D Platooning Models

In the 1D platooning model, vehicles move on a linear track depicted in Figure 2.2. It is formalized in classic B, Event-B and CSP||B. These models are proved to be correct.

Modeling in Classic B The specification described in [Simonin 2007] clearly separates the model of the physical world (environment) and the model of the vehicle behavior (agents). These two parts can be connected to express the full model, and separated to identify what are the specific properties that the model must verify. From the case study of the platooning problem, it proposed a generalized pattern to help the modeling of Influence/Reaction multi-agent systems.

Modeling in Event-B In [Lanoix 2008], the state of the i^{th} vehicle at time t is the pair $(xpos_i(t), speed_i(t))$, where $xpos_i$ represents its position on the track and $speed_i$ represents its velocity. The longitudinal control consists in setting up an acceleration to

2. Collisions between platoons or between a vehicle and an obstacle should of course be considered in a real system. First kind should be taken care by the control law of leaders, second kind is dealt with by lower level emergency systems. Both are outside the scope of this work.

modulate speed. The behavior law is represented by (2.1) extracted from [Simonin 2007], where $MaxSpeed$ is the maximum velocity, $accel_i$ is the acceleration, and Δt is the time increment:

$$\left\{\begin{array}{rcl} n\_speed & = & speed_i(t)+accel_i(t).\Delta t \\ xpos_i(t+\Delta t) & = & \left\{\begin{array}{ll} xpos_i(t)+MaxSpeed.\Delta t & \text{if } n\_speed > MaxSpeed \\ xpos_i(t)-\frac{speed_i(t)^2}{2.accel_i(t)} & \text{if } n\_speed < 0 \\ \left(\begin{array}{l} xpos_i(t)+speed_i(t).\Delta t \\ +\frac{accel_i(t).\Delta t^2}{2} \end{array}\right) & \text{otherwise} \end{array}\right. \\ speed_i(t+\Delta t) & = & \left\{\begin{array}{ll} MaxSpeed & \text{if } n\_speed > MaxSpeed \\ 0 & \text{if } n\_speed < 0 \\ n\_speed & \text{otherwise} \end{array}\right. \end{array}\right. \tag{2.1}$$

The acceleration $accel_i$ is chosen according to the current state of the i^{th} vehicle and the values sensed on the preceding vehicle.

This model is strongly influenced by the non-collision property. It starts from a very abstract Event-B model. Then this first model is refined by adding the various stages of operations (reaction, decision and perception) until it is complete with respect to the behavior law. This stepwise refinement is properly verified at each stage, which ensures that the initial properties remain true in the final model.

The structure of this model consists of an abstract machine and four refinements. Each development introduces a clearly identified concept. The abstract machine `platoon0` sets up the safety property. The refinement `platoon1` splits the movement of the platoon into the movements of each vehicle. The refinement `platoon2` implements the physical reaction law. The refinement `platoon3` introduces the decision law to compute acceleration. The refinement `platoon4` introduces the perception of the environment and implements the decision law.

Modeling in CSP||B This specification [Colin 2008] finely handles communication issues between the components of the vehicle and between the vehicles themselves. Indeed, the platooning model assumes that vehicles must operate at the same time. This hypothesis may even extend to sub-components of a vehicle. This model can ensure that the selected communication protocol does not cause blocking and the perceptions made by the vehicles are sufficient for decisions.

2.6.3 2D Platooning Model

The 2D model [Yang 2011], is based on the following system hypotheses:
– we consider a set of $N(\geq 2)$ vehicles forming a linear platoon,
– the motion of the vehicles is limited by fixed bounds on velocity, acceleration, curvature and derivative of curvature,

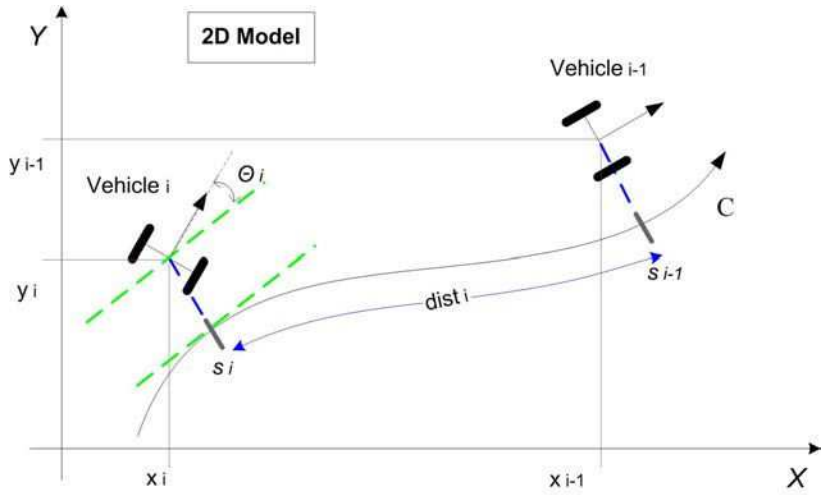

Figure 2.3: 2D platooning model

– we consider forward-only motions on a track which does not intersect itself (no loops),
– we suppose that the frequency of the control algorithm is the same for all vehicles, so they can be modeled as synchronized,
– sensors are perfect and their accuracy is such that the velocity of the previous vehicle can be precisely known,
– actuators of the engine are perfect.

The hypotheses are strong but not that far from reality considering (1) we are not modeling fault, fault-tolerance, or such matters, (2) near perfect abstract sensors or actuators can be built from merging results of several concrete ones.

In the 2D model, vehicles move on a plane shown in Figure 2.3. The vehicle state η must model its position, represented by cartesian coordinates (x,y), and its attitude, represented by the orientation θ of the vehicle's axis with respect to the x-axis. The behavior law now contains a velocity v and a trajectory's curvature κ which are controlled by application of a linear acceleration a and a derivative of the curvature χ. When a control (a,χ) is applied to a state $(x_0,y_0,\theta_0,v_0,\kappa_0)$ at time t for a period Δt, the new state at time $t+\Delta t$ becomes:

$$\begin{cases} x = x_0 + \cos\theta_0 F_C(\Delta t, v_0, \kappa_0, a, \chi) - \sin\theta_0 F_S(\Delta t, v_0, \kappa_0, a, \chi) \\ y = y_0 + \cos\theta_0 F_S(\Delta t, v_0, \kappa_0, a, \chi) - \sin\theta_0 F_C(\Delta t, v_0, \kappa_0, a, \chi) \\ \theta = \theta_0 + v_0\kappa_0\Delta t + (a\kappa_0 + v_0\chi)\frac{\Delta t^2}{2} + a\chi\frac{\Delta t^3}{3} \\ v = v_0 + a\Delta t \\ \kappa = \kappa_0 + \chi\Delta t \end{cases} \tag{2.2}$$

where $F_C(\Delta t, v_0, \kappa_0, a, \chi) = \int_0^{\Delta t}(v_0+at)\cos(v_0\kappa_0 t + (a\kappa_0+v_0\chi)t^2/2 + a\chi t^3/3)dt$
and $F_S(\Delta t, v_0, \kappa_0, a, \chi) = \int_0^{\Delta t}(v_0+at)\sin(v_0\kappa_0 t + (a\kappa_0+v_0\chi)t^2/2 + a\chi t^3/3)dt$.

One aim of the 2D model development is to analyze how the slight increase in complexity of the mathematical model translates into the increase in complexity of the Event-B model and of its development.

2.7 Summary

The development of software systems needs to adopt an appropriate development process to resolve the management problem. The quality of software systems can be improved by applying formal methods. Formal methods can be applied at various points through the development process. One objective of this thesis aims at formal modeling and analysis of a platooning algorithm with the Event-B language. We are interesting in all aspects of integrating methods, either formal or semi-formal, for the system development, covering all engineering development phases from user requirements to verification and validation activities.

Part I

Assessment of Event-B Usability

Chapter 3

Analysis of the 1D Platooning Model

Contents

3.1 Introduction

The 1D platooning model in Event-B (see Appendix D) is the starting point of our research work. We undertook a thorough analysis of this model in order to gain a better understanding of the relations between the proof of a formal model and its correction in the general sense. This leads us to uncover problems related to the presence of deadlocks and some weaknesses of the invariants. Notices that the result of proofs depends the capability of the state-of-the-art the verification tools. At the beginning of our research, only the internal prover and the Atelier B prover are available to use.

3.2 Proofs

In the verification sense, the correctness of an Event-B model fundamentally depends on proofs. When an automatic proof fails [Abrial 2010], it might be:

1. the statement to prove is true, but the automatic prover is not smart enough, so we must prove it with the interactive procedure;
2. the statement to prove is false, so the model has to be significantly modified;
3. the statement to prove cannot be proved, so the model should to be enriched.

The three situations existed in the original 1D platooning model. Case 1 was caused by two *reviewed* sequents; case 2 was caused by another two *reviewed* sequents; and case 3 was found by introducing the deadlock-freeness theorems. In Rodin, *reviewed* sequents mean admitted without proof. Marking sequents as *reviewed* is convenient since the provers will ignore them and can proceed with the general proof. This allows the proof to be discharged interactively in a gradual way. Of course, *reviewed* sequents must ultimately be proven, preferably within Rodin's provers.

3.2.1 Interactive Proof

Two well-definedness proof obligations are marked as *reviewed* in `context2` (see Appendix D.3): "axm12/WD" and "axm14/WD". At the time the specification was developed, Rodin's version was 0.8, the SMT solver plug-in is not available to use. The available provers (the Rodin internal prover and the Atelier B prover) then could not discharge automatically the two following sequents:

$$\begin{array}{l} \neg accel0 = 0 \\ \vdash \\ \neg 2 * accel0 = 0 \end{array}$$

It was an instance of an "obvious" mathematical property:

$$\forall n \cdot n \in \mathbb{Z} \wedge \neg n = 0 \Rightarrow \neg\ 2 * n = 0 \tag{3.1}$$

In order to prove the above sequent, we can adopt the following strategy:

1. select the hypothesis $\neg accel0 = 0$, remove the other hypothesis;
2. add a new hypothesis $2 * accel0 > 0 \vee 2 * accel0 < 0$;
3. discharge the sequent by a case analysis decomposition: $2 * accel0 > 0$ and $2 * accel0 < 0$.

Indeed, the new version of Atelier B prover (since the version 1.0) can automatically discharge the above sequent. We think that a similar proof rule was added to this prover.

3.2.2 False Statement

Two **GRD** proof obligations are marked as *reviewed* in the machine `platoon2` (see Appendix D.8): "move1_max/grd3/GRD" and "move_max/grd3/GRD". They are used to express the consistency of refinements. The guard of the refined events `move1_max` and `move_max` must be stronger than the guard of the abstract events `move1` and `move` in the machine `platoon1` (see Appendix D.7), respectively.

In the proof tree, two goals cause problems:

move1_max/grd4/GRD
Goal: $new\_xpos\_max(xpos(1) \mapsto speed(1) \mapsto magic\_accel) \geq xpos(1)$

move_max/grd4/GRD
Goal: $new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel) \geq xpos(vehicle)$

They relate to the function *new_xpos_max* defined in `context2`:

axm11: $new\_xpos\_max \in \mathbb{N} \times 0..MAX\_SPEED \times MIN\_ACCEL..MAX\_ACCEL \rightarrow \mathbb{N}$
axm12: $\forall xpos0, speed0, accel0 \cdot ($
$xpos0 \in \mathbb{N} \wedge speed0 \in 0..MAX\_SPEED \wedge accel0 \in MIN\_ACCEL..MAX\_ACCEL$
$\Rightarrow ($
$accel0 = 0 \Rightarrow$
$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED$
$\wedge$
$accel0 \neq 0 \Rightarrow$
$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED$
$-((MAX\_SPEED - speed0) * (MAX\_SPEED - speed0)) \div (2 * accel0)$
$)$
$)$

In the case when $accel0 \neq 0$, the function *new_xpos_max* can be defined either by:

$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED$
$-((MAX\_SPEED - speed0) * (MAX\_SPEED - speed0)) \div (2 * accel0)$

or by:

$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0$
$+((MAX\_SPEED - speed0) * (MAX\_SPEED - speed0)) \div (2 * accel0)$
$+speed0 * ((MAX\_SPEED - speed0) \div accel0)$
$+MAX\_SPEED * (1 - (MAX\_SPEED - speed0) \div accel0)$

The two definitions are equivalent and can be easily transformed one into the other in the domain of reals. This transformation is based on a mathematical property:

$$\forall a, b, c \cdot a \in \mathbb{R} \wedge b \in \mathbb{R} \wedge c \in \mathbb{R} \wedge c \neq 0 \Rightarrow a * (b \div c) = (a * b) \div c \tag{3.2}$$

Unfortunately, this equation is not true in the domain of integers which is the only supported number type in the Event-B language. The $\div$ operator means the Integer

division in Event-B. For example, in the domain of integers:

$$5 * (5 \div 8) = 5 * 0 = 0$$

$$(5 * 5) \div 8 = 25 \div 8 = 3$$

We must use the second version to define the *new_xpos_max* in Event-B in order to interactively discharge the *reviewed* goals in the POs "move1_max/grd3/GRD" and "move_max/grd3/GRD". Hence, the correct version for `axm12` is:

axm12: $\forall xpos0, speed0, accel0 \cdot ($
$xpos0 \in \mathbb{N} \land speed0 \in 0..MAX\_SPEED \land accel0 \in MIN\_ACCEL..MAX\_ACCEL$
$\Rightarrow ($
$accel0 = 0 \Rightarrow$
$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED$
$\land$
$accel0 \neq 0 \Rightarrow$
$new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0$
$+((MAX\_SPEED - speed0) * (MAX\_SPEED - speed0)) \div (2 * accel0)$
$+speed0 * ((MAX\_SPEED - speed0) \div accel0)$
$+MAX\_SPEED * (1 - (MAX\_SPEED - speed0) \div accel0)$
$)$
$)$

This mistake cannot be caught when proving the context. Both definitions are well-defined in Event-B. When a machine's formula to prove involves this complex kinematic function, the prover must expand the "function call" by its body. The impossible equation $(a * (b \div c) = (a * b) \div c)$ becomes a lemma which cannot be proved. The problem is that the failure of the proof is difficult to interpret: is it a weakness of the current prover, or, is it a genuine incorrect sequent? This raises interesting questions about the modeling strategies to recommend.

3.2.3 Unprovable Goal

After the issues raised in Section 3.2.1 and Section 3.2.2 have been solved, the original 1D platooning model can be fully proved. We could declare it "correct" with respect to preserving the non-collision invariant. Unfortunately, collisions were observed on executable implementations of the same model. They were finally explained by the occurrences of deadlocks in certain circumstances.

For each machine, we can manually construct a deadlock-freeness theorem to guarantee that the machine should work for ever. However, the Rodin platform lacks a mean to automatically construct it. In Chapter 4, we will present a tool to help the construction of deadlock-freeness (DLF) theorems.

In the reviewed 1d platooning model, the deadlock-freeness theorems for the machines `platoon0` and `platoon1` can be easily proved. But a problem is found for the machine `platoon2`. The deadlock-freeness theorem is depicted in Figure 3.1.

$$
\begin{array}{l}
(\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle = 1) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed \in 0 \mapsto MAX\_SPEED) \wedge \\
\qquad (nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel))) \\
) \vee (\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle = 1) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed > MAX\_SPEED) \wedge \\
\qquad (nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel))) \\
) \vee (\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle = 1) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed < 0) \wedge \\
\qquad (nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel))) \\
) \vee (\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle \in 2 \mathinner{..} VEHICLES) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed \in 0 \mathinner{..} MAX\_SPEED) \wedge \\
\qquad (nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)) \\
\qquad (xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE)) \\
) \vee (\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle \in 2 \mathinner{..} VEHICLES) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed > MAX\_SPEED) \wedge \\
\qquad (nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE)) \\
) \vee (\\
\quad \exists\ magic\_accel, nspeed, nxpos \cdot (\\
\qquad (vehicle \in 2 \mathinner{..} VEHICLES) \wedge \\
\qquad (magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL) \wedge \\
\qquad (nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (nspeed < 0) \wedge \\
\qquad (nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)) \wedge \\
\qquad (xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE)) \\
) \vee (\\
\quad (vehicle = VEHICLES + 1) \\
)
\end{array}
$$

Figure 3.1: The DLF theorem for the machine platoon2

Goal 1 for the event move_normal:
$xpos(vehicle-1)-xpos(vehicle)$
$\quad -(speed(vehicle)+MIN\_ACCEL \div 2) > CRITICAL\_DISTANCE$

Goal 2 for the event move_reduce:
$xpos(vehicle-1)-xpos(vehicle)$
$\quad -speed(vehicle)*speed(vehicle) \div (-2*MIN\_ACCEL) > CRITICAL\_DISTANCE$

Figure 3.2: The unprovable goals in the machine platoon2

The two goals shown in Figure 3.2 cannot be proved in the generated **THM** proof obligation under the referenced axioms and invariants. The "failed" goals uncover a critical weakness in the invariant; it is discussed in the next section.

3.3 Non-Collision Property

Any implementation of a platooning system must guarantee the absence of collision between vehicles from the same platoon. The non-collision property can be expressed by a natural language sentence in a requirement document, such as:

"*No collision shall occur between the vehicles in a platoon.*"

The goal's statement is correct, but not accurate enough. Let us see how it was formalized in the 1D platooning model.

3.3.1 Machine platoon0

The Event-B machine `platoon0` represents the first abstract model of the platooning problem. The model is restricted to the longitudinal control. So, we observe only the longitudinal position of the vehicles. All vehicles make a simultaneous move at a time. The non-collision property is expressed by the following invariant:

$$\forall v \cdot (v \in 2\,..\,VEHICLES \Rightarrow xpos0(v-1)-xpos0(v) > CRITICAL\_DISTANCE)$$

where

– $VEHICLES \in \mathbb{N}$ is the number of vehicles in the platoon,
– $CRITICAL\_DISTANCE \in \mathbb{N}$ is the safe minimal distance between two adjacent vehicles,
– $xpos0 \in 1\,..\,VEHICLES \rightarrow \mathbb{N}$ is the longitudinal position of the vehicles.

At this abstract view, no collision occurs between a vehicle and its predecessor. The POs of the deadlock-freeness theorem are easily discharged, so the system can safely run forever.

3.3.2 Machine platoon1

The first refinement `platoon1` decomposed the simultaneous movement into independent movements of each vehicle, starting from the leader. The non-collision property has to be strengthened: no collision happens to the vehicle which has to move. The refined invariant is expressed as:

$$\forall v \cdot (v \in 2\,..\,vehicle - 1 \Rightarrow xpos(v-1) - xpos(v) > CRITICAL\_DISTANCE)$$

where
– $vehicle \in 1\,..\,VEHICLES + 1$ identifies the current vehicle which has to move,
– $xpos \in 1\,..\,VEHICLES \rightarrow \mathbb{N}$ is the position of each vehicle during the movement.
The POs of the deadlock-freeness theorem cannot be discharged. This implies an implicit deadlock for the initialization event which can lead to a collision position but also preserves the invariant. For discharging the POs of the deadlock-freeness theorem, this invariant should be strengthened as:

$$\forall v \cdot (v \in 2\,..\,VEHICLES \Rightarrow xpos(v-1) - xpos(v) > CRITICAL\_DISTANCE)$$

Therefore any initial collision position is excluded from the platooning model.

3.3.3 Machine platoon2

The machine `platoon2` refines `platoon1` to implement the reaction laws which are given in the equation (2.1) (see Page 25). Each vehicle moves as if reacting to an instantaneous acceleration. A new variable *speed* is introduced so the acceleration can be applied and the new position can be computed. Thus, new elements (speed and acceleration) can be observed, however, the invariant modeling the non-collision property is not strengthened.

As we know, deadlocks can be observed on the execution of this refinement. The unprovable goals in Figure 3.2 hint at the weakness of the invariant: speed and acceleration should also be considered.

Let

$$\begin{aligned} a &= speed(vehicle) + MIN\_ACCEL \div 2 \\ b &= speed(vehicle) * speed(vehicle) \div (-2 * MIN\_ACCEL) \\ brake\_distance &= max(\{a, b\}) \end{aligned}$$

The goals in Figure 3.2 can be united into a single expression:

$$xpos(vehicle - 1) - xpos(vehicle) > CRITICAL\_DISTANCE + brake\_distance$$

In order to achieve the above goal, a strengthened invariant is evidently required which must consider the speed and the acceleration to compute an extra braking distance in the most critical situations: if a vehicle brakes at maximal deceleration, the following vehicle must be able to brake too and not collide its predecessor. So, the invariant must have the following form:

$$\forall v \cdot (v \in 2..VEHICLES \Rightarrow xpos(v-1) - xpos(v) > CRITICAL\_DISTANCE + brake\_distance)$$

Indeed, the model presented in [Scheuer 2009] is a collision-free model which presents a similar formula taking into account the speed differential:

$$\forall v \cdot (v \in 2..VEHICLES \Rightarrow xpos(v-1) - xpos(v) > CRITICAL\_DISTANCE + max(\{0, (speed(v-1) * speed(v-1) - speed(v) * speed(v))/(2 * MIN\_ACCEL)\}))$$

Unfortunately, we cannot simply apply this invariant to achieve a deadlock-free platooning. A drastic rewriting of the specification `context2` and `platoon2` is required. More importantly, the 1D platooning model uses a trick, $\Delta t = 1$, to simplify the modeling, which assumes that the time of an execution cycle is equal to a small time slice. How this trick influences the transformation of equations and the modeling strategy is yet an unresolved question. This work is out of the scope of this thesis.

3.3.4 Machine platoon3 and platoon4

Since the machine `platoon2` has not preserved the non-collision property, the refinements `platoon3` and `platoon4` must be modified too. The current discharged POs of these refinements are meaningless. Indeed, we can use the animation and simulation techniques to quickly identify the deadlock behaviors in `platoon3` and `platoon4`. We cannot escape a complete rework of those two refinements with *all* the proofs to make again.

This is a strong call for the validation as early as possible of the formal models. It is particularly important with behaviors like the temporal properties which are difficult to specify in Event-B. Otherwise, we run a serious risk of losing a lot of hard work.

3.4 Summary

Event-B does not incorporate a notion of time or of temporal logic. In fact, there is not even a notation to express that an event follows another one. Temporal properties are often the most critical part of safety-concerned systems. Deadlock-freeness, an important safety property, has a very limited support by the current Rodin platform.

The analysis of an existing specification is a case study on the notion of “correctness” as applied to formal models. It shows why proofs are not sufficient. It also shows how the technical work, i.e., the refinements, can influence the requirements and their expressions in subtle ways. This makes a strong case for tools to help developers validate their models.

Chapter 4

Automatic Generation of DLF Theorems

Contents

4.1 Introduction

From an operational point of view, a computation modeled in Event-B starts by firing the `INITIALISATION` event; it halts when no event can be fired, i.e., no event has its guard evaluated to true. There are two opposing situations related to the halting problem.

In the first situation, we want the system to reach a terminal state. Then, we must prove that the computation *will* halt. Event-B provides a notion of *variant* which is associated to proof obligations and can be used to prove the termination of a chosen set of events.

In the second situation, we expect the system to cycle endlessly. We must then prove that the computation will *not* halt. Event-B provide us with a **DLF** (Deadlock-Freeness) rule.

There are other techniques to study the halting status of a model. For instance:
– Animation with an interactive tool such as Brama [Servat 2007] or simulation with JeB allows one to observe the occurrence of deadlocks within specific scenarios.
– Model checking allows one to check systematically for (an absence of) deadlocks within a finite space or partial transitions. For the infinite or very large state space, ProB [Leuschel 2008] uses a constraint-based approach to find deadlocks [Hallerstede 2011].

These techniques can help us to find a deadlock, but they cannot prove that the system is free of deadlocks. Using the theorem proving technique on DLF theorems, we can formally assess that the system is actually free of deadlocks. The DLF rule is not integrated into the Rodin platform. We have to manually write the DLF theorems. A tool to replace the error-prone manual writing of ad-hoc formulas is required for enhancing Event-B usability. This is one of our contributions.

4.2 Deadlock-Freeness Rule

Depending on the system being modeled, the deadlock-freeness can be a property of either the whole model or of a part of the model. In our work, we use the symbols and notations defined in Table 9.1 (Chapter 9) to describe the formal definition of this rule.

4.2.1 Deadlock-Freeness of Complete Model

The DLF rule for a model (defined in [Abrial 2010] Section 2.4.21) states that, at all time, one of the guards $Grd_{E^1}(x_{E^1}, s, c, v)$, ..., $Grd_{E^m}(x_{E^m}, s, c, v)$ is true. So, at least one event other than `INITIALISATION` can always be fired. Formally, this is expressed by the following sequent which makes explicit the set of axioms $Axm(s,c)$ and invariants $Inv(s,c,v)$.

$$\begin{array}{|l} Axm(s,c) \\ Inv(s,c,v) \\ \vdash \\ \bigvee_{i=1}^{m} \exists x_{E^i} \cdot Grd_{E^i}(x_{E^i}, s, c, v) \end{array}$$

On the Rodin platform, the provers can automatically reference the axioms $Axm(s,c)$ and invariants $Inv(s,c,v)$ to discharge the above sequent, so the DLF theorem for a model can be simplified as:

$$\bigvee_{i=1}^{m} \exists x_{E^i} \cdot Grd_{E^i}(x_{E^i}, s, c, v) \tag{4.1}$$

When this formula is constructed and inserted in the specification as a **theorem**, Rodin automatically generates a **WD** proof obligation and a **THM** proof obligation. Note that there is only one general DLF theorem for a given model.

4.2.2 Deadlock-Freeness of a Subset of Events

Some systems may be structured in such a way that some events are always enabled. For instance, the control system of an automated vehicle may accept input from the driver at any time. This would be modeled as an event which is always enabled, i.e., whose guard is always true. So, the general DLF theorem stated above is trivially true. But, for the model to be correct, the part which specifies the actions on the vehicle must be free from deadlocks. So it is important to provide specifiers with a way to express the DLF

Machine	Events	Guards	DLF #lines
platoon0	1	2	6
platoon1	3	8	16
platoon2	7	34	54
platoon3	9	42	68
platoon4	15	79	121

(a) 1D platooning model

Machine	Events	Guards	DLF #lines
platoon0	1	5	19
platoon1	3	10	28
platoon2	19	184	249
platoon3	21	177	264
platoon4	39	354	637

(b) 2D platooning model

Table 4.1: DLF theorem size

theorem on a subset of events. The selection of events concerned can be easily done with a small user-interface. The theorem itself has the same form as the equation 4.1 for the system. Note that several partial DLF theorems can be generated for the same model.

4.3 Exigence of a Tool

We can write manually the DLF theorems for a small Event-B model; however, this becomes soon unpractical as the size of the model grows. Let us consider the two platooning models which motivated this work. These two models are presented in Appendixes D and E. Table 4.1 gives some figures showing the evolution of the size of DLF theorems.

Two points should be noted. Firstly, the size of DLF theorems is roughly twice the sum of the length of guards of all events. So, it grows as fast as the refinement does and it adds up quickly to several tens of lines. Secondly, the deadlock-freeness is not a monotonous property with respect to refinements: it should be established for each refinement. In [Idir 2010], each machine generated from a business process model need manually write this theorem to verify the property of deadlock-freeness.

Automating the generation of DLF theorems is needed for several reasons:

– the “copy and paste” procedure is highly unreliable. The probability to introduce an error is high and the length of the formula makes spotting errors hard,
– any modification of the guards requires a modification of the theorems. In particular, it should be easy to re-generate the theorems while correcting a model,
– manual generation of a theorem is not intellectually challenging (it is boring in fact) and takes a lot of development time.

The core of the generation is a syntactic manipulation of the model: extracting the guards, gluing them together in a single formula, and inserting it as a **theorem** into the **INVARIANTS** part of an Event-B machine. Programming tools and techniques to cover such tasks are well known.

A small user-interface is needed to select the subset of events in the case of partial DLF theorems generation. This requires a small effort to construct it.

4.4 Implementation Issue

The idea of generating a DLF theorem is very simple. Rodin is an open-source platform implemented with the Eclipse framework. We just need to develop a small external plug-in in Java. However, we discovered a difficulty tied to the development state of a model when the generator is used.

During Development One advantage of using Event-B to model a system is that we construct it step by step with the refinement technique. At each refinement, POs are generated for the newly introduced elements. Discharging these POs is the minimum condition to start a new refinement.

If a system needs to be free of deadlock, we must discharge the POs generated for the DLF theorems at each refinement. Our tool is easy and efficient enough to be used at each refinement.

After Development Modifying a refinement becomes more time-consuming for an existing Event-B model. After a modification is made on a machine, three tools (the Static Checker, the Proof Obligation Generator and the Prover) are automatically called on the modified machine and on all its refinements. The build time increases quickly with the model size. Notably, many formerly discharged POs are re-marked as "Undischarged", thus entailing plenty of repeated re-proving works. To overcome this, the DLF theorems must be generated on an independent machine cloned from the machine we want to edit. Only incremental POs for the cloned machine are generated by the Rodin tools. The status of existing POs is left untouched. Using cloned machines is a more efficient strategy to check for the deadlock-freeness property of an existing Event-B model.

4.5 Usage

After the generator of DLF theorems is installed on Rodin, a button "**DLF**" is displayed on the tool-bar. Generally, we apply this tool in four steps:

1. open a machine and click the button "**DLF**", then the user interface shown in Figure 4.1 comes out,
2. select the DLF theorem type (a list of events can be selected for the partial DLF) and the target machine;
3. click the button "**OK**" to generate the concerned DLF theorem;
4. discharge the POs of the DLF theorem in the target machine with the provers.

Our tool must be seen as complementary to the execution of models to study deadlocks and their absence. On the one hand, a DLF theorem of a thousand lines can be generated in a few seconds, however, its proof can require a lot of resources. When a deadlock is present, the analysis of the failure of the proof gives us a deep understanding of the missing invariants or erroneous guards that affect the model. On the other hand,

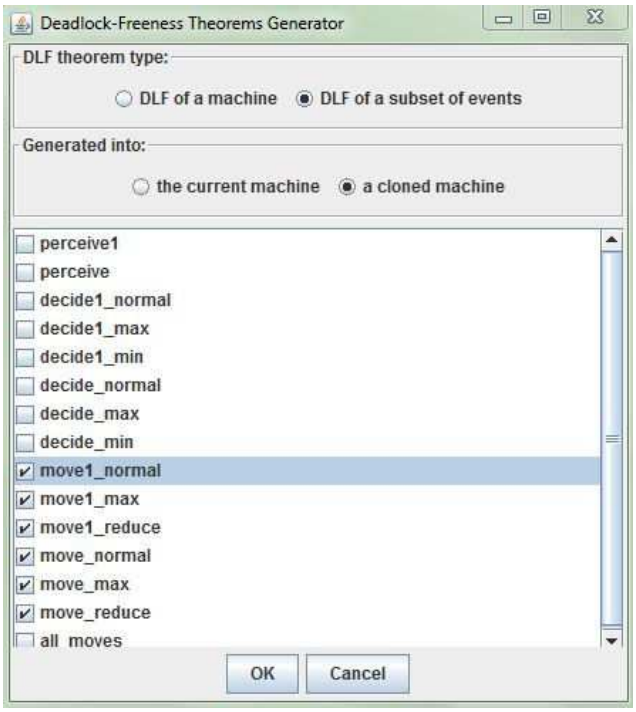

Figure 4.1: Generator of DLF theorems

Machine	Proofs of DLF	Animation/Simulation Behavior
platoon0	discharged	normal
platoon1	discharged	normal
platoon2	No	deadlock
platoon3	No	deadlock
platoon4	No	deadlock

Table 4.2: Deadlock-Freeness in the reviewed 1D platooning model

execution techniques help us to assess quickly the presence of deadlocks. They provide us with an intuitive perception of the causes of deadlock. Table 4.2 shows the relation between the two approaches on the reviewed 1D platooning model.

Neither execution nor DLF theorems generation is the best and only tool to tackle the problem of deadlock-freeness. Their respective use depends on the size of the model and the cost in the development.

4.6 Summary

The lack of means to automatically construct the DLF theorems in Event-B is one of the limitations to generally use this method in a software development. Our tool is a practical attempt to provide specifiers with a simply way to express a very important property.

A very positive aspect of our experience is the assessment of the maturity of Rodin. The development effort we had to put into the project was reasonable. Of course, the first use of the Rodin APIs requires an important learning effort. But this investment pays back afterwards since the API is easy to use and quite powerful. It is then reasonable the try to overcome limitations of Event-B by developing small assisting tools rather than waiting major evolution of the formal framework.

Discharging the proof obligations of DLF theorems is of course the biggest time-consuming activity. The size of the generated formulas, while not a surprise, still raises the issue of their manageability. The problem is actually not specific to our tool but comes from the formal core to Event-B. Assessing formal properties of large B models [Leuschel 2011] is a very active research domain.

An experimental version of DLF theorem generator is accessible at the website http://dedale.loria.fr/?q=en/plug-in-dlf-generator.

Chapter 5

Scaling Up with Event-B

Contents

5.1 Introduction

This chapter relates our experience with the modeling of a realistic platooning algorithm [Daviet 1996] for the control of autonomous vehicles in Event-B.

The 1D model (see Appendix D) is a simplified version of the problem. This model was important on three respects: it allowed us to identify the "hard" parts, it prototyped the properties of interest, and it provided us with a neat structure for the development.

The 2D model (see Appendix E) considered the platooning problem from a realistic point of view. Going from a 1D model to a 2D model do not seem like a big change. Instead of one value, the control law must now compute two values: linear acceleration and derivative of the curvature. However, working on a plane introduces notions such as trigonometric functions and curvature. While this means a modest increase in the complexity for a mathematically literate person, those new concepts introduce genuine

difficulties for the specifier; e.g., how to model a sine function when co-domains are restricted to integers?

In this chapter, we discuss some scaling-up issues (model structure, equation adaptation, temporal properties and adaptation of tools) when going from a 1D model to a 2D model. We could solve some issues, concerning the consistency between refinements and the adaptation of the mathematical model. Other issues can be solved but at a higher cost, proving global temporal properties is among them. Last, the existing animators failed to execute the 2D platooning model.

Notice that we have no intention to extend the Event-B language, the scaling-up issues only focused on the existing available tools for the Rodin platform.

5.2 Model Structure

An important question when we started the 2D modeling was: can we keep the same development structure as for 1D modeling? We had two reasons. First, a great deal of effort had been put into it so it is intelligible, consistent with the general MAS (Multi-Agent System, a system composed of multiple interacting agents within an environment) model [Ferber 1996], and then easy to validate. Second, we can expect proof structures (and even whole proofs) to be similar if we keep the same development structure.

We have been able to keep the exact same structure of the development. The same refinements with the same rationales are present in both models. In fact, we used the 1D platooning model as a "road-map" to develop the new model.

Both models have the same structure depicted in Figure 5.1. It consists of an abstract machine and four refinements. Each development introduces a clearly identified concept. The abstract machine `platoon0` sets the safety property. The refinement `platoon1` splits the platoon's movement into movements of each vehicle. The refinement `platoon2` implements the physical reaction laws. The refinement `platoon3` introduces the decision law to compute acceleration. The refinement `platoon4` introduces the perception of the environment and implements the decision laws.

5.2.1 Decomposition of Events

During the refinement, the number of events increased much more steeply due to the combination of bounded parameters.

Both models implement the reaction laws in the machine `platoon2` and the decision laws in the machine `platoon4`. We need to decompose the abstract events `move1` and `move` in the machine `platoon1`, and the abstract events `decide1` and `decide` in the machine `platoon3` into more concrete ones. Let us consider the decomposition of the event `move`.

The mathematical equation in the 1D model indicates that three cases must be considered when computing a new state. This is due to the fact that the speed (n_speed_i) is bounded.

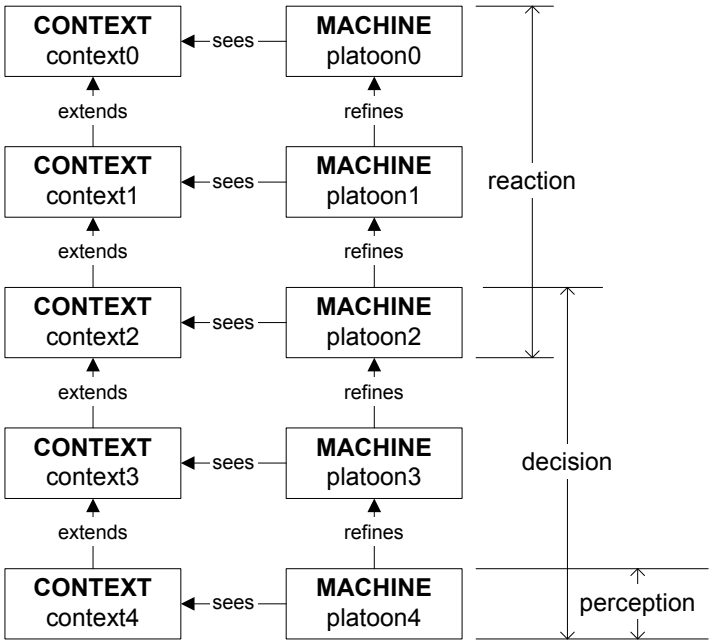

Figure 5.1: The structure of platooning models

In Event-B, conditional definitions are expressed by the use of guards. This means that the `move` event must be decomposed into three events by a bounded parameter n_speed_i, one for each situation (n_speed_i reaching lower bound, reaching upper bound, or within bounds), following the template in Table 5.1.

n_speed_i	< 0	$\in 0..MAX\_SPEED$	$> MAX\_SPEED$
move	*move_reduce*	*move_normal*	*move_max*

Table 5.1: Decomposition of the `move` event in the 1D model

In the 2D model, we have to consider two bounded parameters: a speed (n_speed_i) and a curvature (n_κ_i).

$$\begin{cases} n\_speed_i & = speed_i + accel_i.\Delta t \\ n\_\kappa_i & = \kappa_i + \chi_i.\Delta t \end{cases} \tag{5.1}$$

The analysis must then take into account the combination of three cases for n_speed_i and three cases for n_κ_i. So, the `move` event is refined into nine following the template of Table 5.2.

$n\_speed_i \setminus n\_\kappa_i$	$< -MAX\_\kappa$	$\in -MAX\_\kappa..MAX\_\kappa$	$> MAX\_\kappa$
< 0	*move_vmin_κmin*	*move_vmin_κ*	*move_vmin_κmax*
$\in 0..MAX\_SPEED$	*move_v_κmin*	*move_v_κ*	*move_v_κmax*
$> MAX\_SPEED$	*move_vmax_κmin*	*move_vmax_κ*	*move_vmax_κmax*

Table 5.2: Decomposition of the `move` event in the 2D model

5.2.2 Increase in Complexity

The multiplication of events depicted above happened a few times. It should be noted that other refinement strategies for the abstract event `move` could have been chosen. For instance, we could have kept the refined event unique, but at the expense of very complex guards. Our trade-off lengthens the specification text and increases the number of proof obligations but each proof is much simpler.

Table 5.3 shows the increase in complexity when passing from the 1D platooning model to the 2D platooning model according to the number of model elements and POs.

	1D initial model	2D model
Sets	0	1
Constants	15	50
Axioms	27	86
Variables (last refinement)	10	16
Invariants	16	29
Events (last refinement)	15	39
Guards (last refinement)	81	354
Theorems	1	46
Variants	3	3
Automatic POs	187	743
Manual POs	29	177
Reviewed POs	4	0
Undischarged POs	0	0
Total POs	220	920

Table 5.3: Increase in complexity

While the number of variables roughly doubled when going to the 2D model, all other measures varied by a four-fold increase. Interestingly, the ratio between manually and automatically discharged proof obligations increases just a little: most of the new proof obligations are simple ones. The most important increase is the number of theorems. They are used to ease the proofs by introducing only once standard mathematical properties. This is a consequence of introducing more complex arithmetic expressions in the 2D model.

5.3 Physical and Mathematical Equations

5.3.1 1D Equation Adaptation

The formulas in the 1D equation (2.1) (see Page 25) are basic arithmetic expressions; they contain no special mathematical functions. The values of $xpos_i$, $speed_i$ and $accel_i$ can easily be modeled as integer numbers. It suffices to choose a system of units small enough to reach the accuracy needed in practice. Hence, they can be expressed straight away in Event-B.

5.3.2 2D Equation Adaptation

By contrast, a simple look at the 2D equation (2.2) (see Page 26) shows that we need to transform it to be able to expressed it in Event-B. The most obvious "problems" are the sine and cosine functions (meaningless on integers) and the integral functions in the definition of F_C and F_S expressions within the 2D equation.

The Discretization Issue The heart of the difficulty lies in the discretization of continuous kinematic values such as position, speed or acceleration. The question is then: why not use continuous values? We are not ready to answer positively for two reasons.

The first reason is practical. Current provers within the B world consider only integer numbers. Even with these "simple" numbers, proofs are often complex and intricate. It is not clear that provers doing a good job with real numbers will be available soon. In [Burdy 2012], some experiments have stated with the Atelier B tool which clearly require an evolution of the B language and an extension of the tools to support real numbers.

The second reason is deeper. Software systems are inherently discrete. Because of numerous latencies in the autonomous car (sensing data, computing controls, driving actuators), the control system will operate at a rather slow frequency. So, the actual system will run as if time is discrete.

The B formal method aims at producing a code which is proven to maintain functional invariants. So, we need to introduce the discretization at some point. In [Abrial 2012], three approaches to handle the continuous action system are proposed with Event-B: (i) discrete variables together with continuous variables, (ii) discrete systems as an abstraction of continuous ones, and (iii) refining a discrete systems into a continuous ones. Because the Event-B language and the Rodin platform are not support the real numbers, these proposed approached use some "cheating" to achieve the proofs, for example, giving some specific integer values to the constants that are normally assigned to some real values.

Our position is that we must introduce this fundamental feature early in the models: as soon as we need "continuous" values in the specification. Of course, we must then develop techniques and strategies to take care of this feature.

Although reasonably simple from a mathematician point of view, the 2D equation cannot be translated directly in Event-B. In practice, we need to "refine" it. We used three heuristics.

(1) Free Physical Units In Event-B, the easiest representation of continuous kinematic values such as position, velocity, acceleration are integer numbers. By keeping the physical units unspecified but homogeneous (e.g., the unit of velocity is equal to unit of distance divided by unit of time), we can adapt the representation to the desired accuracy of the computations. Distances can be millimeters as well as meters, and times can be milliseconds as well as seconds.

(2) Approximate Mathematical Functions The range of sine or cosine in the domain $\mathbb{Z}$ is a three value set: it is not very interesting. To solve this problem, we introduce a special dimensionless constant μ and we consider $\mu\cos\theta$ and $\mu\sin\theta$ instead of $\cos\theta$ and $\sin\theta$. We do the same with F_C and F_S and consider μF_C and μF_S. By choosing a μ with a big value, expressions can be reasonably coded with integers.

Event-B provers know about standard rules of arithmetic but ignore trigonometric or general calculus rules. In order to use the provers, we use Taylor series and identities to transform expressions into arithmetic approximations.

Therefore, the vehicle state $\eta = (x, y, \theta, v, \kappa)$ is represented by a 6-tuple $(x, y, \gamma^\theta, \sigma^\theta, v, \kappa)$. The values of x, y, v and κ are integers; the units must be taken small enough to obtain a good accuracy. The values of γ^θ and σ^θ are also integers which respectively represent $\mu\cos\theta$ and $\mu\sin\theta$. We define a carrier set *Point* to denote the set of all possible vehicle states. The approximate, but accurate, reaction law of the 2D platooning is then:

$$\begin{cases} x = x_0 + (\gamma_0^\theta \tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi) - \sigma_0^\theta \tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi))/\mu^2 \\ y = y_0 + (\gamma_0^\theta \tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi) + \sigma_0^\theta \tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi))/\mu^2 \\ \gamma^\theta = (\mu_C \gamma_0^\theta - \mu_S \sigma_0^\theta)/\mu \\ \sigma^\theta = (\mu_C \sigma_0^\theta - \mu_S \gamma_0^\theta)/\mu \\ v = v_0 + a\Delta t \\ \kappa = \kappa_0 + \chi\Delta t \end{cases} \tag{5.2}$$

where
- $\beta = v_0\kappa_0\Delta t + (a\kappa_0 + v_0\chi)\Delta t^2/2 + a\chi\Delta t^3/3$,
- $\mu_C = \mu - \beta^2/(2\mu)$,
- $\mu_S = \beta - \beta^3/(6\mu^2)$,
- $\tilde{F_C}(\Delta t, v_0, \kappa_0, a, \chi)$ and $\tilde{F_S}(\Delta t, v_0, \kappa_0, a, \chi)$ expanded as Taylor series which represent μF_C and μF_S, respectively.

Event-B translation (see Appendix E.3) is straightforward but yields overly long expressions.

(3) Check and Rewrite Mathematical Formulas for Provability Many properties of formulas on real numbers can be safely assumed when we restrict their use to integer numbers, but not all, e.g., the equation (3.2) (see Page 33). In the 1D platooning model, we found four reviewed goals that concern the division operator. They may sometimes invalidate proofs. Through the rewriting of formulas (see Section 3.2.2), the proof obligations can be discharged.

5.4 Temporal Properties

Temporal properties can be classified into two categories [Lamport 1977]. Safety properties specify that nothing bad will happen. Liveness properties specify that something good will eventually happen. Event-B does not support the notations of temporal logics. Modeling the temporal properties leads to a big challenge. The thesis [Rehm 2009]

presents some propositions of patterns to handle calendar, times etc, but it lacks a tool to automatically apply these patterns. The paper [Bellegarde 2002] proposes to extend the classic B event systems for the verification of dynamic constraints under fairness assumptions, this approach also lacks an implementation.

In the platooning model, let us take a look of the deadlock-freeness: a safety property which should be guaranteed.

Huge Formula In Event-B, we can construct a DLF theorem (4.1) (see Page 40) to guarantee that the machine should work for ever. This formula works well on small models but does not scale up.

While the construction of this formula is straightforward, it has the drawback of leading to a huge formula. In the 2D model, it would be around 637 lines long in the last refinement (see Table 4.1). A manual construction may introduce a non obviously detectable error. We have presented a DLF theorems generator in Chapter 4.

Failure of Proof The most important difficulty comes when the proof cannot be concluded. The inability to discharge proof obligations is at first frustrating, but on second though, it is an excellent drive toward correctness. By understanding why a formula leads to an unprovable goal, we gain deeper insights into the model itself. Like counter-examples provided by model-checkers, impossible goals are good indicators toward expressing correct properties.

The application of this technique to the platooning model led us to uncover a problem with the invariants: a constraint on the differential speed which was omitted in the definition of the critical distance between two vehicles. It should be noted that we understood the error through a complex reflexion which involved intuitions on the mathematical equation, observation and reproduction of freezes of the system within the simulation, and analysis of the formula. The big size was a real burden.

5.5 Adaptation of Tools

Formal methods depend heavily on automated support. They require long, intricate, and generally tedious chains of reasoning to discharge or establish properties, even trivial ones. This is the nature of formal proof systems. Effective tools are not a “nice addition” to a formal method, but a key factor for its deployment. Event-B is supported by Rodin, a framework which integrates gracefully tools to edit, verify and validate models. Thanks to the rapid evolution of Rodin itself and the set of available plug-ins, larger models and developments are now supported.

5.5.1 Edition

Unlike most programming languages, which utilize the text-plain files to store code, the implementation of Event-B in Rodin uses a database to store the formal texts and proofs in an XML format. This design facilitates the proofs, but raises a problem for large models.

The older default editor in Rodin is the Event-B Editor. It provides a structural edition environment and a pretty-print view for formal models. The increase in size shown in Table 5.3 did not pose any problems to this editor. We can also use a text-based editor such as Camille [Bendisposto 2010] to edit the platooning models.

However, both editors showed their weaknesses (e.g., waiting a long time for a response, out of heap memory error) while editing larger models such as the MIDAS model [Wright 2009b] in which a refinement can have thousands of lines.

Since the version 2.4 of Rodin, the Rodin Editor is the new default editor to support large models. It is based on the same principles as the historical structured Event-B Editor. But newcomers may find it less easy to use.

5.5.2 Verification

The Rodin architecture provides an excellent environment for theorem proving activities. Various external provers are available to use. The Atelier B Prover [Cle 2009a] is the unique external prover at the time the platooning specifications were developed, its capability to deal with arithmetic proofs was improved since that moment, now the interactive proof described in Section 3.2.1 is automatically discharged by this prover. The external provers such as Isabelle and SMT Solvers are well integrated into the Rodin platform.

These provers were able to deal with the increase in complexity. In fact, the general strategy of breaking a verification proof into several smaller proof obligations as used by B spreads the increase in complexity on much more proof obligations, but each one remains reasonably simple. Of course, the theorem proving requires some experiences and skills. A good Event-B specification normally has more than 80% automatic proofs. This ratio is influenced by some factors, such as the refinement strategy, the prover used and the auto/post tactics for proving. In the 2D platooning specification, we use many theorems to easy proofs. This strategy increased the amount of POs, but also improved the ratio of automatic proofs.

5.5.3 Validation

A verified platooning model preserves the non-collision invariants, however, it must also be free from deadlocks to prevent actual collisions. We were interested in the observation of deadlocks in the two models through the validation activities.

We used intensively animation to understand the collision problem in the 1D platooning model. In particular, animation helped us to understand which values lead to deadlocks. From those, we could abstract to general configurations, and then relate to parts of the deadlock-freeness theorem. The positive experience with the 1D model development induced us to use animation early in the 2D model development. Unfortunately, animators failed us even on the first abstract model.

We tried with two different animators, Brama [Servat 2007] and ProB [Leuschel 2008]. In both cases, the notion of *Point* (5.2) (see Page 50) was the first visible obstruction. We needed to code it, either crudely as integers with Brama or more abstractedly as symbols with ProB. Either way, a list should be provided by hand. This is not realistic and even meaningful.

The second obstruction is that the 2D model uses several uninterpreted functions which are only defined by some properties. The analytic definition of those functions is delayed until the model reaches an implementable state. Although we could provide some adequate external implementations of these functions, we have no way to link them to the animators.

This can be explained by two important features of Event-B: non-determinism and definition of values by their properties. Animators are then more oriented toward "picking" values rather than "computing" values. This orientation is fine most of the time, but it should not be exclusive. Sometimes, even in abstract models, we know some expressions are deterministic computations: kinematic functions are a good example. In those cases, we would appreciate to be able to tell this to the animator.

The failure of animating the 2D platooning model motivates us to develop a new simulation framework, JeB, which is described in the next part. Both 1D and 2D models can be simulated with JeB.

5.6 Summary

Our experience with a real-world model is both reassuring and worrying. This is consistent with our findings on using Event-B for the modeling of the transportation domain [Mashkoor 2010]. On the very positive side: we could model a reasonably complex algorithm and prove its correctness. The *Daviet-Parent algorithm* is a good representative of a class of problems of great practical importance: problems for which we have empirical solutions, some prototype implementations, and a strong need to certify it before we can use it in practice.

Event-B is a good candidate for formal modeling and development of real systems. First, the language has sufficient power to express complex mathematical models and algorithms. Second, the formalism embodies a sound, effective, and easy to use refinement based process. Last, the tool support for edition and verification is up to the task; new validation tools can also be easily to integrate into the Rodin platform. Of course, stronger provers or syntactic sugar-coating to help navigate long texts would be welcome improvements but are not mandatory to make the method usable.

Another reason to be optimistic was our ability to deal with a physical and mathematical model which incorporates complex functions and continuous time. This has required some sweat and efforts, but was never a blocking factor. We think that there are a few "conditioning techniques" that can be used at the mathematical level to put a continuous model in a form suitable to Event-B. We have identified some of them.

The worrying issue lies with the checking of temporal properties, either formally through proofs, or pragmatically through execution. On the formal side, the current situation is not adequate. Only "coarse" properties can be expressed and even then, awkwardly. This is clearly an area where research is needed. On the pragmatic side, we need better execution tools. Such tools have a very important property: they act on the formal model itself. We can be confident that the observation is based on the model's behavior. Hence we integrated the simulation tool JeB into the Rodin platform.

Part II

JavaScript Simulation Framework for Event-B

Chapter 6

JeB Design

Contents

6.1 Introduction

The correctness of a piece of software requires two quality insurance activities: verification – Have we built the piece right?– and validation – Have we built the right piece? For a long time, research in formal methods focused rightfully on formal frameworks and their supporting tools. As the supporting tools for the verification of formal models became mature, the tools for validation activities become an important issue. Validating a formal specification is to assess whether it describes the right problem in a given situation. A common opinion is that the cost to correct a mistake introduced in an early stage of the development is much higher than the cost for a mistake in a latter stage. A forgotten or badly specified requirement incurs the highest cost. Then, there is a clear interest to validate formal specifications as early as possible.

We think that validation should be organized like verification: along the refinement chain. In principle, the validation is simple: experts or potential users read a formal specification to judge if it represents a good model of the expected system. In practice, this does not work well since we, as humans, are not very good at reading and analyzing

long mathematical texts. A better way is to look at executions of a specification and to judge whether the observed behaviors are consistent with the expected ones [Balzer 1982, Fuchs 1992]. The problem then becomes technical: how to execute the model?

An Event-B formal model can be executed in certain conditions. Within the Rodin platform, several *animators* and *translators* have been developed to execute Event-B specifications. Unfortunately, the class of directly animatable specifications is limited. The major cause of failure is the non-determinism of abstract specifications which leads to the same kind of combinatorial explosions faced by model-checkers.

In animators, like Brama, AnimB and ProB, the executable representation of the model is internal to the tool; users cannot access it directly. As their main advantage, these tools have the shortest path between the model and the observations of system behaviors. But these tools often failed to animate a non-deterministic Event-B model which contains abstract types, uninterpreted functions or large domains. In this case, the user has no other choice except than modifying the original model to adapt it to the tool. We can do some safe transformations [Mashkoor 2009b] to extend this class. However, many specifications are still non-animatable.

To extend the class of executable specifications, we can generate a prototype of the model in a programming language. Translators, like B2C and B2ALL, transform Event-B models into programs written in a mainstream language such as C or Java. The automatic generation of code brings more flexibility to the users. For some complex formulas, users can find executable solutions by modifying some part of the generated code in order to execute it. But these translators have two main limitations: a) the translation is limited to a subset of the Event-B notations, restricted on the last deterministic models; b) they have no default graphic execution environment to display the evolution of states and to interact with users.

JeB [1] , A **J**avaScript simulation framework for **E**vent-**B**, is based on an observation and a simple idea. We observed that while some specifications are hard to animate, we could easily write programs to emulate them. The explanation is that using non-animatable features such as non-determinism and high abstraction in early refinements is recommended, even when we know how they will be reified. The idea was then to associate developers, domain experts and final users in the process of constructing simulations, so their intelligence would overcome the difficulties in the few cases where automatic solutions could not be applied.

6.2 Requirements for a Simulation Generator

The analysis of the limits of the existing tools discussed above leads to a set of requirements for a new technique and its supporting tool: simulation. They are summarized in Table 6.1.

REQ-1 captures the idea that we want some validation activities along the refinement chain, in parallel with the verification activities.

1. The latest version of JeB is accessible at http://dedale.loria.fr/?q=en/JeB

Requirement	Description
REQ-1	The simulation can be applied all along the refinement chain, from the abstract machine to the last refinement.
REQ-2	The construction of a simulation must be a reasonably easy and low-cost activity.
REQ-3	All Event-B mathematical notations should be supported.
REQ-4	The simulation must be consistent with an operational interpretation of an Event-B model; it must allow users to observe as many behaviors as possible on the simulated model.
REQ-5	An interface for integrating user hand-coded functions should be provided.
REQ-6	Users should have full control over the simulation, in particular, they should be able to provide the non-deterministic values when needed.
REQ-7	An easy-to-use graphic user interface should be provided.
REQ-8	Users should be able to create their own graphic presentation of the state and its evolution.

Table 6.1: Requirements for a simulation generator

REQ-2 is for cost-effectiveness. The cost of using a validation tool for each (or most) refinement must be balanced against the cost and the risk of limiting the validation to the last step of the development. Decreasing the cost of using a validation tool is increasing its use.

REQ-3 is about the completeness of the tool. Brama and AnimB do not support some Event-B mathematical notations, B2C and B2ALL only support a subset of Event notations. These tools often require that the specification has to be modified before they can be used. Our strategy is to let the automatic tool work with the actual specification and push the adaptation to the generated operational model.

REQ-4 is concerned with the correction of the simulation. The purpose of validation is to check that certain behaviors are possible and that certain others are impossible. So, to be useful as a validation tool, a simulation should guarantee that if a behavior is observed then, it is allowed in the model.

REQ-5 is the extensibility of the translated mathematical model. The uninterpreted functions defined in an early context need an implementation by a human in the generated simulators. Some complex Event-B mathematical formulas, like nested quantifications and non-deterministic assignments, require users to provide their own implementations to override the default generated functions for better efficiency.

REQ-6 is about the control of simulations. The users should have the capability to decide which event among the enabled events can be fired. The users should have the capability to input values for the parameterized events in a simulation cycle.

REQ-7 is about the interaction between the simulators and the users. The user interface should be graphic and simple to use.

REQ-8 is about the visualization of the model's state. For complex systems, it is often necessary to provide users with a specific and ad-hoc graphical representation so they can analyze the behaviors.

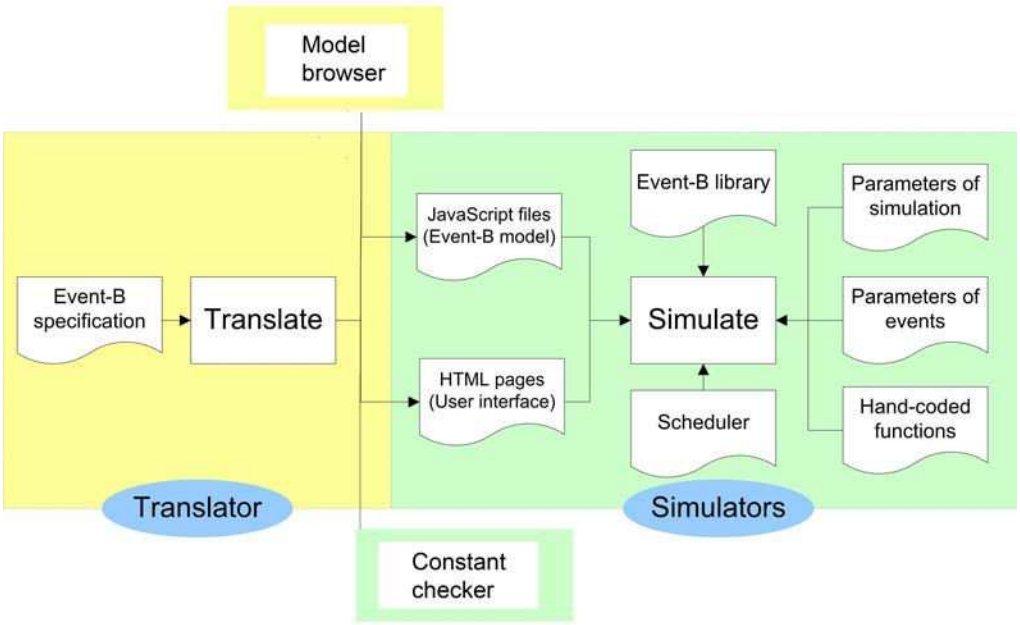

Figure 6.1: JeB simulation framework

6.3 Architecture of the Simulation Framework

Running a simulation of an Event-B model is a process which goes through several steps. First, the machines and contexts are translated into the executable target language. Then, the users complete the generated program by providing some implementations for the elements that could not be treated by the translator. Last, the users set-up and drive each simulation run by providing values for the constants in the contexts, choosing values for the non-deterministic traits, and picking the event to fire at each cycle. The last two steps are highly dependent on the runtime environment used for the simulations.

The architecture of JeB reflects this process. An important decision was to use Web browsers as the runtime environment. The rational will be discussed in the next section. An obvious consequence is that HTML and JavaScript become the target languages.

JeB's framework is schematized in Figure 6.1.

The most important part of JeB is the translator and the simulators.
- The translator is installed as a Rodin plug-in. It automatically generates an executable model in JavaScript and an HTML interface for each Event-B machine.
- The simulators:
  - Each simulator includes an HTML page for the graphic interface and a JavaScript model for the machine specification.
  - The Event-B library is a JavaScript library which provides procedures allowing the execution of Event-B set constructs and first order predicates. It supports all the

Event-B mathematical notations.
- The scheduler controls the user interactions and schedules the execution of all enabled events.
- The different parameters concern the instantiation of the simulations. Some are about the whole simulation, some about the events and some about the user's hand-coded functions.

The major advantages of this framework are its openness and its extensibility. All generated simulators are stored as plain text files and we can use common text editors to customize them. The graphic interface uses the HTML language, which is easily customized to special graphic displays as users demand. The existing JavaScript graphic libraries can also be used. The users can provide their own implementations to override the default generated functions; it is mandatory in some cases in particular when non-determinism occurs. Although this strategy imposes some burden on the users, it should be compared with the problems posed by animators and translators. When those tools fail, users are required to modify the Event-B model itself, typically by introducing a more deterministic refinement. Even when possible, this kind of work is quite complex.

To validate an Event-B model with the JeB simulation framework, we follow a four-step process:

1. generate the simulators from a given Event-B model (automatic),
2. define the values of all constants and implement some user hand-coded functions (manual),
3. set up the graphic display of states (optional), and
4. run the simulators, observe and analyze the behaviors during the simulations.

During the implementation of JeB, it appeared that unplanned, but very useful, tools could be realized as by-products:
- a model browser which allows users to navigate complex models made of numerous refinements,
- a constant checker which allows users to assess the validity of the values bound to the constants defined in contexts.

6.4 Implementation Choices

The challenge of the JeB simulation framework is to provide a practical tool which is cheap and powerful enough to be used on all refinements. Our first version of the translator [Yang 2012] used C as a target language. The generated simulators needed to be compiled and executed in a text console with many interventions. To achieve a graphic execution environment, we used a server/client model. Each simulator ran in a server mode, an extra graphic client got data from the simulator then displayed them. This was not a practical and efficient solution.

Inspired by the capabilities of JavaScript/HTML when using them to create the graphic clients, we finally switched to JavaScript/HTML as the target language for simulators. Their major qualities satisfied our purpose:

– First, JavaScript is embedded in a Web browser and interpreted directly. There is no need of an extra compilation step before an execution. With HTML, it is easy to generate a lightweight and efficient user interface as a standard web page which can be modified and extended as user requires. The quality of the simulation interface is essential to observe and analyze detailed behaviors.
– Second, JavaScript is an object-oriented language which allows us to use sophisticated object programming techniques. This helps to solve two issues. One, technical, concerns name conflicts which are a common plague in an automatic translation. The other, practical, concerns the relation between the Event-B specification and the generated JavaScript programs. Using objects allows the JeB translator to produce a code whose structure is close to the Event-B text structure. It facilitates the navigation in the code when there is a need to inspect it for specific observations.
– Third, JavaScript uses dynamic interpretation that allows users to provide their own functions to override the generated ones. We use this feature to separate the user hand-coded functions from the main simulator code. Users only need to modify some configuration files in which they replace a function by their own implementation.
– Fourth, JavaScript has first-class functions, which provide us the capability to translate nested quantified predicates and expressions in a systematic way.

The *raison d'être* of refinement-based methods is to allow developers to master the correctness of complex systems by introducing slowly the implementation constraints. Determinism or actual data-structures are not likely to be found in early refinements. Nevertheless, when specifiers introduce some abstraction, they have generally a clear conception of how it could be implemented. The architecture of the JeB framework integrates our philosophy that a simulation is a collaborative process between automated tools and humans. Developers, domain experts and final users are involved at three levels:

– developers provide hand-coded functions for some uninterpreted functions and an efficient implementation of some generated code,
– domain experts provide the values of constants, event parameters in different scenarios, and give the suggestion to set up extra graphic visualizations of states,
– final users run the simulations, observe and analyze the behaviors to validate an Event-B model according to their expectations.

6.5 Translation Strategies

This part gives the main translation principles.

6.5.1 Annotations vs. Set Library

In the early development of the translator, we used some annotations [2] as hints to the translator in order to generate the concrete data types of the target language. But we

2. They are marked as comments in a formal text.

quickly found out that this is not a good solution: the native Event-B mathematical objects are based on set theory, therefore we cannot construct a simple mapping between Event-B and a classical programming target language. This limitation of using annotations motivated us to develop a set library which supports the Event-B set notations. Eventually this set library has evolved into a complete JavaScript library which supports all Event-B mathematical notations, such as predicates. With this design, the translator just maps an Event-B concrete syntax tree node to an API defined in that library. So annotations are not needed anymore for the current translator.

6.5.2 Interfaces for User Hand-coded Functions

JavaScript has first-class functions that can be assigned to variables and manipulated like any other object. It uses a dynamic interpretation, so only the last definition of a function will be considered. These features facilitate us in providing an interface for the user hand-coded functions. JeB provides interfaces for predicate formulas (axioms, invariants, guards) and assignment formulas (actions). These elements are translated into calls to specific functions. Users can easily find a translated function according their labels in an Event-B model and the namespace structure, then use a function with the same name to override the generated one in the user configuration files.

6.5.3 Invariant, Witness and Variant

When the abstract variables are replaced by concrete variables in a refinement, Event-B uses a gluing invariant to link two successive models. This is needed to to ensure that a refinement is consistent with the machine that it refines. For the same reason, when an event parameter disappears in the refined events, Event-B uses a witness statement to tell the machine how the abstract parameter should be refined. The variant is an expression which imposes the non-divergence of the new events, it is a tool to prove the termination of a chosen set of events. The role of gluing invariants, witnesses and variants is to derive proof obligations to ensure the consistency between refinements.

Unlike the multi-level animation of refinements in Section 2.5.3, we focus on the behaviors of the current refinement, the consistency of refinements should be well guaranteed by the associated proof obligations. This decision simplifies the design of the JeB translator. Therefore in JeB, each machine is translated to an independent simulator which is executed in a standalone environment. There is no links between them. But the abstract variables in the gluing invariants and the abstract parameters in witnesses become undeclared identifiers in the concrete translated simulators which raise errors when running the simulation. Notice that the gluing invariant, witness and variant have no influence on the behaviors. So the JeB translator does not translate them.

6.5.4 Quantified Formulas

The major difficulty of translating Event-B formulas is connected with quantified formulas. In Event-B, the quantified formulas exist in quantified predicated (universal quantification and existential quantification), quantified expressions (set comprehension, lambda, quantified union and quantified intersection), and non-deterministic assignment "*becomes such that*". They can also be nested. Our first idea was to use pattern matching to identify typical cases for which an efficient translation is available. However, it failed for most cases. Then we choose to include quantified notations in APIs in the JavaScript library for Event-B. So the difficulties to deal with quantified formulas are transfered to the implementation of that library. According to their defined APIs, the JeB translator outputs some wrapped functions for the quantified formulas, and outputs an enumerated domain set for each quantified identifier (see Appendix B.8.1). With this strategy, we can translate all nested quantified formulas in a systematic way. Of course, users can provide their own implementation using the interface for user hand-coded functions.

6.6 Summary

The application of formal methods deeply depends on the development of tools. While tool support for the verification of formal models is becoming mature, the validation becomes an important question. One method for validating formal models is to execute the model and observe its behavior. Current execution tools for Event-B, translators are used to generated the executable code from the last deterministic refinement; animators provide users with automatic execution mechanisms of the models, but their use is limited to certain type of models. So, we propose a practical tool, JeB, which allows us to simulate any Event-B model for the activities of visualization and validation.

Its design philosophy is that the construction of the simulation can be designed as a collaborative work between the tool and the user. While the translator automatically produces the main bulk of the work, the user provides the ad-hoc solutions on the few elements which block the simulations.

Four Event-B models served as guidelines or testbeds to develop the JeB tool. Through these models, we experimented different design and implementation strategies. As a result, there is no need to adapt or modify an Event-B model in order to execute it. This is a major progress comparing to the existed animation and translation tools for Event-B models.

Chapter 7

JeB Implementation

Contents

7.1 Introduction

One of design goals of the JeB framework is to facilitate the analysis of Event-B models by the specifiers. So, we implemented three tools:

– a project browser which displays the general structure of an Event-B project and helps to navigate it,
– a constant checker which allows users to verify that the entered values for constants are consistent with the axioms,
– a machine simulator which allows users to execute the model.

The three tools are generated directly from the RODIN internal representation of the models, which is done by straight traversing of the structure and the formulas of the machines and contexts. The generation process produces JavaScript and HTML code.

The execution mechanism relies on a specific JavaScript library. This library provides an interface which mimics most of the logic and set operators found in Event-B expression. This is consistent with the JeB design philosophy: the implementation of the JeB translator is independent to the implementation of the JeB simulator. Here, we gave a default implementation of this library which is composed of two JavaScript files: a `set.js` file for the Event-B language and a `jeb.js` file for the simulation controller. These two files are copied to each simulated Event-B project which can be found in the directory `jeb` created in the project's workspace. Users can modify them to provide their own simulator implementation which will override the default one.

The generated code is structured so that users can easily insert their own code. This facility is provided so that non-executable and non-deterministic traits can be replaced by more deterministic and executable instances.

In the following sections, we give technical details about our implementation. This implementation is only a prototype. We focus on the feasibility of JeB implementation by some simple algorithms, the performance and efficiency are not considered in this thesis.

7.2 Namespace

Managing names is always a tricky part when translating toward high-level languages. We must avoid name conflicts among the translated names and with the names already used in the environment. Furthermore, we want users to be able to trace back a JavaScript name to its counterpart in Event-B. While JavaScript does not have a notion of namespace in the strict sense, identical benefits can be obtained by creating a global object and adding all objects and functions to this object. A short prefix is associated with the common "namespaces" listed in Table 7.1.

The following examples show how to access elements using our namespace scheme; let `i1`, `e1` and `g1` be the internal references for the translated invariant, event and guard object instances:

Namespace	Prefix	Description
jeb	jeb	the top namespace
jeb.constant	$cst	constant identifiers
jeb.axiom	$axm	axioms
jeb.variable	$var	variable identifiers
jeb.invariant	$inv	invariants
jeb.event	$evt	machine events
$evt.<eventId>.arg	$arg	event parameters
$evt.<eventId>.guard		event guards
$evt.<eventId>.action		event actions
jeb.util		utility
jeb.ui		user interface control
jeb.animator		animator for state
jeb.scenario		simulation scenario
jeb.scheduler		simulation scheduler
jeb.lang		Event-B structure mapping
$B		JavaScript library for Event-B
$anim		animator canvas

Table 7.1: Namespaces used in the simulator code

```
$var.xpos          // access the variable "xpos"
$inv.i1.predicate  // access the predicate of the invariant "i1"
$evt.e1.guard.g1   // access the guard "g1" of the event "e1"
$arg.magic_xpos    // access the parameter "magic_xpos"
```

7.3 Translation of Contexts

For each Event-B context, the JeB translator generates a JavaScript file that contains a translation of the axioms, and an HTML page (which is a constant checker). Using a web browser, users can then check the consistency of the values provided for the sets and the constants by the constant checker. The structural mapping of a context from Event-B to JavaScript/HTML is shown in Table 7.2.

Event-B	JavaScript/HTML
CONTEXT `name`	(1) `name.js` (context model) (2) `name.html` (user interface)
EXTENDS `context_names`	list of `context_name` in the web page for navigation
SETS `identifiers`	list of `$cst.identifier`
CONSTANTS `identifiers`	list of `$cst.identifier`
AXIOMS `predicates`	list of `jeb.lang.Axiom` instances
END	

Table 7.2: Structural mapping of a context

7.3.1 Sets and Constants

The sets and constants are translated as standard identifiers prefixed by `$cst`. Users can instantiate them in the global configuration file `jeb_user.js` (used for a project) or in the particular configuration file of a given machine, `machineName_user.js`.

Contexts in Event-B reference two kinds of sets: enumerated sets and carrier sets. Users can instantiate enumerated sets by collecting elements already defined in the `CONSTANTS` clause. For carrier sets, the JavaScript library for Event-B provides the capability to instantiate them either as a collection of integers, symbolic strings or as a `class` in the object-oriented sense.

7.3.2 Axioms

JeB translates each axiom as an instance of `jeb.lang.Axiom`, prefixed by `$axm`. This structure has several properties: `id` is a unique reference to the axiom, `label` contains the axiom's name written in the context, `domNode` links the axiom's instance to the HTML page to display its evaluation. The most important property is `predicate` which is a method to evaluate the translated predicate. In JavaScript, this structure looks like:

```
jeb.lang.Axiom = function( id, label ) {
    this.id = id;                       // axiom id
    this.label = label;                 // axiom label
    this.domNode = jeb.ui.$( id );      // axiom DOM node
    this.predicate = function(){}       // overridden by the JeB translator
    ...
};
```

7.3.3 Constant Checker

For each context, a page presenting like in Figure 7.1 is generated. After sets and constants have been instantiated, users can utilize this page to ensure that the actual values satisfy the axioms. The page will alert users when a referenced constant has not been instantiated with an actual value.

7.4 Translation of Machines

The output of a machine translation consists of three files: a JavaScript file which contains the executable version of the model, an HTML file to control the simulations, and a JavaScript machine-configuration file. To be executed, the translated model requires a JavaScript library. The machine configuration file contains the functions used to generate random values for the events' parameters or stubs when no automatic generation is possible. Users use this file to construct a particular simulator by replacing some argument generator functions and implementing the stubs.

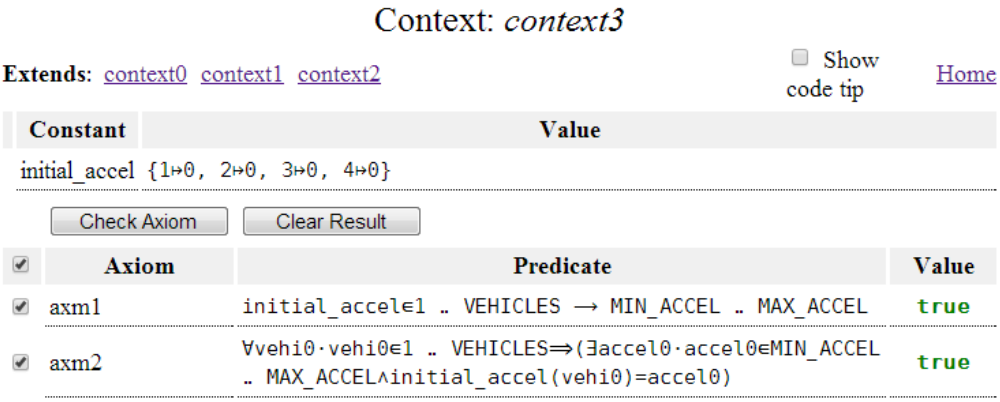

Figure 7.1: Constant checker

The structural mapping of a machine from Event-B to JavaScript/HTML is summarized in Table 7.3

Event-B	**JavaScript/HTML**
MACHINE `name`	(1) `name.js` (machine model) (2) `name.html` (user interface) (3) `name_user.js` (configuration file)
REFINES `machine_names`	list of `machine_name` in the project diagram page
SEES `context_names`	list of `<script type='text/javascript' src='context_name.js'></script>`
VARIABLES `identifiers`	list of `$var.identifier`
INVARIANTS `predicates`	list of `jeb.lang.Invariant` instances
VARIANT `expression`	not (yet) translated
EVENTS `labels`	list of `jeb.lang.Event` instances
END	

Table 7.3: Structural mapping of a machine

7.4.1 Variables

Each variable is translated as an instance of `jeb.lang.Variable`, prefixed by `$var`. The JavaScript structure `jeb.lang.Variable` is defined as:

```
jeb.lang.Variable = function( id ) {
    this.id = id;                       // variable id
    this.value = undefined;             // variable value
    this._value = undefined;            // variable primed value
    this.domNode = jeb.ui.$( id );      // variable DOM node
    ...
};
jeb.lang.Variable.prototype.updateView = function() {
    this.domNode.innerHTML = this.value;
};
```

where `id` is a unique reference, `value` is assigned the current value, `_value` is assigned the previous value as defined by the prime symbol ($'$) in the before-after-predicates, `domNode` binds to an element in the HTML page to display the value, and `updateView` is the method used for updating the user interface.

The value of a variable can be any Event-B mathematical object. It is checked against the preservation of the invariant after each simulation cycle.

7.4.2 Invariants

Invariants are translated as instances of `jeb.lang.Invariant` with a prefix `$inv`. The JavaScript structure `jeb.lang.Invariant` is defined as:

```
jeb.lang.Invariant = function( id, label ) {
    this.id = id;                       // invariant id
    this.label = label;                 // invariant label
    this.domNode = jeb.ui.$( id );      // invariant DOM node
    this.predicate = function(){}       // overridden by the JeB translator
    ...
};
```

where `id` is a unique reference, `label` is a label given by the specifier in the machine, `domNode` binds to an element in the HTML page to display the invariant, `predicate` is a method to evaluate the translated Event-B predicate.

7.4.3 Events

Events are the core elements of Event-B. Essentially, they are guarded substitutions. The structural mapping of an event from Event-B to JavaScript/HTML is summarized in Table 7.4.

Event-B	**JavaScript/HTML**
`label`	(1) a `jeb.lang.Event` instance (2) an event view in the HTML page
REFINES `event_labels`	not translated
ANY `identifiers`	list of `jeb.lang.Parameter` instances
WHERE `predicates`	list of `jeb.lang.Guard` instances
WITH `witnesses`	not translated
THEN `actions`	list of `jeb.lang.Action` instances
END	

Table 7.4: Structural mapping of an event

In the translated model, each event is an instance of `jeb.lang.Event` prefixed by `$evt`. The structure of `jeb.lang.Event` is defined as:

```
jeb.lang.Event = function( id, label ) {
    this.id = id;                       // event id
    this.label = label;                 // event label
```

```
    this.domNode = jeb.ui.$( id );      // event DOM node
    this.enabled = true;                // activation condition
    this.parameter = {};                // parameters namespace
    this.guard = {};                    // guards namespace
    this.action = {};                   // actions namespace
    this.bindArguments = function(){}  // overridden by the JeB translator
    ...
};
jeb.Event.prototype.updateParametersView = function() {...}
jeb.Event.prototype.evaluateGuards = function() {...}
jeb.Event.prototype.doActions = function() {...}
jeb.Event.prototype.testGuards = function() {...}
```

where `id`, `label`, and `domNode` have the same role as for invariants. The property `enabled` contains the result of the evaluation of the guards. The properties `parameter`, `guard`, `action` are namespaces for the event's parameters, guards and actions, respectively.

The dynamic part of the event consists in five methods:
- `bindArguments`: binds the input values to the event parameters;
- `updateParametersView`: updates the parameter view in the user interface;
- `evaluateGuards`: evaluates this event's guards and sets the property `enabled` to the result of the evaluation;
- `doActions`: executes this event's actions, updates the view of variables in the user interface, and saves the system state to the scenario;
- `testGuards`: repeatedly calls the `bindArguments` and `evaluateGuards` methods until this event property `enabled` is true or it is decided that no value can be found.

7.4.4 Event Parameters

The so-called "event parameters" in Event-B are the variables introduced in the `ANY` clause of events. However, they are used for two different purposes. Deterministic variables are actually local variables. Non deterministic variables are true parameters in the programming sense. Before evaluating its guards and executing its actions, the event instance calls its method `bindArguments` to bind its parameters to the input values in the user interface and to compute the values of local variables. The binding operation allows us to translate the guards and actions into independent functions. When executed, the action method will get the parameter values from the event property `parameter`.

JeB uses a simple algorithm to decide whether an event parameter is a local variable or a true parameter:

1. let `ident` be an identifier declared in the `ANY` clause of an event `Evt`, and `expr` be an Event-B expression;
2. if there exists an equality such as `ident = expr` or `expr = ident` in the guards, `ident` is a local variable;
3. otherwise, `ident` is a true parameter.

Each true parameter par_i is associated with an argument generator function, get_par_i. For highly non-deterministic parameters, the JeB translator only generates empty functions that should be replaced in the configuration files.

Another issue of the translation of parameters is the order of their declaration. The Event-B language is designed for theorem proving, therefore the order of parameter declarations has no effect. However, in the MIDAS case study, we often found that local variables reference some parameter values before these parameters are declared. A naive translation would cause errors when `bindArguments` is called. In such cases, the JeB translator will adjust parameters' order in the user input interface and in the `bindArguments` function according to their occurrence order in an event guard.

Each identifier (true parameter or local variable) declared in an `ANY` clause is translated as an instance of `jeb.lang.Parameter` whose structure is defined as:

```
jeb.lang.Parameter = function( id, type, eventId ) {
    this.id = id;                                  // parameter id
    this.type = type;                              // 1: parameter,
                                                   // 2: local variable
    this.domNode = jeb.ui.$( id );                 // parameter DOM node
    this.domNodeInput = jeb.ui.$( id + '.input' ); // input DOM node
    this.eventId = eventId;                        // reference to its
                                                   // event parent
    ...
};
jeb.lang.Parameter.prototype.updateView = function() {
    this.domNode.innerHTML = this.value;
};
```

where `id` is a unique reference, `type` indicates whether the parameter is local variable or a true parameter, `domNode` binds to an element in the HTML page to display the parameter value, `domNodeInput` binds to an input HTML element so users can enter values during the simulations, `eventId` references the event, and `updateView` is a method for updating the display on the HTML page.

7.4.5 Event Guards

An event guard is a predicate formula. The enabled status of an event is the result of computing all its guards. The JeB translator translates each guard to an instance of `jeb.lang.Guard`. The structure of `jeb.lang.Guard` is defined as:

```
jeb.lang.Guard = function( id, label, eventId ) {
    this.id = id;                       // guard id
    this.label = label;                 // guard label
    this.domNode = jeb.ui.$( id );      // guard DOM node
    this.eventId = eventId;             // reference to its event parent
    this.predicate = function(){}       // overridden by the JeB translator
    ...
};
```

where `id`, `label`, `domNode` have the same meaning as for invariants. The property `eventId` refers to the event and `predicate` is a method for evaluating the translated formula.

7.4.6 Event Actions

An event action is an assignment. All actions of an event compose its substitution which changes the model's state. The translation of actions raises two issues. The first one is the ordering of assignments. The semantics of Event-B requires the assignments to be simultaneous. The second is the nature of the assignment. Event-B proposes three kinds of assignments: "*becomes equal to*" (*act1* and *act2* below), "*becomes member of*" (*act3*) and "*becomes such that*" (*act4*). The first kind of assignments is deterministic, the other kinds of assignments are non-deterministic.

Let a, b, c, x, y, z be variables declared in a machine, S be a set declared in a context. The following actions [1] are defined in an event:

act1 :	$a := b$
act2 :	$b := a$
act3 :	$c :\in S$
act4 :	$x, y :\mid x' > y \wedge y' < x' + z$

The ordering of the actions *act1* and *act2* is a classical difficulty when using a sequential language. Notice that $:=$ and $:\in$ are special cases of the general $:\mid$ assignment. The above actions can be rewritten by the unified non-deterministic form:

act1 :	$a :\mid a' = b$
act2 :	$b :\mid b' = a$
act3 :	$c :\mid c' \in S$
act4 :	$x, y :\mid x' > y \wedge y' < x' + z$

With the above transformation, the primed variables act as intermediate variables, so the ordering issue disappears. The JeB translator uses this implicit unified non-deterministic form. During the execution of actions, the assignments are always made to the primed value (`_value`) of a variable. After all actions have been realized, a copy from the property `_value` to the property `value` is executed for each assigned variable.

For the non-deterministic assignments, we defined an interface for each kind of assignments in the JavaScript library for Event-B which provides a default implementation. Users can override the default implementation with their own.

With the above strategies, each action is translated as an instance of `jeb.lang.Action`. The structure of `jeb.lang.Action` is defined as:

```
jeb.lang.Action = function( id, label, eventId ) {
    this.id = id;                       // action id
    this.label = label;                 // action label
    this.eventId = eventId;             // reference to its event parent
```

1. The primed variables denote the after-values in a before-after predicate

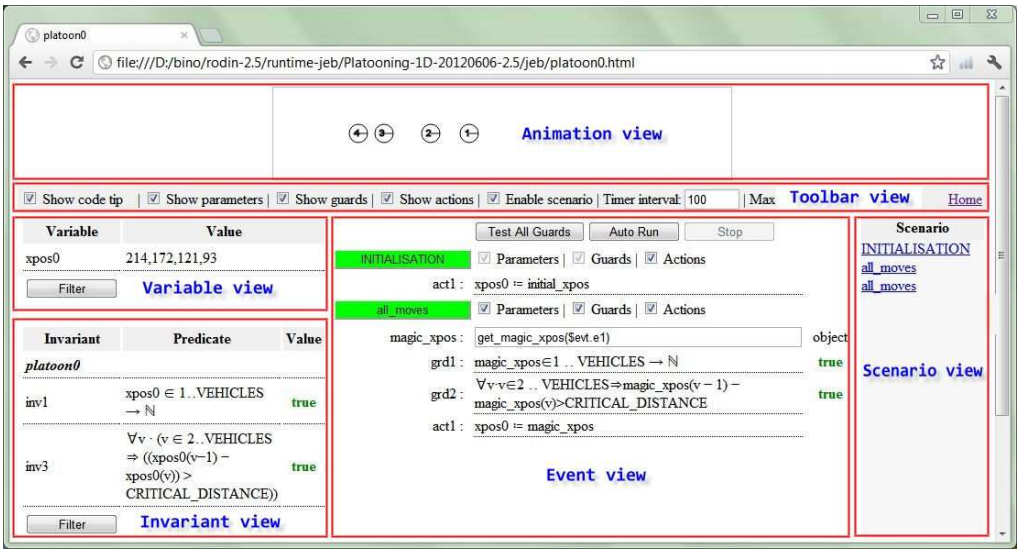

Figure 7.2: A machine user interface

```
    this.assignment = function(){}     // overridden by the JeB translator
    ...
};
```

where `id` is a unique reference, `label` is the label given by the user, `eventId` references the event, and `assignment` is the method to realize the assignment which will be generated by the JeB translator.

7.4.7 User Interface

The JeB translator automatically generates a user interface like an HTML page shown in Figure 7.2. There is one page per machine. When the page is opened for the first time, the user is alerted if any referenced constant has not been instantiated.

The user interface consists of five views:

- Animation view: the top-most area is a graphic view of the state.
- Toolbar view: the toolbar is used to set up general parameters of simulations. Users can customize the page display informations by toggling on or off the check-boxes.
- Variable and invariant views: the left column shows the variables and the invariant [2] and displays their current values.
- Event view: the middle column is the event view which shows all events, their activation status, their parameters, their guards and their actions.
- Scenario view: the right column is the scenario view. The sequence of fired events, with their parameter values, is kept so that the analysis of behaviors can "backtrack" to previous states and explore other choices.

2. The invariant view is useful when simulations are used to "fix" tricky invariant formulas.

7.5 Translation of Formulas

The presentation given in Sections 7.3 and 7.4 is an overview of the structural translation of Event-B components. Axioms, invariants and guards are, by essence, predicate formulas; actions are, by essence, assignment formulas. Such formulas are wrapped into independent JavaScript functions by the JeB translator. Each Event-B formula in Rodin has a concrete syntax tree representation; each node of this tree is marked by a unique tag (an integer constant). To simplify the task of the JeB translator, we developed a complete JavaScript library for Event-B (see Appendix C), hence every Event-B mathematical notation has a unique API definition in that library. The JeB translator recursively traverses the syntax tree, maps each node of this tree to the library API, and finally outputs a JavaScript expression for each formula.

In this section, we give a concise description of the translation for each node, organized by topic. The superscript $(^{TR})$ denotes a traversal of a child node. The superscript $(^{TR*})$ denotes a special treatment of a translation. The symbol $\$B$ denotes the namespace for the JavaScript library for Event-B.

The following tables show that the JeB translator completely supported the Event-B notations. Informally, the hypothesis 3 in Chapter 9 is reasonable. A complete list of translation rules and symbols definitions can be found in Appendix B.

7.5.1 Predicates

Logical primitives

Event-B Notation	Event-B Syntax	Translation
True predicate	$\top$	$\$B.bTrue()$
False predicate	$\bot$	$\$B.bFalse()$

Logical Predicates

Event-B Notation	Event-B Syntax	Translation
Implication	$P \Rightarrow Q$	$\$B.implication(P^{TR}, Q^{TR})$
Equivalence	$P \Leftrightarrow Q$	$\$B.equivalence(P^{TR}, Q^{TR})$
Conjunction	$P_1 \land \ldots \land P_n \ (n \geq 2)$	$\$B.and(P_1^{TR}, \ldots, P_n^{TR})$
Disjunction	$P_1 \lor \ldots \lor P_n \ (n \geq 2)$	$\$B.or(P_1^{TR}, \ldots, P_n^{TR})$
Negation	$\neg P$	$\$B.not(P^{TR})$

Quantified Predicates

Event-B Notation	Event-B Syntax	Translation
Universal	$\forall x_1, \ldots, x_n \cdot P \ (n \geq 1)$	$\$B.forAll(P^{TR*}, [x_1^{TR*}, \ldots, x_n^{TR*}])$
Existential	$\exists x_1, \ldots, x_n \cdot P \ (n \geq 1)$	$\$B.exists(P^{TR*}, [x_1^{TR*}, \ldots, x_n^{TR*}])$

Equality

Event-B Notation	Event-B Syntax	Translation
Equality	$E = F$	$\$B.equal(E^{TR}, F^{TR})$
Inequality	$E \neq F$	$\$B.notEqual(E^{TR}, F^{TR})$

Set Predicates

Event-B Notation	Event-B Syntax	Translation
Set membership	$E \in S$	$\$B.belong(E^{TR}, S^{TR})$
Not a set membership	$E \notin S$	$\$B.notBelong(E^{TR}, S^{TR})$
Proper subset	$S \subset T$	$\$B.properSubset(S^{TR}, T^{TR})$
Not a proper subset	$S \not\subset T$	$\$B.notProperSubset(S^{TR}, T^{TR})$
Subset	$S \subseteq T$	$\$B.subset(S^{TR}, T^{TR})$
Not a subset	$S \not\subseteq T$	$\$B.notSubset(S^{TR}, T^{TR})$
Finite set	finite(S)	$\$B.finite(S^{TR})$
Partition of a set	partition$(S_1, \ldots, S_n)$ $(n \geq 1)$	$\$B.partition(S_1^{TR}, \ldots, S_n^{TR})$

Number Predicates

Event-B Notation	Event-B Syntax	Translation
Less than	$m < n$	$\$B.lessThan(m^{TR}, n^{TR})$
Less equal	$m \leq n$	$\$B.lessEqual(m^{TR}, n^{TR})$
Greater than	$m > n$	$\$B.greaterThan(m^{TR}, n^{TR})$
Greater equal	$m \geq n$	$\$B.greaterEqual(m^{TR}, n^{TR})$

7.5.2 Expressions

Identifiers

Event-B Notation	Event-B Syntax	Translation
Free identifier	χ	1) $\$cst.\chi$ if χ is a constant 2) $\$var.\chi.value$ if χ is a variable 3) $\$arg.\chi.value$ if χ is a parameter
Bound identifier	χ	1) χ if χ is not a primed identifier 2) $\$var.\chi.\_value$ if χ is a primed identifier

Boolean

Event-B Notation	Event-B Syntax	Translation
Boolean TRUE	TRUE	$\$B.TRUE$
Boolean FALSE	FALSE	$\$B.FALSE$
Bool expression	bool(P)	$\$B.bool(P^{TR})$

Arithmetic

Event-B Notation	Event-B Syntax	Translation
Integer literal	α	$\$B('\alpha')$
Subtraction	$m - n$	$\$B.minus(m^{TR}, n^{TR})$
Quotient	$m \div n$	$\$B.divide(m^{TR}, n^{TR})$
Reminder	$m \bmod n$	$\$B.mod(m^{TR}, n^{TR})$
Exponentiation	$m \mathbin{\widehat{}} n$	$\$B.pow(m^{TR}, n^{TR})$
Addition	$m_1 + \ldots + m_n$ $(n \geq 2)$	$\$B.plus(m_1^{TR}, \ldots, m_n^{TR})$
Multiplication	$m_1 * \ldots * m_n$ $(n \geq 2)$	$\$B.multiply(m_1^{TR}, \ldots, m_n^{TR})$
Predecessor	$\mathrm{pred}(m)$	$\$B.pred(m^{TR})$
Successor	$\mathrm{succ}(m)$	$\$B.succ(m^{TR})$
Unary minus	$-m$	$\$B.unminus(m^{TR})$

Sets

Event-B Notation	Event-B Syntax	Translation
Set extension	$\{E_1, \ldots, E_n\}$ $(n \geq 1)$	$\$B.SetExtension(E_1^{TR}, \ldots, E_n^{TR})$
Set comprehension	$\{x_1, \ldots, x_n \cdot P \mid E\}$ $(n \geq 1)$ $\{E \mid P\}$ (short form)	$\$B.SetComprehension(P^{TR*}, E^{TR*}, [x_1^{TR*}, \ldots, x_n^{TR*}])$
Power set	$\mathbb{P}(S)$	$\$B.PowerSet(S^{TR})$
Non-empty subsets	$\mathbb{P}_1(S)$	$\$B.PowerSet1(S^{TR})$
Cartesian product	$S \times T$	$\$B.CartesianProduct(S^{TR}, T^{TR})$
Empty set	$\varnothing$	$\$B.EmptySet$
Boolean set	BOOL	$\$B.BOOL$
Interval	$m \mathinner{..} n$	$\$B.UpTo(m^{TR}, n^{TR})$
Integer numbers	$\mathbb{Z}$	$\$B.INTEGER$
Natural numbers	$\mathbb{N}$	$\$B.NATURAL$
Positive numbers	$\mathbb{N}_1$	$\$B.NATURAL1$
Set difference	$S \setminus T$	$\$B.setMinus(S^{TR}, T^{TR})$
Set union	$S_1 \cup \ldots \cup S_n$ $(n \geq 2)$	$\$B.setUnion(S_1^{TR}, \ldots, S_n^{TR})$
Set intersection	$S_1 \cap \ldots \cap S_n$ $(n \geq 2)$	$\$B.setInter(S_1^{TR}, \ldots, S_n^{TR})$
Generalized union	$\mathrm{union}(U)$	$\$B.union(U^{TR})$
Generalized intersection	$\mathrm{inter}(U)$	$\$B.inter(U^{TR})$
Quantified union	$\bigcup x_1, \ldots, x_n \cdot P \mid E$ $(n \geq 1)$ $\bigcup E \mid P$ (short form)	$\$B.quantifiedUnion(P^{TR*}, E^{TR*}, [x_1^{TR*}, \ldots, x_n^{TR*}])$
Quantified intersection	$\bigcap x_1, \ldots, x_n \cdot P \mid E$ $(n \geq 1)$ $\bigcap E \mid P$ (short form)	$\$B.quantifiedInter(P^{TR*}, E^{TR*}, [x_1^{TR*}, \ldots, x_n^{TR*}])$
Cardinality	$\mathrm{card}(S)$	$\$B.card(S^{TR})$
Minimum	$\min(S)$	$\$B.min(S^{TR})$
Maximum	$\max(S)$	$\$B.max(S^{TR})$

Relations

Event-B Notation	Event-B Syntax	Translation
Ordered pair	$E \mapsto F$	$\$B.Pair(E^{TR}, F^{TR})$
Relations	$S \leftrightarrow T$	$\$B.Relations(S^{TR}, T^{TR})$
Total relations	$S \nleftrightarrow T$	$\$B.TotalRelations(S^{TR}, T^{TR})$
Surjective relations	$S \leftrightarrow\!\!\!\!\rightarrow T$	$\$B.SurjectiveRelations(S^{TR}, T^{TR})$
Total surjective relations	$S \nleftrightarrow\!\!\!\!\rightarrow T$	$\$B.TotalSurjectiveRelations(S^{TR}, T^{TR})$
Domain	$\mathrm{dom}(r)$	$\$B.dom(r^{TR})$
Range	$\mathrm{ran}(r)$	$\$B.ran(r^{TR})$
Relation image	$r[S]$	$\$B.relationImage(r^{TR}, S^{TR})$
Domain restriction	$S \lhd r$	$\$B.domainRestriction(S^{TR}, r^{TR})$
Domain subtraction	$S \lhd\!\!\!- r$	$\$B.domainSubtraction(S^{TR}, r^{TR})$
Range restriction	$r \rhd T$	$\$B.rangeRestriction(r^{TR}, T^{TR})$
Range subtraction	$r \rhd T$	$\$B.rangeSubtraction(r^{TR}, T^{TR})$
Forward composition	$r_1 ; \ldots ; r_n \ (n \geq 2)$	$\$B.forwardComposition(r_1^{TR}, \ldots, r_n^{TR})$
Backward composition	$r_1 \circ \ldots \circ r_n \ (n \geq 2)$	$\$B.backwardComposition(r_1^{TR}, \ldots, r_n^{TR})$
Overriding	$r_1 \lhd\!\!\!- \ldots \lhd\!\!\!- r_n \ (n \geq 2)$	$\$B.override(r_1^{TR}, \ldots, r_n^{TR})$
Direct product	$r_1 \otimes r_2$	$\$B.directProduct(r_1^{TR}, r_2^{TR})$
Parallel product	$r_1 \parallel r_2$	$\$B.parallelProduct(r_1^{TR}, r_2^{TR})$
Inverse relation	$r^{\sim}$	$\$B.converse(r^{TR})$

Functions

Event-B Notation	Event-B Syntax	Translation
Partial functions	$S \nrightarrow T$	$\$B.PartialFunctions(S^{TR}, T^{TR})$
Total functions	$S \rightarrow T$	$\$B.TotalFunctions(S^{TR}, T^{TR})$
Partial injections	$S \rightarrowtail T$	$\$B.PartialInjections(S^{TR}, T^{TR})$
Total injections	$S \rightarrowtail T$	$\$B.TotalInjections(S^{TR}, T^{TR})$
Partial surjections	$S \twoheadrightarrow T$	$\$B.PartialSurjections(S^{TR}, T^{TR})$
Total surjections	$S \twoheadrightarrow T$	$\$B.TotalSurjections(S^{TR}, T^{TR})$
Total bijections	$S \rightarrowtail\!\!\!\!\twoheadrightarrow T$	$\$B.TotalBijections(S^{TR}, T^{TR})$
Lambda	$\lambda L \cdot P \mid E$	$\$B.Lambda(P^{TR*}, E^{TR*},$
	$\lambda E \mid P$ (short form)	$[x_1^{TR*}, \ldots, x_n^{TR*}])$
Identity	id	$\$B.id$
First projection	prj_1	$\$B.prj1$
Second projection	prj_2	$\$B.prj2$
Function image	$f(E)$	$\$B.functionImage(f^{TR}, E^{TR})$

7.5.3 Assignments

Event-B Notation	Event-B Syntax	Translation
Becomes equal to	$x_1, \ldots, x_n := E_1, \ldots, E_n$	$\$B.becomesEqualTo([x_1^{TR}, \ldots, x_n^{TR}],$
	$(n \geq 1)$	$[E_1^{TR}, \ldots, E_n^{TR}])$
Becomes member of	$x :\in E$	$\$B.becomesMemberOf(x^{TR}, E^{TR})$
Becomes such that	$x_1, \ldots, x_n :\mid Q(x_1', \ldots, x_n')$	$\$B.becomesSuchThat([x_1^{TR}, \ldots, x_n^{TR}],$
	$(n \geq 1)$	$Q^{TR*}, [x_1'^{TR*}, \ldots, x_n'^{TR*}])$

7.6 Interpretation of Translated Formulas

The interpretation of translated Event-B formulas is realized by the JavaScript library described in Appendix C. Informally, we carefully define the detailed implementation rules for each API according to its semantics in the Event-B mathematical language. The hypothesis 2 in Chapter 9 assumes that the interpretation of the translated Event-B formulas in JavaScript is correct. Therefore, we can obtain an observational equivalence between an Event-B formula interpretation and its JavaScript formula interpretation.

7.7 Simulation Control

7.7.1 Simulation Scheduler

The non-deterministic operational model of Event-B requires that execution cycles are composed of three steps: the evaluation of the guards of all events, the choice among the set of enabled events, and the execution of the substitutions of the chosen event. The generic cycle runs as:

1. setup contexts, prompt user if some constants have no values;
2. execute the *INITIALISATION* event;
3. do forever
   (a) save the state of variables;
   (b) update the variable view;
   (c) compute the invariants;
   (d) if one invariant is false then stop;
   (e) while no event is enabled do
      – build the set of enabled events by computing the guards of all events with arguments from the Web browser;
   (f) randomly pick an event to trigger from the set of enabled events;
   (g) save the state of arguments;
   (h) execute the actions part of the selected event.

Note that the step 3.(d) is useless when executing a fully proven model. However, our experience has instructed us that it is a very useful feature when preparing and "fixing" complex refinements. The step 3.(f) uses a random choice strategy to pick an enabled event while users can implement their own strategy.

This cycle is implemented in a simulation scheduler which is executed in an auto mode or in a manual mode. The most important functions of the scheduler are:

Function	Description
jeb.scheduler.init	starts a simulation cycle
jeb.scheduler.onEventClick	manually executes an enabled event
jeb.scheduler.autoRun	runs a simulation in auto mode
jeb.scheduler.stop	stops a simulation in auto mode
jeb.scheduler.testAllGuards	computes the guards of all events
jeb.scheduler.pickEvent	picks an enabled event; a random choice function is provided by default, users can override it to implement their own strategy
jeb.scheduler.execute	executes an enabled event
jeb.scheduler.checkInvariants	checks all invariants

7.7.2 Parameters of a Simulation

The user configuration files can also be used to finely tune simulations. It is possible to run several different kinds of simulations from the same generated core. The global parameters can be easily set or reset in the user configuration files, some of them can be directly set in the Web browser. For example:

```
jeb.ui.TIMER_INTERVAL = 100; // set timer interval to 100 milliseconds
jeb.ui.MAX_TRY_ARGUMENTS = 10; // set the maximum number of try arguments
                               // for probabilistic generation of parameters
jeb.ui.CODE_TIP_DISPLAY = true; // show code tips
jeb.ui.PARAMETERS_DISPLAY = true;// show event parameters in the event view
jeb.ui.GURADS_DISPLAY = true; // show event guards in the event view
jeb.ui.ACTIONS_DISPLAY = true; // show event actions in the event view
jeb.ui.SCENARIO_ENABLED = true; // enable the scenario functionality
```

7.7.3 Scenario Controller

The simulator keeps a history of the execution sequence of events. The system state is saved at every simulation cycle in a scenario object. A state is composed by the values of all variables and all event parameters. Users can click on an event label in the scenario view to restore a historical state. It is then possible to explore alternative behaviors. The scenario functionality can be enabled or disabled in the user interface. The implementation of the scenario controller is realized by two functions:

Function	Description
jeb.scenario.save	saves a state to the scenario object
jeb.scenario.restore	restores a state from the scenario object

7.7.4 Animator

Users can observe the simulations through the variable view in the user interface. However, an extra graphic display of system state will facilitate the observations and analysis. The JeB simulator includes an HTML5 `canvas` to implement a graphic display of the system's states. Two functions implement the display:

Function	Description
jeb.animator.init	initializes the display area
jeb.animator.draw	draws an image of the system state

The first function is called once at the start of a simulation, the second is called at each simulation cycle. Optionally, if users realize these two functions and run the simulation in the auto mode, then a graphical animation is shown on the top area in the user interface.

7.8 Event-B Project Diagram

When traversing an Event-B project, the JeB translator generates an index page in HTML as shown on Figure 7.3. This page provides users with a diagrammatic view of the specification [3]. This facility helps users to understand more easily complex models and to navigate the generated simulators. We found it quite useful when experimenting with the MIDAS model [Wright 2009b, Wright 2010] which goes through 39 refinements of the abstract machine and 21 refinements (extensions) of the contexts.

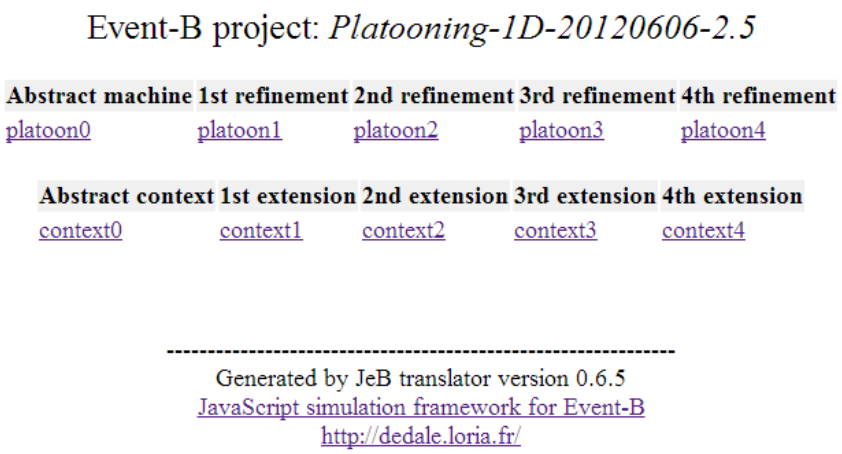

Figure 7.3: An Event-B project diagram

7.9 Summary

Combining the ideas behind existing animators and translators, the JeB translator outputs a JavaScript model and generates an HTML user interface for each Event-B context and machine. The generated models and the web-based user interface provide users with an integrated execution environment for the simulations. However, the JeB translator produces only one part of a complete simulation. The generated simulators require a runtime library to execute them. This library is implemented in JavaScript and composed of two main parts: one part supports all Event-B mathematical notations, the other part

3. This idea is the same as the one exposed in http://wiki.event-b.org/index.php/Project_Diagram

provides auxiliary utilities, such as the scheduler, the scenario controller and the animator. With this library, all type formulas in Event-B can be translated in a systematic way. However, users still retain the ability to provide their own implementations. Furthermore, the generated Event-B project diagram is useful to better understand complex models and to navigate the simulators.

Chapter 8

JeB Utilization and Analysis of Simulations

Contents

8.1 Introduction

Using JeB to simulate Event-B models is a collaborative process between automated tools and humans. After the simulators are automatically generated from a given Event-B model by the JeB translator, we still need do three things to run them:

1. provide the values of all constants,
2. provide parameter functions for auto-run mode,
3. set up the graphic display of states (optional).

In this chapter, we describe the usage of JeB to create simulations for the 1D and 2D platooning models. To use JeB, the users must be familiar with the JavaScript and the APIs (the JavaScript library for Event-B) defined in Appendix C. The code shown in this chapter is more technical. These simulation examples demonstrated the feasibility of our approach to overcome the difficulties of animating the Event-B specifications. Then we give some observations for models used in our development and some analysis from the validation point of view.

8.2 Simulation of the 1D Platooning

The 1D Platooning (see Appendix D) case study is used as a reference. It can be animated with the existing animators by some strategies.

When launched from the Rodin interface, the JeB translator considers all the Event-B components in the project and generates one simulator per machine, either abstract or refined. The generated simulators is located in a "`jeb`" directory under each Event-B project workspace. A simple naming scheme allows us to manage independently the different simulators. Let us detail the effort needed to simulate the abstract machine `platoon0` (see Appendix D.6).

8.2.1 Minimal Simulation

When the simulator page `platoon0.html` for the abstract machine `platoon0` is opened in a Web browser, an alert reminds us that some constants have no values. Abstract constants in contexts need to be instantiated for the model to be executed. This can be done in one of: three ways:

1. set the value in the file `jeb_user.js` after running the JeB translator,
2. set the value in the file `<context>.js`,
3. set the value in the file `<machine>_user.js`.

In the first case, the values of constants are permanently kept. In the second and third cases, the values need to be set again after each run of the translator since new instances of the files `<context>.js` and `<machine>_user.js` are then created. The values are available to all machines in the first and second cases, but they are restricted to one particular machine in the third case.

Using the first technique, we add the following lines in the file `jeb_user.js` (the prefix `$cst` denotes constants):

```
$cst.VEHICLES = $B('4');
$cst.CRITICAL_DISTANCE = $B('20');
$cst.initial_xpos = $B.TotalFunctions(
    $B.UpTo($B('1'), $cst.VEHICLES),
    $B.NATURAL).makeSetExtension( function(v) {
        return $B.multiply($B.minus($cst.VEHICLES ,v),
            $B.plus($cst.CRITICAL_DISTANCE, $B('1')));
    });
```

The constant function `$cst.initial_xpos` utilizes some APIs from the JavaScript library for Event-B in Appendix C.

After refreshing the page `platoon0.html`, the button `INITIALISATION` turns green and becomes enabled. Firing the `INITIALISATION` event leads to the next alert about the function `get_magic_xpos` not being defined for the event `all_moves`. Among the parameters of an Event-B event, some can be considered as true parameters, while the others are akin to local variables, in the usual programming sense. The JeB translator associates each true parameter par_i with a function get_par_i. By default, those functions are called at the beginning of each simulation cycle. If the range of a parameter can be detected, the JeB translator will generated a random choice function from the range, otherwise, only a function stub will be generated in the file `<machine>_user.js`. In our example, JeB generates the following lines in the file `platoon0.user.js`:

```
// Auto-generated function: argument generator
/*
var get_magic_xpos = function( eventId ) {
    if (eventId == $evt.e1) {
        // @TODO
    }
};
*/
```

We implemented this function using the following code in `jeb_user.js` file, so it can be reused in successive refinements.

```
var get_magic_xpos = function() {
    var __magic_xpos = $B.TotalFunctions(
        $B.UpTo($B('1'), $cst.VEHICLES),
        $B.NATURAL).makeSetExtension( function(v) {
            return $B.plus($var.xpos0.value.getImage(v),
                $B.UpTo($B('-5'), $B('10')).anyMember() );
        });
    return __magic_xpos;
};
```

After implementing this function, we refresh the page `platoon0.html` and then click the `Auto Run` button. The simulation will execute until the `Stop` button is clicked or the system reaches a state where no events are enabled. Instead of using the auto-run feature, we could activate the events manually, one at a time. Parameter values are given through the fields in the event view. Users can provide literal values or call other argument generator functions instead of the default ones.

8.2.2 Graphic Display

The JeB simulator includes an HTML5 `canvas` to implement a graphic display of the system's states. Optionally, we need to implement two functions to construct a graphic display: `jeb.animator.init` which initializes the display, and `jeb.animator.draw` which draws an image of the system state and is called at each simulation cycle.

For the machine `platoon0`, the following code is written in the `jeb_users.js` file. The first function defines a 100x500 pixels `canvas`. The second function uses circles to schematize the vehicles' positions.

```
jeb.animator.init = function() {
    $anim.canvas.width = 500;
    $anim.canvas.height = 100;
    $anim.canvas.style.display = '';
};

jeb.animator.draw = function() {
    var i, x,
        pos = $var.xpos || $var.xpos0,
        data = pos.value,
        coord_x = -Math.floor(data[0]/$anim.canvas.width)
                  * $anim.canvas.width;
    $anim.clearRect(0, 0, $anim.canvas.width, $anim.canvas.height);
    $anim.beginPath();
    for (i=0; i<data.length; i++) {
        x = data[i].right % $anim.canvas.width;
        $anim.moveTo(x, 50);
        $anim.arc(x, 50, 10, 0, Math.PI*2, true);
        $anim.strokeText(i+1, x-5, 53);
    }
    $anim.closePath();
    $anim.stroke();
};
```

8.2.3 Simulation of the Refinements

All refinements in the 1D model can be executed with JeB. The work required on each refinement is similar to what is presented above. Table 8.1 lists all the names in the specification on which an action is required to set up a running simulation. This gives a rough idea of the effort needed to use JeB. It should be noted that the functions for the graphic display are defined only once for all simulations: it can be used on all refinements.

8.3 Simulation of the 2D Platooning

The 2D Platooning (see Appendix E) case study extends the previous one by adding the lateral control. The 2D specification is created by a sequence of requirements which

Machine	Activity
platoon0	constants: `VEHICLES`, `CRITICAL_DISTANCE`, `initial_xpos`
	parameters: `get_magic_xpos`
platoon1	parameters: `get_magic_xpos_vehicle`
platoon2	constants: `MAX_SPEED`, `MIN_ACCEL`, `MAX_ACCEL`, `initial_speed`, `new_speed`, `new_xpos`, `new_xpos_max`, `new_xpos_min`
	parameters: `get_magic_accel`
platoon3	constants: `initial_accel`
platoon4	constants: `IDEAL_SPEED`, `ideal_distance`, `new_accel`

Table 8.1: Activities for the 1D platooning simulations

follows exactly the same structure as for the 1D specification. The most important features are the presence of an abstract carrier set and some uninterpreted functions. Brama, AnimB and ProB fail to execute the models because they cannot setup the contexts in the first step of animations. This case study acts our main test-bed.

8.3.1 Carrier Sets

In the 2D specification, the vehicles move in a space modeled by an abstract carrier set: `Point`. This is an abstraction of the kinematic space which has six dimensions: $(x, y, \gamma^{\theta}, \sigma^{\theta}, v, \kappa)$ representing the geometric position, the orientation, the velocity and the trajectory's curvature. `Context0` introduces the carrier set `Point` whose constructor and access functions are introduced in `context2` shown in Figure 8.1.

CONTEXT *context0*
SETS *Point*
...
CONTEXT *context2*
CONSTANTS
$new\_point,\ x,\ y,\ \gamma\theta, \sigma\theta, v, \kappa$
AXIOMS
axm_new_point: $new\_point \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times 0..MAX\_SPEED \times -MAX\_\kappa..MAX\_\kappa \rightarrow Point$
axm_x: $x \in Point \rightarrow \mathbb{Z}$
axm_y: $y \in Point \rightarrow \mathbb{Z}$
axm_γθ: $\gamma\theta \in Point \rightarrow \mathbb{Z}$
axm_σθ: $\sigma\theta \in Point \rightarrow \mathbb{Z}$
axm_v2: $v \in Point \rightarrow 0..MAX\_SPEED$
axm_κ2: $\kappa \in Point \rightarrow -MAX\_\kappa..MAX\_\kappa$
...

Figure 8.1: The definition of Point in Event-B

Brama, AnimB and ProB fail with such a carrier set because we cannot provide a concrete and meaningful representation which can be efficiently enumerated. Yet, specifiers often know about sensible instantiations. Taking an object-oriented point of view, `Point` can

be implemented as a JavaScript object as follows:

```
$cst.Point = function(x, y, γθ, σθ, v, κ) {
    var obj = {};
    obj.x = x;
    obj.y = y;
    obj.γθ = γθ;
    obj.σθ = σθ;
    obj.v = v;
    obj.κ = κ;
    obj.__proto__ = $cst.Point.prototype;
    return obj;
};

$cst.new_point = $B.SetExtension();
$cst.new_point.functionImage = function(x, y, γθ, σθ, v, κ) {
    return new $cst.Point(x, y, γθ, σθ, v, κ);
};
$cst.new_point.concrete = false;

$cst.x = $B.SetExtension();
$cst.x.functionImage = function( point ) {
    if ( point instanceof $cst.Point ) {
        return point.x;
    }
};
$cst.x.dom = $cst.Point;
$cst.x.T = $B.INTEGER;
$cst.x.concrete = false;

...
```

The constant functions, such as *new_point*, x, y, $\gamma\theta$, $\sigma\theta$, v, κ are implemented in an abstract way with a mapping function `functionImage` and setting their property `concrete` to `false`. We will compute a function image on demand by the function `functionImage`, so we don't need to enumerate all the possible pairs as is required for animators Brama and AnimB.

8.3.2 Functions Defined by Properties

The 2D specification uses several functions which are only defined by some properties:
– `dist`: the longitudinal distance between two points,
– `y_dist`: the lateral distance between two points,
– `nearest`: the nearest point belonging to a trajectory to a given point.
These functions are in interface to link external implementation. Here we only give their main constraint properties, how to compute them depends the user decision. Usually, they are complicated; we cannot simply abstract them in the Event-B model. For example, the vehicles' geometry must be taken into account for the computation of the distances;

different definitions of the `nearest` will produce subtly different control algorithms. Again, Brama, AnimB and ProB cannot be used to link external implementation.

Implementing the `dist` function in JavaScript by using the canonical euclidean definitions is straightforward. The `y_dist` and `nearest` functions are slightly trickier: their implementations depend on the implementation of the trajectories.

8.3.3 Generation of Arguments and Definition of Constants

Simulating the 2D platooning model requires the definition of arguments generator functions similar to the ones used in 1D. Since the control algorithm computes two quantities (acceleration and wheel angle), we need one more generator function in the machine `platoon2`. The 2D model contains more constants and so induces a little more work to set up the simulations.

8.4 Observations on JeB Usage

8.4.1 Simulation Cost

It is interesting to note that the cost of using JeB is reasonable (about 2% of the simulation code must be provided by humans on four case studies) and can be used as a tool for the development process. Table 8.2 is the result of our observations on the cost of using JeB. It give the size of the specifications (number of components and number of lines of Event-B), the size of the complete simulation code (number of lines of JavaScript and HTML), the size of the manual code that the user must provide (number of lines in JavaScript) and the ratio that describes the percent of manual code compared to the complete simulation code.

Event-B Model	Number of Components	Event-B Texts	Simulation Code	Manual Code	Ratio (%)
1D Platooning	10	600	13 000	160	1.2
2D Platooning	10	1 800	28 300	460	1.6
Transport-domain	23	1 100	22 400	470	2.1
MIDAS	104	21 800	542 300	1 300	0.2

Table 8.2: Simulation cost

8.4.2 1D Platooning Model

During its development, the 1D specification went through several stages containing erroneous behaviors. We must insist on the fact that each stage is formally correct: all POs are discharged.

In an early version, vehicles could move backward which is an unexpected behavior. This behavior was introduced in the abstract machine `platoon0` (see Appendix D.6).

Strengthening the guard of the event `all_moves` with

$$\forall v \cdot v \in 1..VEHICLES \Rightarrow magic\_xpos(v) \geq xpos0(v)$$

corrects the problem. The anomalous behavior is highly visible on the graphic display in auto-run mode.

Another problem, much more bothering, concerns the deadlocks present in the published version [Lanoix 2008]. In Event-B, the deadlock in some systems is the absence of any enabled event. The execution then comes to a halt. Since all refinements of the specification are executable with JeB, we can pin down the observation of the deadlock in the second refinement. Playing with the history and manual setting of arguments, we can analyze the precise conditions which provoke deadlocks.

A third problem in the last refinement, still present, concerns an emergent behavior: oscillations. Oscillations occur when the platoon reaches a constant speed while, inside, the vehicles engage in a cycle of accelerations and braking around the platoon's average speed. This is an emerging behavior: nothing in the specification addresses it explicitly. In practice, such a behavior should be avoided for several reasons, among which energy consumption or passengers comfort for instance. Oscillations are visible and their apparition can be finely analyzed.

8.4.3 2D Platooning Model

The 2D specification (see Appendix E) can be executed with JeB. We concentrated our observations on a few questions.

First, we looked for deadlocks as for the 1D case study. We reused the knowledge gained from the simple specification to check thoroughly the second refinement. The deadlocks can be observed since the second refinement.

The second question is the apparition of oscillations, both lateral and longitudinal.

The third question is specific to the 2D model: how close to the reference trajectory do the vehicles actually stay? Interestingly, setting up the simulation and making the observations raised important issues about the practical definition of the notions of "closest point of" and "distance from" a trajectory.

The last question is the representation of `trajectory` which is formalized as:

inv_temp:	$temp \in \mathbb{N}$
inv_traj :	$trajectory \in 0..temp \rightarrow (1..VEHICLES \rightarrow Point)$

In fact, `trajectory` may represent an very large list. This raises no problem in the Event-B language, but cannot be implemented naively as memory would be exhausted during a long simulation. An acceptable representation is to use a fixed size array: we just need the most recent points to compute the lateral control in reality. The 2D platooning specification needs be improved.

8.4.4 Transport-domain Model

In Event-B, axioms are used as hypotheses when discharging POs, theorems and, more generally, predicates. Only well-definedness POs are generated for axioms. A consequence is the absence of inherent checks for the consistency of the set of axioms. There is an urgent need for assessing the consistency of contexts as inconsistent axioms make all formula provable, and the (formal) correctness of the models is defined as the proof of all POs.

Axioms can also be "wrong" in a less dramatic way: they can just specify another set of values than what was intended. The JeB constant checker can help to find mistakes in axioms. For the transport-domain model, we have found such an error in the context `Net1`. The axiom `axm10` is defined as:

axm10: $\forall s1, s2 \cdot s1 \in stations \land s2 \in stations \Rightarrow$
$(netHubs[\{s1\}] \cup netHubs[\{s2\}] \neq \varnothing \Rightarrow areConnected(s1 \mapsto s2) = TRUE)$

The mistake is caused by missing a precondition $s1 \neq s2$, because the relations based on *stations* in this model are not reflexive. The correct version is:

axm10: $\forall s1, s2 \cdot s1 \in stations \land s2 \in stations \land s1 \neq s2 \Rightarrow$
$(netHubs[\{s1\}] \cup netHubs[\{s2\}] \neq \varnothing \Rightarrow areConnected(s1 \mapsto s2) = TRUE)$

8.4.5 MIDAS Model

Using the JeB constant checker, a mistake is found in the context `InstructionConFlagMch1`. There are four axioms in this context:

axm1: $Null3Inst \subseteq Null2Inst$
axm2: $ConFlagWriteInst \subseteq Null2Inst$
axm5: $Null2Inst = Null3Inst \cup ConFlagWriteInst$
axm6: $Null2Inst \cap ConFlagWriteInst = \varnothing$

Evidently, they are easily corrected by a condition:

$Null2Inst = \varnothing \land Null3Inst = \varnothing \land ConFlagWriteInst = \varnothing$

But, they are meaningless. In fact, the sets of *Null3Inst* and *ConFlagWriteInst* are not empty as is specified in [Wright 2009b]. So, we think that the axiom `axm6` is a spelling error. The correct version is:

axm6: $Null3Inst \cap ConFlagWriteInst = \varnothing$

In fact, these four axioms can be expressed by one axiom with the *partition* notation:

axm1: partition$(Null2Inst, Null3Inst, ConFlagWriteInst)$

8.4.6 Comparison between Existing Animators

8.4.6.1 Technical Comparison

Table 8.3 is a concise technical comparison between the existing animators which are integrated to the Rodin platform.

	Brama	**AnimB**	**ProB**	**JeB**
Interpretation language	Java	Java	Prolog	JavaScript
Event-B notation supporting	partial	partial	total	total
Rodin version supporting	under 1.0	2.x	2.x	2.x
Axioms/Invariant checking	yes	no	yes	yes
Treatment of carrier sets	set of integers	set of symbols	set of symbols	set of integers or symbols, JS objects
Treatment of functions	set of pairs	set of pairs	set of pairs, lambda	set of pairs, JS functions
Constant input interface	yes	yes	no	yes
Visualization	Flash	Flash	BMotion Studio	HTML/JavaScript
Multi-level animation	yes	yes	yes	no

Table 8.3: Technical comparison between the existing animators

Interpretation language Brama and AnimB use the same Java library to interpret Event-B formulas. ProB is based on Prolog to interpret Event-B formulas. JeB uses a JavaScript interpretation. All these languages can do an adequate job.

The Event-B notation supporting Unlike ProB and JeB, the implementation of Brama and AnimB partially supports the Event-B notations. Therefore, some models are not directly animatable with Brama and AnimB, they must undergo some transformations to be adapted to the tools.

Rodin version supporting AnimB, ProB and JeB support the latest Rodin version 2.x at this moment. In contrast, Brama only supports the Rodin version under 1.0. The models used in different Rodin versions are not compatible, so, to use Brama, we must downgrade the model version.

Axioms/Invariant checking To validate an Event-B model, we must guarantee that the observed behaviors would not be prevented in the original model. AnimB does not check the axioms and invariants, so it cannot be trusted for validation activities.

Treatment of carrier sets Unlike enumerated sets, carrier sets are a difficult point to execute Event-B models, they are very abstract or infinite. Animators can use tricks to instantiate them if they are finite. To do that, Brama represents them by set of integers; AnimB represents them by set of symbols, these symbols are not needed to be defined as constants; ProB automatically either represents them by sets of symbols whose size is decided in its preferences, or explicitly defines them as enumerated sets whose elements must be defined as constants in an extended context. JeB is more flexible, it can use JavaScript objects to represent infinite carrier sets.

Treatment of functions To give values to functions, Brama and AnimB define them as enumerated sets of pairs. This representation is not convenient to users and prevents the use of functions with an infinite domain. ProB can compute a function image by a lambda abstraction; JeB does the same by a JavaScript function.

Input interface for constants Both Brama, AnimB and JeB provide an interface to setup an animation with specific constants and set values. ProB has no such facility, it will automatically decide their values based on the constraint-solving technique. To setup a specific animation with ProB, we must use the context extension mechanism.

Visualization A good visualization is important for the successful deployment of formal methods. Brama and AnimB can create sophisticated visualizations using Flash. The BMotion Studio based on ProB uses a graphical editor with a number of default controls and observers to create visualizations, it is sufficient for most cases. JeB provides a lightweight graphical environment based on HTML/JavaScript which allows us to create any sophisticated visualization.

Multi-level animation Sometimes it is difficult to analyze a refinement relationship for complex models. Brama, AnimB and ProB provide a multi-level animation facility to help detecting the refinement errors. To simplify the design and the implementation, JeB does not provide this facility; the consistency of refinements should be guaranteed by the associated proof obligations.

8.4.6.2 Usage Comparison

Table 8.4 summarizes the usage of the four tools on our case studies.

Animating the 1D Platooning model To animate the 1D Platooning model, Brama and AnimB need some transformations [Mashkoor 2009b] to the original model, this is time consuming. ProB can automatically animate it with a limit on observable behaviors, this is useful to find deadlocks or a counter-example. To enlarge the observation scope with ProB, the context extension mechanism can be used. Using JeB, the effort to build

	Brama	**AnimB**	**ProB**	**JeB**
1D Platooning animation	yes	yes	yes	yes
2D Platooning animation	no	no	no	yes
Transport-domain animation	yes	yes	yes	yes
MIDAS animation	partial	partial	partial	yes

Table 8.4: The usage of four animators on our case studies

the simulations is small; a large parts of user's code can be reused through the different refinements.

Animating the 2D Platooning model Brama, AnimB and ProB cannot execute this model because they cannot setup the contexts in the first step of animations: an infinite carrier set and some uninterpreted functions are their main obstacles. The simulations realized with JeB are our only way to analyze its behaviors.

Animating the Transport-domain model This model uses many carrier sets. To animate it, Brama needs an instantiation of them by enumerated sets of integers; AnimB needs an instantiation of them by enumerated sets of symbols; JeB needs an instantiation of them by enumerated sets of integers or symbols. ProB cannot directly animate it; we must use the context extension to define these carrier sets by enumerated sets of constants, this will introduce plenty of constants to the extended context.

Animating of MIDAS model This model is very large and complex. Brama and AnimB can partially animate it after some transformations. ProB can directly animate some refinements with a careful setting of its preferences (e.g., the size of integers, the size of unspecified differed sets). The main difficulties to these animators are the large state space and some uninterpreted functions which cannot be given an efficient definition with the Event-B language. JeB can animate all refinements of this model, but its performance needs to be improved.

8.5 Analysis from a Validation Point of View

8.5.1 Validation of Axioms

Event-B models use axioms in the contexts to specify the static properties of systems. These axioms are just hypotheses, i.e., we assume they are correct, so we can use them to discharge the generated proof obligations. The Well-definedness POs are automatically generated for axioms, which only guaranteed their types. For a complex context, some mistakes can be found by the JeB constant checker, such as in the transport-domain model and in the MIDAS model. A wrong axiom can discharge any goal in a sequent if we reference it as a hypothesis, which will cause a serious problem in the theorem

proving system. So it is necessary to verify the correctness of axioms by semi-formal reasoning, i.e., we evaluate them by providing constant values.

8.5.2 Validation of Properties

The 1D specification looks simple. The last refinement consists of 15 events and around 140 individual logical formulas which form the invariants and the guards. Many of those formulas are very simple as they do not express more than types. Yet, getting the specification right was a difficult task. In this section, we discuss how JeB would have helped in this task.

The backward movement problem was found by a picky human reader. In fact, it could have stayed unnoticed for a long time as the forward-only move is an hypothesis of the whole system rather than a necessary condition. Platooning systems with backward moves may be designed but at the probable expense of higher complexity. Since both hypotheses can lead to correct systems, the verification procedure (the proofs) cannot detect the problem in the model.

The deadlock problem of the 1D model was first identified and understood by using animations realized with Brama and ProB. But both tools failed on the 2D model because of the difficulties of setting up the contexts illustrated in Section 8.3.1 and 8.3.2. By contrast, it is easy to set up and implement contexts to study the apparition and reasons for the deadlocks which appear for both 1D and 2D models.

The issue of deadlock-freeness in Event-B specifications is a complex one. The standard POs do not protect from deadlocks. It is possible to automatically build a theorem and POs which ensure the absence of deadlock (see Chapter 4). However, the size of the formula to prove grows very fast with the number of events and guards. While it is possible to prove this theorem with time consuming effort, one interesting possibility lies in using simulations to identify the deadlocks and their prevention, before attempting the proofs.

In the case of the 1D model, the improvement offered by JeB is mostly about cost and practicality. To use Brama and AnimB, we must apply some transformation [Mashkoor 2009b] to the original model, it is time consuming. ProB can automatically animate the 1D model and detect deadlocks, but the observable behaviors are very limit; to enlarge the observations, we must use context extension to explicitly set realistic data to constants. Using JeB, the effort to build simulations of the different machines is small. Furthermore, it is spread over all the refinements since large parts of user's code can be reused. Because JeB is based on the Web standards, we could build a simple informative display with a few lines of code.

As the 2D model is failed to be animated, the simulations realized with JeB are our only way to analyze the system behaviors. The issue raised in the observations about the distance functions is actually a crucial one for the direction of a concrete refinement. In practice, trajectories are not continuous lines, but sequences of points perceived through imperfect sensors. We can expect that the tracking behavior of the vehicles is dependent

of the actual computations of the distances. JeB allows us to experiment with several practical algorithms before we commit them to a formal refinement.

8.6 Summary

Using JeB to validate models has a cost. The effort is measured as a number of actions and the percentage of code of the simulations. Our graphics are simple and, of course, more sophisticated graphics would require more code. It is worth noting that the amount of user-provided code is small: around 2%. It is also interesting to note that, although the simulations of the refinements are independent, user-provided codes can often be shared. This is particularly true of the argument generator functions.

Chapter 9

Correctness of Simulations

Contents

9.1 Introduction

The definition of the correctness of simulations is based on a simple property: only behaviors allowed in the original Event-B models can be observed during the simulations.

Table 9.1 summarizes the symbols and notations that we used. M is an Event-B machine and P is a translated JavaScript simulator of M. The prime ($'$) notation denotes the state after the substitution. The value of a predicate Ψ after the application of a substitution σ is noted as $[\sigma]\Psi$. The superscript (^{t}) notation denotes the translated expressions. The dot (.) notation is used to access JavaScript object properties. The semicolon (;) notation represents the sequential execution of JavaScript statements.

In the following, we assume two hypotheses.

Hypothesis 1 *There exists a function f mapping all JavaScript simulation values V^t to Event-B values V, where $f \in V^t \rightarrow V$*

Elements	**Event-B**	**JavaScript**
machine	M	P
sets	s	s^t
constants	c	c^t
axioms	$Axm(s,c)$	$f_{Axm}()$
variables	v	v^t
invariants	$Inv(s,c,v)$	$f_{Inv}()$
events	$Evts = \{E^1, E^2, ..., E^m\}$	$f_{Evts} = \{f_{E^1}, f_{E^2}, ...f_{E^m}\}$
initialization event	E^0, $E^0 \notin Evts$	f_{E^0}, $f_{E^0} \notin f_{Evts}$
actions	$Act_{E^0}(s,c,v)$	$f_{E^0}.f_{Act}()$
before-after predicate	$BAP_{E^0}(s,c,v')$	
a particular event	E^i, $E^i \in Evts$	f_{E^i}, $f_{E^i} \in f_{Evts}$
parameters	x_{E^i}	$x^t_{E^i}$
guards	$Grd_{E^i}(x_{E^i},s,c,v)$	$f_{E^i}.f_{Grd}(x^t_{E^i})$
actions	$Act_{E^i}(x_{E^i},s,c,v)$	$f_{E^i}.f_{Act}(x^t_{E^i})$
before-after predicate	$BAP_{E^i}(x_{E^i},s,c,v,v')$	
interpretation function	$bool$	$eval$
boolean values	$TRUE$ $FALSE$	$true$ $false$
equal operator	$=$	$==$
substitution	$x := y$	$x^t = y^t$
event occurrences	$e_j, e_j \in \{E^0\} \cup Evts$	$f_{e_j}, f_{e_j} \in \{f_{E^0}\} \cup f_{Evts}$

Table 9.1: Symbols and notations

Hypothesis 1 is the statement that the domain of values in Event-B can be interpreted on the domain of values in JavaScript.

Hypothesis 2 *Let x be the collection of all free identifiers for a predicate Ψ; x^t and $f_\Psi()$ are the translated identifiers and a predicate function respectively and x_0 be the given values of x^t. There exists an interpretation function eval in JavaScript, where*

$$eval(x^t = x_0); eval(f_\Psi()) == true; \quad \Leftrightarrow \quad bool([x := f(x_0)]\Psi) = TRUE$$
$$eval(x^t = x_0); eval(f_\Psi()) == false; \quad \Leftrightarrow \quad bool([x := f(x_0)]\Psi) = FALSE$$

In the left hand side of the equivalence, it is JavaScript view; and in the right hand, it is Event-B view. Hypothesis 2 states that Event-B predicates can be interpreted as JavaScript predicates. It assumes that Event-B and JavaScript are based on the same logic.

9.2 Consistent Behavior

9.2.1 Semantics of an Event-B Machine

The standard semantics of Event-B is defined as a set of proof obligations (POs). The POs expressing the correctness of an individual machine are of three kinds: preservation of invariants, feasibility, and, for some models, absence of deadlock.

Let M be an Event-B machine with sets s, constants c and variables v; the semantics of M must satisfy invariant and feasibility POs.

The invariant POs for M are defined as

$$Axm(s,c) \wedge BAP_{E^0}(s,c,v') \Rightarrow Inv(s,c,v')$$

$$\bigwedge_{i=1}^{m} Axm(s,c) \wedge Inv(s,c,v) \wedge Grd_{E^i}(x_{E^i},s,c,v) \wedge BAP_{E^i}(x_{E^i},s,c,v,v') \Rightarrow Inv(s,c,v')$$

Using the generalized substitution notation, the above formulas are equivalent:

$$Axm(s,c) \Rightarrow [Act_{E^0}(s,c,v)]Inv(s,c,v)$$

$$\bigwedge_{i=1}^{m} Axm(s,c) \wedge Inv(s,c,v) \wedge Grd_{E^i}(x_{E^i},s,c,v) \Rightarrow [Act_{E^i}(x_{E^i},s,c,v)]Inv(s,c,v)$$

The feasibility POs for M are defined as

$$Axm(s,c) \Rightarrow \exists v' \cdot BAP_{E^0}(s,c,v')$$

$$\bigwedge_{i=1}^{m} Axm(s,c) \wedge Inv(s,c,v) \wedge Grd_{E^i}(x_{E^i},s,c,v) \Rightarrow \exists v' \cdot BAP_{E^i}(x_{E^i},s,c,v,v')$$

For some system models, the Deadlock-freeness PO can be expressed as

$$Axm(s,c) \wedge Inv(s,c,v) \Rightarrow \bigvee_{i=1}^{m} \exists x_{E^i} \cdot Grd_{E^i}(x_{E^i},s,c,v)$$

9.2.2 Operational Interpretation of an Event-B Machine

An Event-B machine constitutes a kind of state transition system. We can give a simple operational interpretation to a machine. We assume two hypotheses: the execution of an event is considered to be instantaneous and two events cannot occur simultaneously. The execution following then the steps:

1. execute the initialization event,
2. evaluate all events' guards,
3. if no event has its guards true, then the model execution stops,
4. if the guards of some events are true, then one of the corresponding events necessarily occurs and the state is modified accordingly; continue with step 2.

This behavior clearly shows a form of non-determinism as several guards might be true simultaneously. Event-B makes no assumption concerning the specific event which is executed among those whose guards are true. There are other forms of non-determinism: the assignment of values to the events' parameters, and the "such-that" substitutions in actions.

Definition 1 (Traces of an Event-B Machine) *A finite sequence of event occurrences*

$$e_0 e_1 e_2 ... e_n$$

is a trace of an Event-B machine M if and only if
e_0 *is the initialization of M and*

$$\{e_1, e_2, ..., e_n\} \subseteq Evts$$

and

$$\bigwedge_{j=0}^{n} \mathsf{fis}(e_j)$$

where fis *is the feasibility predicate of the sequence which is the point-wise extension of the feasibility PO defined in [Abrial 1996].*

The set of all finite traces of a machine M is called $Traces(M)$. The following property characterizes traces by the existence of intermediary states v_j in which the guard of e_j holds and where the pair (v_{j-1}, v_j) is in the before-after predicate of event e_j:

Property 1 (Trace characterization) Let v be the variables in a machine M, then:

$$e_0.e_1.e_2...e_n \in Traces(M)$$
$$\Leftrightarrow$$
$$\exists v_0, ..., v_n \cdot [v' := v_0]BAP_{e_0}(s,c,v') \wedge [v := v_0]Inv(s,c,v) \wedge$$
$$\bigwedge_{j=1}^{n} ([v := v_{j-1}]Grd_{e_j}(x_{e_j}, s, c, v) \wedge [v, v' := v_{j-1}, v_j]BAP_{e_j}(x_{e_j}, s, c, v, v') \wedge [v := v_j]Inv(s,c,v))$$

9.2.3 Execution of Simulators

Let P be a translated JavaScript simulator of an Event-B machine M with translated sets s^t, translated constants c^t and translated variables v^t. Let s_v, c_v be the values of s^t, c^t for the instantiated P. For P to be correct, the following properties, derived from the semantics of M, must hold. The numbers refer to the simulation cycle description presented in Section 7.7.1.

1 Setup context:
Let s_v and c_v be the actual values given to carrier sets and constants,
Condition 1 (Correct Context Setup)

$$eval(s^t = s_v);$$
$$eval(c^t = c_v);$$
$$eval(f_{Axm}()) == true;$$
$$\Rightarrow$$
$$bool([s,c := f(s_v), f(c_v)]Axm(s,c)) = TRUE$$

2 Execute the initialization event:
Let v_0 be the value of variables v^t after initialization,
Condition 2 (Correct Initialization)

$$
\begin{aligned}
&eval(v_0 = f_{E^0}.f_{Act}());\\
&eval(f_{Inv}()) == true\\
&\Rightarrow\\
&\text{bool}([s,c,v' := f(s_v), f(c_v), f(v_0)]BAP_{E^0}(s,c,v')) = TRUE \wedge\\
&\text{bool}([s,c,v := f(s_v), f(c_v), f(v_0)]Inv(s,c,v)) = TRUE
\end{aligned}
$$

3 Evaluate events' guards:
Let v_j be the current value of variables v^t, $x_{E^i j}$ be the current value of parameters $x^t_{E^i}$,
Condition 3 (Events' Enabledness) For some events,

$$
\begin{aligned}
&eval(f_{E^i}.f_{Grd}(x_{E^i j})) == true\\
&\Rightarrow\\
&bool([s,c,v,x_{E^i} := f(s_v), f(c_v), f(v_j), f(x_{E^i j})]Grd_{E^i}(x_{E^i}, s, c, v)) = TRUE
\end{aligned}
$$

4 Random choice of an event to execute:
Let v_{j+1} be the next value of variables v^t, f_{E^i} be a random choice event which satisfies Condition 3,
Condition 4 (Reachable state)

$$
\begin{aligned}
&eval(v_{j+1} = f_{E^i}.f_{Act}(x_{E^i j}));\\
&eval(f_{Inv}()) == true\\
&\Rightarrow\\
&\text{bool}([s,c,v,v',x_{E^i} := f(s_v), f(c_v), f(v_j), f(v_{j+1}), f(x_{E^i j})]BAP_{E^i}(x_{E^i}, s, c, v, v')) = TRUE \wedge\\
&\text{bool}([s,c,v := f(s_v), f(c_v), f(v_{j+1})]Inv(s,c,v)) = TRUE
\end{aligned}
$$

9.2.4 Correctness of Simulation

Definition 2 (Traces of Execution Simulator) *A simulation trace is a finite sequence* $f_{e_0}(); f_{e_1}(x_{f_{e_1}}); f_{e_2}(x_{f_{e_2}}); \ldots; f_{e_n}(x_{f_{e_n}})$ *of event firings such that:*

$$\bigwedge_{i=1}^{n} eval(f_{e_i}.f_{Grd}(x_{f_{e_i}})) == true$$

$TraceExecution(P)$ *denotes the set of all simulation traces of the simulator P.*

Definition 3 (Simulation Correctness) *Let* $f_{Trace} \in f_{Evts} \rightarrow Evts$ *the function which relates JavaScript events with their Event-B counterparts and* map *the function which applies its functional argument to all elements in a sequence. The simulator P is a correct simulation of the model M if*

$$\forall t \cdot t \in TraceExecutions(P) \Rightarrow \mathsf{map}(f_{Trace}, t) \in Traces(M)$$

Theorem 1 *If the execution of the simulator P satisfies the conditions 1, 2, 3, and 4, then the simulation is correct.*

Proof Let t be a trace of the simulator P which satisfies the four conditions 1, 2, 3, and 4:

$$t = f_{e_0}(); f_{e_1}(x_{f_{e_1}}); f_{e_2}(x_{f_{e_2}}); \ldots; f_{e_n}(x_{f_{e_n}})$$

Let $v_j (j \in 0..n)$ be the value of the state after executing the event f_{e_j}. We construct a trace with event occurrences $e_0; e_1; e_2; \ldots; e_n$ and intermediate values (preserved the invariant)

$$f(v_0); f(v_1), f(x_{f_{e_1}}); f(v_2), f(x_{f_{e_2}}); \ldots; f(v_n), f(x_{f_{e_n}})$$

that satisfies property 1.

9.2.5 Proof Obligations

The production of a simulation involves automated parts and human interactions. Proving the correctness of the former parts is the classical problem of compilers' correctness. We will not deal with this issue in this thesis; we assume it as hypothesis 3.

Hypothesis 3 *There exists a complete and correct syntactical translation mechanism from all elements in an Event-B model toward a JavaScript program.*

Human actions are more interesting as they touch a critical issue: how to introduce safely informal actions into a formal method? For each identified informal action, we propose a definition of correctness in the form of a specific PO. Assessing the correctness of a given simulator amounts to discharge a series of POs generated from the user-provided values and code. Some of those POs can be expressed and discharged in Event-B, but others cannot and need to be discharged by a classical semantic reasoning on JavaScript programs.

Considering a trace of a simulation $t = f_{e_0}(); f_{e_1}(x_{f_{e_1}}); f_{e_2}(x_{f_{e_2}}); \ldots; f_{e_n}(x_{f_{e_n}})$; we construct a trace $e_0; e_1; e_2; \ldots; e_n$ for the translated Event-B machine with
– values of sets and constants $f(s_v), f(c_v)$
– intermediate state values $f(v_0), f(v_1), f(v_2), \ldots, f(v_n)$
– parameter values $f(x_{f_{e_j}})(j >= 1)$ for event e_j
From Definition 3, we can derive the following POs to establish the correction of the simulation.

PO 1 (Valuation of Constants – Event-B)

$$bool([s, c := f(s_v), f(c_v)]Axm(s, c)) = TRUE$$

PO 2 (Valuation of Events Parameters – Event-B)

$$\bigwedge_{j=1}^{n} bool([s,c,v,x_{e_j} := f(s_v), f(c_v), f(v_j), f(x_{f_{e_j}})]Grd_{e_j}(x_{e_j},s,c,v)) = TRUE$$

The POs associated with the hand-coded functions provided by the users depend on their role and place. There are four cases.

1. Parameter value generators: this kind of function is just a facility to run the simulation. The produced values are actually fed to the guard-functions of the event; so, since we assume the translation is correct, only legal parameter values will make the guard true. So, the only requirement on parameter generators is that they produce fairly consistent and reasonable values efficiently.
2. Predicate in an invariant or a guard: let f_Ψ be a user implementation of a particular predicate Ψ. The basis of the PO is to show that f_Ψ is equivalent to a naive translation of Ψ. Discharging such PO requires to reason at the level of the JavaScript programs:
   PO 3 (Invariant and Guard – JavaScript)
   $$eval(f_\Psi()) == true \quad \Leftrightarrow \quad bool(\Psi) = TRUE$$
3. Value returned by an action: let f_{act} be the user implementation of a particular action act. The PO ensures that the set of computed values are admissible values:
   PO 4 (Action – Event-B)
   $$\{f(v_0) \mid v_0 = f_{act}()\} \subseteq \{v' \mid BAP_{act}(x_{E^i},s,c,v,v')\}$$
4. Function defined by properties in a context: this situation requires to transform each property into a program whose correction must be established. A property defining the functional constant g has the form
   $$\forall v_1,...,v_n \cdot type(v_1) \wedge ... \wedge type(v_n) \Rightarrow \Psi(g,v_1,...,v_n)$$
   where Ψ may contain several application of g, and $type(v_k)$ is a typing predicate for v_k. Let g_{user} be the user implementation of g, $m \in \mathbb{N}_1$ be the number of occurrences of g in Ψ, $l \in \mathbb{N}_1$ be the number of parameters of g, the transformation is sketched in the following algorithm:

```
Program Prog =
   for each call to g (i_th call, 1 ≤ i ≤ m)
      for each argument of the call (j_th parameter, 1 ≤ j ≤ l)
         generate a fresh variable a_ij
         generate instruction a_ij = translation of j_th argument expression
      generate a fresh variable for result r_i
      generate r_i = g_user(a_i1, ..., a_il)
   translate the typing expressions to f_type(v_k)
   translate the predicate Ψ to f_Ψ, replacing each call of g
      by their corresponding immediate evaluation result r_i
```

   The following PO ensures the correction of g_{user}:
   PO 5 (Uninterpreted Function – JavaScript)
   $$wp(Prog, f_\Psi) \Rightarrow \bigwedge_{k=1}^{n} f_{type}(v_k)$$

9.3 Discussion about the Hypotheses

The correctness of simulations is based on three hypotheses which focus on the correct implementation of JeB itself. These hypotheses must be correct. In following, we give some reasonable explications of our hypotheses. We are confident that these hypotheses can be proven. In practice, such proofs would use the well established techniques, for examples: unit and integration tests, runtime checking, code coverage, comparison with mathematical laws, comparison with other tools ProB/Brama/AnimB. This work is out of scope of this thesis.

9.3.1 Hypothesis 1

Hypothesis 1 assumes we can find a mapping function f from all values V^t used in JavaScript simulations to the Event-B value spaces V. First, we list the Event-B types from which all Event-B mathematical objects are constructed.

In Event-B, there are three kinds of basic data types:

- *IntegerType*: $\mathbb{Z}$ is the set of all integers.
- *BooleanType*: *BOOL* is the set of booleans, which has two elements *TRUE* and *FALSE*.
- *GivenType*: Carrier sets are user defined types.

Furthermore, two other kinds of data types can be constructed from any type α and β:

- *PowerSetType*: $\mathbb{P}(\alpha)$ contains the sets of elements of α.
- *ProductType*: $\alpha \times \beta$ is the set of pairs where the first element is of type α and the second element is of type β.

In JavaScript, we use *objects* to construct the concrete types used in simulations.

- Object $\$B.Integer$: The JavaScript `Integer` object is an implementation of arbitrary precision integers; the mapping function is defined as $\{i \cdot i \in \mathbb{Z} \mid \$B.Integer(\text{"i"}) \mapsto i\}$.
- Object *Boolean*: the JavaScript `Boolean` type represents a logical entity having two values, called *true* and *false*, the mapping function is defined as $\{true \mapsto TRUE, false \mapsto FALSE\}$.
- Object $\$cst.S$: we construct an object $\$cst.S$ for each carrier set S, the detailed data structure of $\$cst.S$ is specified by users, for every instance object S_i of $\$cst.S$, we have: S_i *instanceof* $\$cst.S \Rightarrow f(S_i) \in S$.

Let Set_A, Set_B be two concrete set objects. We can recursively construct Event-B power set values and product values from the following JavaScript objects:

- Object $\$B.PowerSet$: $\$B.PowerSet(Set_A)$ represents the power set of $\mathbb{P}(f(Set_A))$.
- Object $\$B.CartesianProduct$: $\$B.CartesianProduct(Set_A, Set_B)$ represents the concrete Cartesian product $f(Set_A) \times f(Set_B)$.

9.3.2 Hypothesis 2

Hypothesis 2 assumes that the interpretation of the translated Event-B formulas in JavaScript is correct. Therefore, there exists a relation which defines the observational equivalence between the Event-B formula interpretation and the JavaScript formula interpretation. With this hypothesis, we can get the semantic correctness for the translated Event-B formulas.

The Event-B modeling language is based on very classic forms of set-theory and logic. As SETL [Schwartz 1986] has shown, we know how to implement correctly operational interpretations of those theories. So, our hypothesis is sound.

In our case, the critical point, with respect to the hypothesis, is the correctness of the implementation of the library detailed in Appendix C. Although we did not conduct a formal proof, the careful definition of the implementation rules of each API makes us confident about the correctness.

9.3.3 Hypothesis 3

Unlike Hypothesis 2 which expresses a property at the semantic level, Hypothesis 3 just requires a form of correctness at the syntactical level. As above, the critical issue is the library and its APIs. Each node in a concrete Event-B syntax tree should be automatically mapped to an API defined in JavaScript. The structure of an Event-B syntax tree should be the same with the generated JavaScript expression. The Appendix B defines the details of such mappings to assure the syntactical correctness.

9.4 Summary

JeB allows one to easily build simulators. Once a simulation is set up, the issue becomes whether it is “safe” to use it as an observation tool of the model. This, of course, depends on the kind of usage we want to put simulation in.

To be used for the validation of models, simulations must guarantee that any behavior of the simulation is a behavior specified in the model. Casting this intuition into a formal frame requires to define behaviors, observations and the relation between the Event-B code and the JavaScript code.

This chapter defined proof obligations to guarantee the correctness of using simulation for the validation. Some POs must be discharged in the JavaScript world, others are expressed in the Event-B world. How to automatically generate them, how to integrate them into the Rodin platform, will be an interesting research field in future.

Chapter 10

Conclusion and Future Work

Contents

10.1 Conclusion

The starting point of our work was an assessment of the usability of Even-B. We studied the development of platooning control algorithms, and more specially how it scaled up when we move from a simplified 1D version to a more realistic 2D version. The critical analysis of the 1D platooning model uncovered some anomalous behaviors which we traced down to a deadlock problem. The difficulty of expressing the deadlock-freeness theorems in Event-B motivated us to develop a tool, the generator of DLF theorems, to automatically construct these theorems.

Our assessment confirmed that the mathematical proofs are not sufficient to assure the correctness of a formal specification. A formal specification should also be validated. We believe that the validation activities, like the verification activities, should be associated with each refinement during the development. To do that, we need better validation tools. The state-of-the-art tools which can execute Event-B models failed in the 2D platooning model: there is too much non-determinism in the early refinements. Therefore we designed and implemented a new execution tool, JeB, which is a JavaScript simulation framework for Event-B. JeB provides developers with a Rodin translator plug-in which translates Event-B to JavaScript, with a runtime which supports the mathematical notations of Event-B and the execution model, and with interfaces to add hand-coded pieces of code to reduce the non-determinism. The generated simulators run inside common Web browsers. They can be used to validate all machines in a chain of refinements.

The generated part of the simulators and the runtime environment are straight interpretations of the formal operational semantics of Event-B. They can be safely used for simulation. The hand-coded additions, which are necessary to execute the simulations, introduce a risk of implementing an incorrect model. To be useful as validation tools, simulations must ensure that any observed behavior of the simulation is a behavior specified in the model. Therefore, we have defined a set of proof-obligations which, when discharged, guarantee the correctness of the simulations with respect to the model.

10.2 Future Work

10.2.1 Technique

The current implementation of the JeB framework can be seen as a prototype. To reach an industrial-strength level, work is needed on two main areas.

The default implementation of the JavaScript library for Event-B used in the JeB tool focused on two points : 1) the full support of Event-B mathematical notations; 2) the interpretation to satisfy the Event-B language semantic. So efficiency was not our main goal in the first step. In the future, we must improve its efficiency so that the JeB tool can be adopted by industrial projects.

At present, the proof obligations to guarantee the safe usage of simulations (see Chapter 9) must be manually generated. They are not integrated into the Rodin platform. We need a tool to automatically generate and manage them.

Those two technical points open interesting and difficult research questions which concern the proofs of properties mixing JavaScript and Event-B objects. The library, implemented in JavaScript, should be proved against its specification, written in Event-B. Some proof obligations on the simulator require users to work with expressions on both languages. There is a strong need for a theory and the associated provers.

10.2.2 Refinement Process for Event-B

Like any complex artifact, *good* formal models need a precise and definite construction process. A *good* formal model should have the following properties: (a) it is logically consistent, (b) it has proven functional properties, (c) it meets non-functional properties, and (d) it is a reasonable model of the problem.

A formal refinement is the keystone around which the B method is designed. Its embodiment into the language and the support tools allows one to develop pieces of software where an implementation is proven against its specification. Refinements break down the verification process into discharging many, but small, proof obligations. So issues (a) and (b) are well taken care of by Event-B. To deal with the issues (c) and (d), i.e., the validation of the specification, we propose an extended refinement process depicted in Figure 10.1.

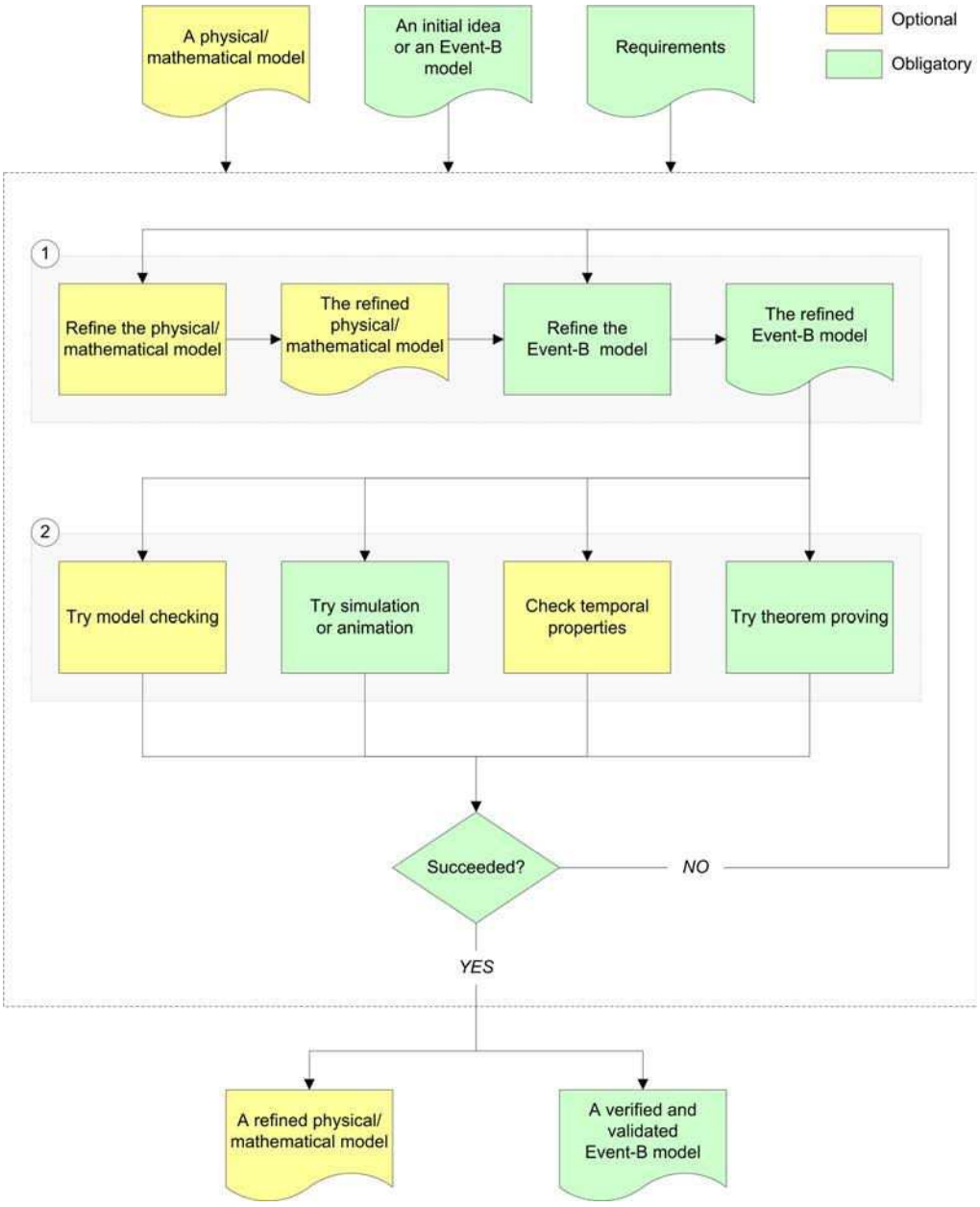

Figure 10.1: A step of refinement for Event-B

The core idea is to associate the validation activities with the refinement chain to incrementally construct a system. The mathematical proofs can guarantee the coherence within successive refinements, while the validation activities ensure that the system satisfies the user purpose at each level of refinements. A development step is then composed of different activities:

- Refinement of the physical/mathematical model. This activity is optional. It depends on the complexity of the developed system. The mathematical expressions are refined so that they can be translated into Event-B and lead to provable properties. In Section 5.3, we gave some examples extracted from the platooning case study.
- Refinement of the Event-B model. This activity is obligatory. It is recommended to make small refinements at each step. The requirements are then gradually introduced into the formal model. Different refinement strategies can be applied: for instance first, the functional and then, the safety requirements. The choice of the strategy relies on the experience of the developer and of the kind of system to develop.
- Simultaneous verification and validation. The order of verification and validation activities is not important, our final goal is to achieve a verified and validated formal specification. We can simultaneously carry out the verification and validation activities, but it is common sense to begin with the cheapest ones. In our experience, the ProB can be applied first to quickly check Event-B models, e.g., deadlocks, invariant violations. JeB (see Part II) can be used after to make "quick and dirty" simulations to help "fix" the expression of complex invariants and guards. Once the refinements seem reasonable, we must discharge all proof obligations. When needed, the generator of DLF theorems (see Chapter 4) can be used to prove the deadlock-freeness property. Last, a complete validation with JeB could be realized.

Event-B was designed around the idea of mastering verification; it encourages a linear waterfall-like development process. Our propositions, both at the process level and at the tool level (JeB), open up some possibilities such as intermediate deliveries. We believe that some of the best ideas of the current process practice could be incorporated to make an Agile-like formal refinement process. Of course, this raises many research issues.

10.2.3 Methodology

Historically, formal methods have been viewed as an exclusive alternative to traditional development methods. The whole software system needed to be developed from scratch. The development process should start with an abstract model of the whole system and proceed by formal refinements and proofs of correctness towards a final implementation. This is not a realistic way to introduce formal methods into the practice in industry. A better approach is to integrate formal methods into the traditional development process and focus on the critical parts of a system. Formal methods complement and improve existing development practices in an evolutionary way.

We think that there is no "one true way" that can be adopted for all kinds of system developments. We must consider the total costs, the time constraint, the experience of developers, the acceptance of final users, and then use different approaches for the development of different system parts. For non-critical system components, we can use

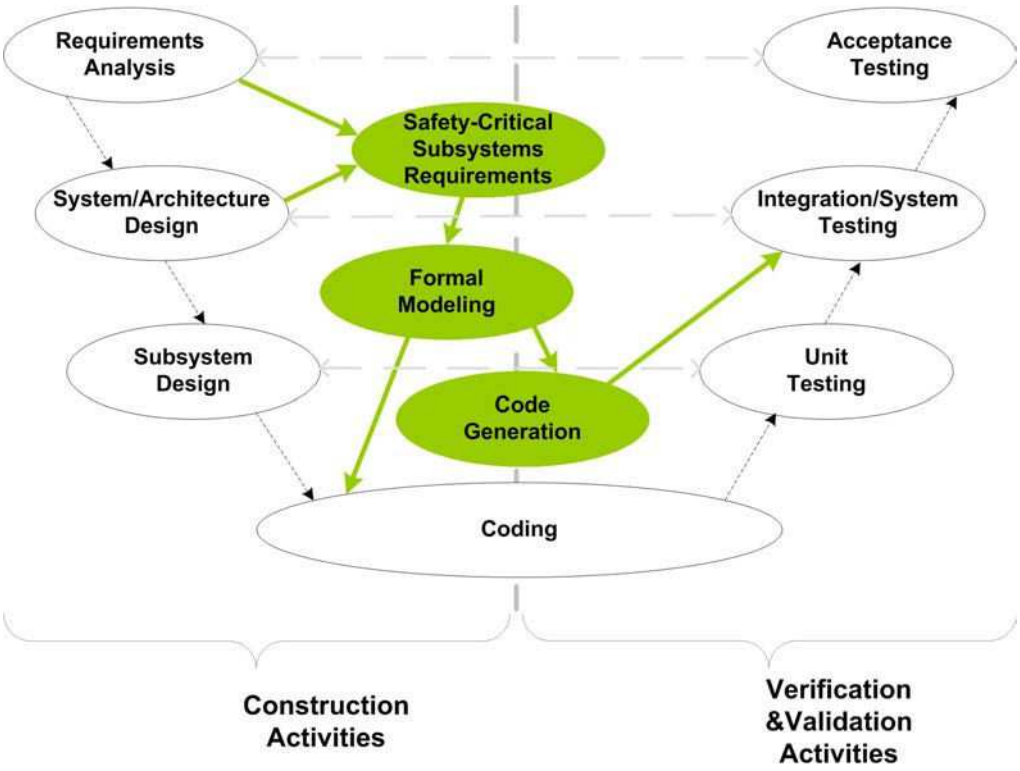

Figure 10.2: An extended V-Model

standard approaches to develop them. For critical system components, we should use the formal methods to gain a high reliability and robustness of the system components.

We propose an extended V-Model depicted in Figure 10.2 to use formal methods to develop the safety-critical subsystems. First, we need to extract the requirements of the safety-critical subsystems so they can be cast as an initial formal model of the critical properties. Then we refine this initial model into more and more concrete specifications. If the last formal specification contains enough implementable elements, it may be directly translated into executable code. Otherwise it can be used as the precise specification against which the code should be assessed. Last, some integration and system testings are necessary to assure the validity and compatibility of the formally developed part within the whole system.

The extended V-Model mixes informal methods and formal methods to develop complete systems. We must also consider the formal development itself to address its weaknesses. [Abrial 2006] discusses two of them and we believe that there is a third weakness.

The first weakness is the quality of the requirements document. Formal methods are not meant to obviate the need of strong requirement analysis. A complete, unambiguous and consistent requirement document is of utmost importance for formal methods. But the requirements document is usually written as a semi-formal text mixing natural language, diagrams, drawings, and mathematical formulas. Such texts are notoriously difficult to analyze for properties such as completeness and consistency. Therefore errors or

omissions may affect the initial abstract model.

The second weakness is the code generation. The last refinement may be translated into executable code in two steps: 1) the concrete model is translated into a classical programming language code; 2) the code of the first translation is then translated into an executable code by a usual compiler. Very few compilers [Leroy 2009a, Leroy 2009b] such as the CompCert project have actually be proven to make correct translations .

The third weakness is that we have a fixed and complete set of requirements at the beginning of the development. We know that for most projects, new requirements are discovered and formulated as the development proceeds. The maturation of the requirements has several sources: decisions are needed to orient the refinements, emergent behaviors appear and must be controlled, users may prompt for new requirements as they understand better the impact of the future system, etc.

Proponents of formal methods must acknowledge those weaknesses which, from an industrial perspective, are risks to be managed. There is strong need of research on these questions [Romanovsky 2013].

Appendix A

Présentation de la thèse en français

A.1 Motivation

Contexte de la recherche Les approches classiques de développement de systèmes sont principalement basées sur des activités humaines. Malheureusement, *errare humanum est* (l'erreur est humaine), ce qui est source de problèmes relatifs à la sécurité.

La qualité des systèmes peut être améliorée en contrôlant les erreurs humaines pendant le processus de développement, par exemple, en appliquant des méthodes formelles lors du développement de sous-systèmes critiques. Les méthodes formelles permettent de raisonner rigoureusement sur des logiciels et sur leur construction à l'aide de la logique mathématique. Elles permettent d'obtenir une bonne garantie relativement à l'absence de « bug » dans les logiciels. Elles permettent aussi d'assurer qu'une implantation d'un logiciel est conforme à sa spécification.

Dans cette thèse, un système est dit être *correct* s'il satisfait à deux conditions :

– *Le système est vérifié : le système doit être cohérent et son implantation doit être conforme à sa spécification initiale. La vérification peut s'effectuer grâce à des techniques de preuves lorsqu'une méthode formelle est utilisée. Les activités de vérification font généralement partie des processus exécutés par les développeurs.*
– *Le système est validé : le système doit remplir l'objectif que les utilisateurs en attendent. Les activités de validation exigent des processus qui impliquent des personnes externes à l'équipe de développement.*

La vérification et la validation sont des procédures indépendantes. L'utilisation des méthodes de spécification formelle permet de garantir plus facilement qu'une spécification est vérifiée. Toutefois, une spécification vérifiée n'est pas automatiquement une spécification validée.

Les vérifications les plus rigoureuses relèvent de démonstrations mathématiques ; elles exigent des ressources importantes (en temps, en coût et en expérience). Ainsi, les méthodes formelles paraissent réservées au développement de systèmes critiques pour la sécurité, où le coût de la recherche des défauts est très élevé. Par exemple, dans

le domaine des chemins de fer ou dans le domaine de l'aérospatiale, des erreurs non détectées peuvent causer la perte de vies humaines.

Cette thèse vise à étendre le domaine d'utilisabilité des méthodes formelles en général et B événementiel en particulier. Pour cela, en partant d'études de cas réalistes (la modélisation et le développement formel d'un algorithme de *platooning*), nous avons identifié un certain nombre de freins à l'usage de B événementiel et nous proposons deux outils permettant de faciliter l'usage de cette technique formelle.

Problème scientifique Le frein principal découvert à partir des études de cas concerne la validation :

> *La preuve mathématique d'une spécification formelle ne suffit pas à garantir sa correction : la vérification n'entraîne pas sa validation.*

Scientifiquement, le problème revient alors à comprendre les activités qui relèvent de la validation, à concevoir et réaliser les outils qui assistent les utilisateurs dans ces activités et à intégrer la validation dans la démarche générale associée à B, c'est-à-dire, le développement par raffinement formel.

Les techniques de développement basées sur la preuve et le raffinement sont conçues autour des activités de vérification. En particulier, ces techniques garantissent que tous les modèles construits pendant le développement répondent à la spécification initiale. Cependant, les exigences non-fonctionnelles sont souvent très difficiles à exprimer dans les logiques classiques ; certaines exigences sont en dehors du cadre mathématique. En conséquence, elles ne sont pas exprimées dans le cahier des charges initial. De plus, l'obtention d'exigences claires et complètes à la phase initiale du développement est connue pour une tâche quasi-impossible. En conséquence, le cahier des charges initial peut être incomplet, ambigu et incohérent. Les développeurs auront à compléter les exigences absentes afin de prendre les décisions nécessaires à leur réalisation. Naturellement, comme ces « exigences complémentaires » sont absentes de la spécification initiale, les preuves liées à la vérification ne peuvent que s'assurer de leur cohérence vis-à-vis du reste du modèle ; les preuves ne peuvent pas assurer que les nouvelles exigences sont compatibles avec ce que souhaitent les utilisateurs. Il est essentiel qu'elles soient validées, de préférence le plus tôt possible après leur vérification.

Problèmes techniques L'amélioration de l'utilisabilité de B événementiel nécessite de répondre à deux problèmes relatifs à la mise en œuvre de la méthode :

- *l'absence d'outils pratiques pour aider les activités de validation,*
- *l'absence d'un guide pour intégrer le raisonnement formel et le raisonnement semi-formel dans un processus de développement.*

L'usage d'une méthode formelle est directement lié à la qualité des outils disponibles pour sa mise en œuvre. Nous avons donc consacré une part importante de notre réflexion et de notre travail à la conception et à la réalisation d'outils concrets qui s'intègrent efficacement dans une démarche rigoureuse.

Objectifs Notre travail concerne les objectifs liés à l'extension de l'utilisabilité de B événementiel :

1. L'objectif le plus important concerne un environnement qui permet de simuler le modèle B événementiel pour la validation et permet de garantir la correction sémantique de simulations. Cet environnement est un complément aux outils existants qui permet d'associer plus étroitement la validation avec la vérification dans le cadre des raffinements.
2. Un second objectif concerne l'assistance à la vérification de l'absence de *deadlock* dans les spécifications.
3. Le dernier objectif, plus méthodologique, vise à penser et étendre la notion de raffinement utilisant dans B événementiel. Le raffinement d'état de l'art souligne les activités de vérification. Nous le complétons avec d'autres activités, telles que l'évolution des exigences, l'adaptation d'équations, la validation par l'animation ou la simulation notamment.

A.2 Contribution

Dans ma thèse, nous avons défini deux contributions importantes :

Contribution I : évaluation de l'utilisabilité de B événementiel

Le point de départ de notre recherche est d'évaluer l'utilisabilité de B événementiel :

– une analyse critique de la spécification du *platooning* en 1D et une explication de certaines anomalies dans le comportement,
– un *plug-in* Rodin (environ 500 lignes en Java) qui génère automatiquement des théorèmes de *Deadlock-freeness*,
– une évaluation de quelques aspects du passage d'un modèle en 1D à un modèle en 2D qui étudie la question de la « montée en charge » de B événementiel.

Contribution II : un outil de simulation en JavaScript pour B événementiel

Notre contribution principale est un cadre de simulation pour B événementiel, basé sur JavaScript. Cet outil permet de construire à peu de frais des simulateurs du modèle B événementiel grâce à l'utilisation d'un traducteur de B événementiel vers JavaScript, un environnement graphique d'exécution, et la possibilité de compléter ce que la traduction n'a pas été en mesure de traiter. Les simulateurs peuvent être utilisés pour valider le modèle B événementiel à chaque étape de raffinement. Ils aident les utilisateurs finaux, les experts du domaine et les développeurs afin de mieux comprendre les modèles mathématiques et les spécifications réalisées.

Cette contribution a deux aspects :

– un environnement intégré de simulation pour les modèles B événementiel, réalisé à l'aide de deux composants principaux :
  – un traducteur (environ 2800 lignes en Java) implanté comme un *plug-in* Rodin qui génère automatiquement des simulateurs du modèle B événementiel et

  - un environnement d'exécution et une bibliothèque (environ 2700 lignes en JavaScript) qui prend en charge toutes les notations mathématiques du B événementiel ;
- une sémantique pour s'assurer de la correction des simulations utilisant une notion d'obligations de preuve. Grâce à cette sémantique, nous pouvons rendre exécutables des modèles très abstraits en les simulant de manière sûre sur des sous-domaines.

A.3 Études de cas

Les études de cas sont un élément crucial de notre démarche scientifique. Plusieurs ont été utilisées dans ce travail ; quatre ont une importance plus particulière.

- Modèle du platooning en 1D : il développe un algorithme de contrôle longitudinal du platooning ; il contient 10 composants et environ 600 lignes de code B événementiel. Nous avons analysé ligne à ligne et proposé plusieurs corrections pour cette spécification développée dans de l'équipe.
- Modèle du platooning en 2D : il étend le modèle du platooning en 1D précédant en ajoutant le contrôle latéral ; il contient 10 composants et environ 1800 lignes de code B événementiel. L'écriture, la vérification et la validation de cette spécification que nous avons complément réalisées ont mis à jour des questions scientifiques et techniques.
- Modèle du domaine de transport : il modélise les propriétés du domaine du transport terrestre ; il contient 23 composants et environ 1100 lignes de code B événementiel. Ce modèle a été développé dans l'équipe, et nous a permis de tester notre outil de validation.
- Modèle du MIDAS : il modélise un système de microprocesseurs avec l'abstraction de données ; il contient 104 composants et environ 21800 lignes de code B événementiel. Cette étude de cas développée dans une autre équipe de recherche présente deux intérêts principaux : sa taille qui permet de contrôler que nos outils peuvent traiter de cas réalistes, ainsi que son « externalité » qui permet de contrôler que nos outils sont utilisables hors de notre contexte immédiat.

A.4 Publications

Conférences internationales

[1] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. Proving the Fidelity of Simulations of Event-B Models. In *The 15th IEEE International Symposium on High Assurance Systems Engineering (HASE)*, Miami, USA, forthcoming 2014.

[2] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. JeB : Safe Simulation of Event-B Models in JavaScript. In *The 20th Asia-Pacific Software Engineering Conference (APSEC)*, Bangkok, Thailand, 2013.

[3] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. The Case for Using Simulation to Validate Event-B Specifications. In *The 19th Asia-Pacific Software Engineering Conference (APSEC)*, Hong Kong, China, 2012.

[4] Faqing Yang and Jean-Pierre Jacquot. Scaling Up with Event-B : A Case Study. In Mihaela Bobaru, Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods (NFM)*, volume 6617 of *Lecture Notes in Computer Science*, pages 438–452. Springer Berlin / Heidelberg, 2011.

[5] Faqing Yang and Jean-Pierre Jacquot. An Event-B Plug-in for Creating Deadlock-Freeness Theorems. In *14th Brazilian Symposium on Formal Methods (SBMF)*, São Paulo, Brésil, 2011.

Conférences nationales

[6] Faqing Yang and Jean-Pierre Jacquot. JeB : un environnement de simulation en JavaScript pour B événementiel. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Nancy, France, 2013.

[7] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. Traduction de B événementiel en C pour la validation par la simulation. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Grenoble, France, 2012.

[8] Faqing Yang and Jean-Pierre Jacquot. Prouvé ? Et après ? In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Poitiers, France, 2010.

Résumés d'article pour ateliers

[9] Faqing Yang and Jean-Pierre Jacquot. Generating Executable Simulations from Event-B Specifications. In *Rodin User and Developer Workshop*, Fontainebleau, France, 2012.

[10] Faqing Yang and Jean-Pierre Jacquot. An Event-B Plug-in for Creating Deadlock-Freeness Theorems. In *Rodin User and Developer Workshop*, Fontainebleau, France, 2012.

[11] Atif Mashkoor, Faqing Yang and Jean-Pierre Jacquot. Validation of formal specification : The case for animation. In *3rd Workshop on Security and Reliability (SecDay)*, Trier, Germany, 2011.

A.5 Aperçu de l'ensemble des chapitres

État de l'art Dans ce chapitre, nous présentons un état de l'art sur les processus de développement, les méthodes formelles, la méthode B et des modèles formels du *platooning* existants. Le processus de développement est présenté avec ses activités principales. Les différents processus sont définis par la structure et la pratique de leurs activités. Les méthodes formelles utilisent la logique mathématique pour raisonner rigoureusement sur des logiciels et sur leur construction. La mise en œuvre des méthodes formelles est généralement coûteuse en ressources et ces méthodes sont actuellement réservées pour les logiciels critiques. La méthode B est une méthode formelle de développement de logiciels et de systèmes. Elle permet d'écrire les spécifications de façon abstraite et mathématique avec le langage B. Elle permet d'obtenir une implantation concrète par l'application d'une suite de raffinements.

Analyse de la spécification du platooning en 1D Le modèle du *platooning* en 1D est le point de départ de nos travaux de recherche. Nous avons fait une analyse approfondie de ce modèle afin d'acquérir une meilleure compréhension de la relation entre la preuve d'un modèle formel et sa correction au sens général. Nous avons découvert la présence de *deadlock* liés à une faiblesse des invariants. L'invariant pour exprimer la propriété de non-collision doit avoir la forme suivante :

$$\begin{array}{l}\forall v \cdot (v \in 2\,..\,VEHICLES \Rightarrow \\ \quad xpos(v-1) - xpos(v) > CRITICAL\_DISTANCE + brake\_distance \\ )\end{array}$$

où *brake_distance* est une fonction basée sur la vitesse de véhicule et la décélération maximale.

Générateur de théorèmes de Deadlock-Freeness L'absence de blocage est une propriété essentielle pour certains modèles. Les théorèmes de *Deadlock-freeness* sont connus mais ils ne sont pas pris en compte actuellement pas les environnements supports de B événementiel. Nous avons développé un outil pour la plate-forme Rodin qui génère automatiquement des théorèmes de *Deadlock-freeness* pour un sous-ensemble d'événements dans un modèle B événementiel.

Passage à l'échelle avec B événementiel Ce chapitre concerne notre expérience avec la modélisation d'un algorithme réaliste pour le contrôle des véhicules autonomes en B événementiel. Le passage d'un modèle en 1D à un modèle en 2D ne semble pas introduire un grand changement : on ajoute le contrôle latéral. Mais celui-ci requiert l'introduction de notions telles que les fonctions trigonométriques et les courbes sur le plan qui posent une vraie difficulté en B événementiel. Ce chapitre analyse plusieurs aspects du passage de 1D à 2D : la structure du modèle, l'adaptation d'équations, les propriétés temporelles et l'adaptation des outils. Nous avons résolu certains problèmes concernant la cohérence entre les raffinements et l'adaptation d'équations. Le langage B événementiel ne comporte pas les notions de temps ou de logique temporelle. Nous analysons les problèmes et leurs solutions pour notre étude de cas. Enfin, nous analysons pourquoi les animateurs actuels n'ont pas réussi à exécuter le modèle du *platooning* en 2D.

Conception de JeB JeB est un cadre d'assistance à la validation de modèles écrits en B événementiel. Ce formalisme possède une sémantique opérationnelle bien définie et la validation consiste à observer l'exécution du modèle. Dans la pratique, le non-déterminisme et le haut niveau d'abstraction qu'il est conseillé d'utiliser dans les spécifications et leurs premiers raffinements, empêchent d'automatiser totalement l'exécution.

L'idée principale proposée par JeB est que la construction et l'exploitation de simulations sont un processus de collaboration entre les outils et les humains : la traducteur de JeB génère le code principal et les humains surmontent les difficultés liées au non-déterminisme. L'architecture présentée dans la Figure A.1 est le reflet de cette idée.

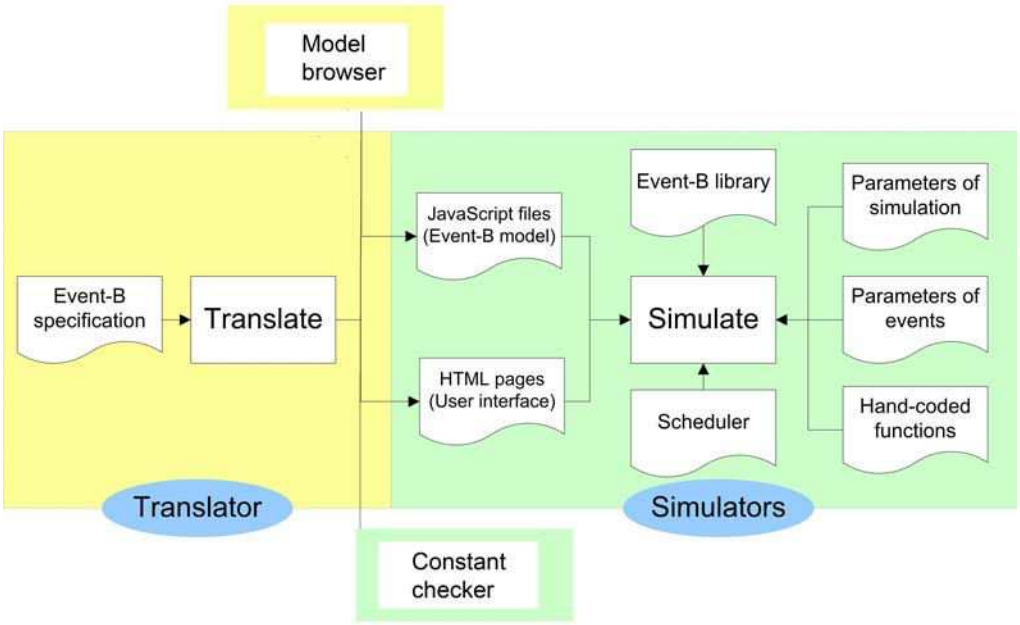

FIGURE A.1 – L'architecture de JeB

JeB est composé de deux parties :

1. Le traducteur est défini comme un *plug-in* Rodin ; il est activé par l'utilisateur. Le traducteur parcourt les machines et les contextes B événementiel ; il génère le code JavaScript permettant d'exécuter le modèle ainsi qu'une interface HTML permettant de visualiser l'exécution. Des outils annexes comme le navigateur de projets B événementiel et le vérificateur de constantes sont également générés.
2. Les simulations nécessitent l'utilisation de quatre éléments :
   - Les programmes générés pas le traducteur forment l'essentiel du code qui sera exécuté. Chaque simulateur inclut une page HTML pour l'interface graphique et une fichier JavaScript pour la spécification de la machine.
   - La bibliothèque B événementiel est une bibliothèque en JavaScript basée sur le langage de la théorie des ensembles et de la logique du premier ordre. Elle prend en charge toutes les notations mathématiques de B événementiel et interprète les formules pendant l'exécution.
   - L'ordonnanceur gère les interactions de l'utilisateur et ordonnance l'exécution des événements déclenchables.
   - Les fichiers de configuration définissent les paramètres globaux des simulations, les paramètres des événements particuliers et les fonctions codées à la main.

Cette conception permet d'isoler les parties qu'on sait traiter automatiquement de celles qui requièrent l'intervention de l'utilisateur. En particulier, la résolution des difficultés

liées au non-déterminisme par l'insertion de solutions ad-hoc programmée par l'utilisateur est bien contrôlée.

Implantation de JeB Ce chapitre présente les détails techniques de notre implantation : la traduction de contextes (constantes, axiomes), la traduction de machines (variables, invariants, événements), la traduction de formules (prédicats, expressions, assignements), l'interprétation de formules (bibliothèque) et le contrôleur de la simulation (exécution).

Le non-déterminisme intrinsèque de B événementiel fait qu'à chaque pas de l'exécution, il y a en général plusieurs choix à effectuer. Nous avons donc adopté une stratégie similaire à celle des animateurs qui passe par trois étapes :
– détermination de l'ensemble des événements déclenchables (toutes les gardes sont évaluées),
– choix d'un événement à déclencher,
– réalisation de l'action de l'événement choisi.
Cette stratégie impose une traduction de l'événement qui doit séparer les gardes et les actions en deux parties. Le choix de l'événement est soit contrôlé par un ordonnanceur, soit fixé par l'utilisateur. Un ordonnanceur qui choisit aléatoirement l'événement est fourni par défaut ; l'utilisateur peut le remplacer pour implanter d'autres stratégies.

Utilisation de JeB Nous décrivons la création de simulations en détail pour les modèles du *platooning* en 1D et 2D. Nous donnons quelques observations pour les modèles utilisés dans notre développement et une analyse du point de vue de la validation. Le processus de simulation passe par quatre étapes :

– la génération des simulateurs par l'activation du traducteur de JeB (automatique),
– la valuation des constantes et la programmation des fonctions qui permettent d'automatiser les exécutions (manuelle),
– la mise en place de l'affichage graphique des états (optionnel) et
– l'observation et l'analyse des exécutions.

Concrètement, l'utilisateur doit compléter les squelettes fournis par le traducteur dans le fichier de configuration. Ceux-ci concernent :
– la valuation de constantes :

```
$cst.CRITICAL_DISTANCE = $B('20');
```

– la définition un ensemble totalement abstrait, tel un *carrier set*, par un objet :

```
$cst.Point = function(x, y, γθ, σθ, v, κ) {
    var obj = {};
    obj.x = x;
    obj.y = y;
    obj.γθ = γθ;
    obj.σθ = σθ;
    obj.v = v;
    obj.κ = κ;
    obj.__proto__ = $cst.Point.prototype;
    return obj;
};
```

– les générateurs d'arguments :

```
get_magic_xpos_vehicle = function() {
    return $B.plus( $var.xpos.value.getImage($var.vehicle.value), $B('10') );
};
```

– l'affichage graphique :

```
jeb.animator.draw = function() {
    ...
    $anim.clearRect(0, 0, $anim.canvas.width, $anim.canvas.height);
    $anim.beginPath();
    for (i=0; i<data.length; i++) {
        x = data[i].right % $anim.canvas.width;
        $anim.moveTo(x, 50);
        $anim.arc(x, 50, 10, 0, Math.PI*2, true);
        $anim.strokeText(i+1, x-5, 53);
    }
    $anim.closePath();
    $anim.stroke();
};
```

Il est intéressant de noter que le coût d'utilisation de JeB reste raisonnable et qu'il peut être utilisé en tant qu'outil de routine pendant tout le développement. Le tableau suivant résulte de nos observations sur l'usage de JeB. Il met en relation la taille des spécifications (nombre de composants et de lignes B événementiel), la taille des simulateurs permettant de les exécuter (nombre de lignes de JavaScript et HTML), la taille du code que l'utilisateur doit fournir après les générations (nombre de lignes de JavaScript), et le taux qui décrit le pourcentage du code manuel par rapport à du code total de simulation.

Modèle B événementiel	Nombre de Composants	Code B événementiel	Code de simulation	Code manuel	Taux (%)
1D Platooning	10	600	13 000	160	1.2
2D Platooning	10	1 800	28 300	460	1.6
Domaine de transport	23	1 100	22 400	470	2.1
MIDAS	104	21 800	542 300	1 300	0.2

L'utilisation de JeB met en évidence très rapidement les *deadlocks* dans les modèles du platooning. Par rapport à d'autres techniques, JeB facilite la compréhension des conditions et situations dans lesquelles apparaissent les *deadlocks.* Il est ainsi plus facile de corriger le modèle. Dans le cadre de MIDAS et du modèle des transports, nous avons observé des comportements « suspects » qui, après analyse, ont permis de découvrir des axiomes « incorrects ». Il faut noter que les axiomes sont logiquement cohérents et que les preuves se déroulent normalement ; l'incorrection concerne la sémantique exprimée par les axiomes.

Correction des simulations JeB permet de créer facilement des simulateurs en composant des éléments automatiquement générés et des éléments fournis par un utilisateur. Sachant que les interventions des utilisateurs sont une source majeure d'erreur, il est crucial, dans le cadre d'une méthode formelle, d'évaluer la confiance qu'on peut avoir dans l'observation des simulations.

Le comportement observé est-il effectivement spécifié par le texte formel ? Pour répondre à cette question, nous avons défini formellement la notion de « fidélité » par la preuve. Celle-ci utilise la notion de *trace.*

– Le comportement d'une machine M est une suite finie d'événements

$$E^0; e_1; e_2; \ldots e_n$$

tel que

$$\forall j.j \in 1..n \Rightarrow e_j \in Evts$$

et

$$\text{fis}(E^0; e_1; e_2; \ldots e_n) \Leftrightarrow true$$

où fis est le prédicat de faisabilité d'une suite d'événements. $Traces(M)$ dénote l'ensemble de tous les comportements de la machine M.

– Une trace de simulations est une suite finie

$$f_{E^0}(); f_{e_1}(x_{f_{e_1}}); f_{e_2}(x_{f_{e_2}}); \ldots; f_{e_n}(x_{f_{e_n}})$$

d'événements déclenchables tels que :

$$\bigwedge_{j=1}^{n} eval(f_{e_j}.f_{Grd}(x_{f_{e_j}})) == true$$

$TraceExecutions(P)$ dénote l'ensemble de toutes les traces de simulation d'un simulateur P.

– Soit $f_{Trace} \in f_{Evts} \rightarrow Evts$, la fonction qui associe aux méthodes JavaScript les événements B événementiel qu'elles implantent et map, la fonction qui applique ses arguments fonctionnels à tout élément dans une suite. La *fidélité* d'un simulateur P par rapport à une machine M est définie par

$$\forall t \cdot t \in TraceExecutions(P) \Rightarrow \text{map}(t, f_{Trace}) \in Traces(M)$$

En pratique, la démonstration de la propriété de fidélité s'effectue à travers la démonstration d'obligations de preuves qui sont associées à chaque élément introduit par l'utilisateur. Il y a six situations, qui produisent cinq obligations de preuve :

1. Valuation d'une constante :

$$bool([s,c := f(s_v), f(c_v)]Axm(s,c)) = TRUE$$

2. Valuation d'un paramètre :

$$\bigwedge_{j=1}^{n} bool([s,c,v,x_{e_j} := f(s_v), f(c_v), f(v_j), f(x_{f_{e_j}})]Grd_{e_j}(x_{e_j}, s, c, v)) = TRUE$$

3. Générateur d'arguments : les valeurs générées sont vérifiées par les gardes d'un événement ; il n'y a donc pas de preuve spécifique à réaliser.

4. Instantiation d'un prédicat non calculable dans un invariant ou une garde :

$$eval(f_\Psi()) == true \quad \Leftrightarrow \quad bool(\Psi) = TRUE$$

5. Valeur affectée lors d'une substitution non déterministe :

$$\{f(v_0) \mid v_0 = f_{act}()\} \subseteq \{v' \mid BAP_{act}(x_{E^i}, s, c, v, v')\}$$

6. Fonction définie par des propriétés :

$$wp(Prog, f_\Psi) \Rightarrow \bigwedge_{k=1}^{n} f_{type}(v_k)$$

où `Prog` est le programme JavaScript construit mécaniquement qui évalue l'axiome définissant la propriété.

A.6 Conclusion et Perspectives

Conclusion Dans cette thèse, nous avons évalué l'utilisabilité de B événementiel avec l'étude de cas du problème du *platooning*. L'analyse critique du modèle du *platooning* en 1D découvre les comportements anormaux dans le modèle original. La difficulté d'exprimer des théorèmes de *Deadlock-freeness* dans le B événementiel nous a motivé à développer un outil pour construire automatiquement ces théorèmes.

Le point important de ma thèse est que les preuves mathématiques ne sont pas suffisantes pour assurer la correction, au sens de l'utilisateur, d'une spécification formelle. Une spécification formelle doit être validée tout au long de la chaîne de raffinement au même titre qu'elle est vérifiée à chaque étape. Pour ce faire, nous avons besoin d'outils de validation qui couvrent les modèles les plus abstraits. Les outils d'exécution de B événementiel existants ont échoué sur le modèle du *platooning* en 2D à cause de trop de caractéristiques non déterministes dans les premiers niveaux de raffinement. C'est pourquoi nous avons défini un nouvel outil d'exécution, JeB, qui est un environnement de simulation en JavaScript pour B événementiel. JeB utilise un *plug-in* Rodin pour générer automatiquement des simulateurs à partir de modèles B événementiel. Les simulateurs générés peuvent être utilisés pour valider le modèle B événementiel à chaque étape de raffinement. Les simulateurs générés par JeB sont basés sur le modèle B événementiel, mais le code introduit par l'utilisateur peut introduire des erreurs dans le comportement de la simulation. Pour la validation d'un modèle B événementiel, la simulation doit garantir que tout comportement de la simulation est un comportement spécifié dans le modèle formel. Nous avons défini certaines obligations de preuves pour garantir la correction de l'utilisation de la simulation.

Perspectives Cette thèse pourrait être poursuivie dans trois directions :

– Technique : il s'agit d'améliorer l'efficacité de la bibliothèque pour B événementiel et d'intégrer les obligations de preuves qui garantissent la correction d'utilisation de la simulation à l'environnement (génération automatique et prouveurs adaptés) afin que l'outil de JeB puisse être adopté par les projets industriels.

– Processus de raffinement pour B événementiel : il s'agit d'associer les activités de validation dans la chaîne de raffinement. Chaque raffinement doit être vérifié et validé pour assurer sa correction. Le processus doit être utilisable sur des projets complexes.
– Méthodologie : il s'agit d'intégrer les méthodes formelles dans le processus du développement. Nous proposons un Modèle du cycle en V étendu qui intègre des techniques informelles et des techniques formelles pour développer un système complet. Les faiblesses des méthodes formelles doivent être reconnues à partir d'un point de vue industriel.

Appendix B

Translation of Event-B Formulas

Contents

B.1 Syntax Tree

Each Event-B formula has a syntax tree representation. The JeB translator uses the concrete syntax tree implemented in the Rodin platform. In this tree, each node has an unique tag (an integer constant) that can be used to identify any Event-B mathematical notation. To simplify the translator task, we developed a complete JavaScript library for Event-B (see appendix C), hence every Event-B mathematical notation has an unique interface definition in that library. The JeB translator recursively traverses the syntax

tree, maps each node of this tree to a defined interface, and finally outputs the parsed result for each formula.

In Rodin, the syntax tree is an n-ary tree where a node can have between 0 and n children. We traverse a given tree in preorder and perform the following operations recursively at each node:

1. visit the root, add the mapping interface to the parsed result
2. for $i = 0$ to n do
   (a) visit $children[i]$, if present
   (b) add the mapping interface to the parameters of its parent interface
3. return the parsed result

We use the following pattern to represent the translation rule from a concrete syntax tree node to its JavaScript output.

Event-B formula	the mathematical formula
Rodin tag name	the node tag name in the Rodin syntax tree implementation
Node type	leaf node (terminal node), unary node, binary node or n-ary node (n is specified in that node)
Node children	the children of a node, empty for a leaf node
Translation	the JavaScript parsed result

The used symbols are described below:

Symbols	**Meaning**
$E, F, E_1, \ldots, E_n$	denote expressions in Event-B
L	denote a pattern of variables in Event-B, possibly including $\mapsto$ and parentheses
$P, Q, P_1, \ldots, P_n$	denote predicates in Event-B
$S, T, S_1, \ldots, S_n$	denote set expressions in Event-B
U	denote a set of sets in Event-B
f	denote functions in Event-B
$m, n, m_1, \ldots, m_n$	denote integer expressions in Event-B
$r, r_1, \ldots, r_n$	denote relations in Event-B
$x, x_1, \ldots, x_n$	denote single variables in Event-B
α	denote a literal integer in Event-B
χ	denote an identifier (free or bound) in Event-B
$'$	denote a primed identifier in Event-B
superscript (^{TR})	denote a traversal of a child node
superscript (^{TR*})	denote a special treatment of a translation
$\$B$	denote the namespace for the JavaScript library for Event-B
$\$cst, \$var, \$arg$	denote the namespaces for the translated constants, variables and parameters, respectively
. (dot)	use to access the properties and methods of a JavaScript object
$[\ldots]$	denote a JavaScript array
$'\ldots'$	denote a JavaScript string

B.2 Relational Predicate Nodes

B.2.1 Equality Node

Event-B formula	$E = F$
Rodin tag name	*EQUAL*
Node type	binary node
Node children	E, F
Translation	$\$B.equal(E^{TR}, F^{TR})$

B.2.2 Inequality Node

Event-B formula	$E \neq F$
Rodin tag name	*NOTEQUAL*
Node type	binary node
Node children	E, F
Translation	$\$B.notEqual(E^{TR}, F^{TR})$

B.2.3 Arithmetic Less Than Node

Event-B formula	$m < n$
Rodin tag name	*LT*
Node type	binary node
Node children	m, n
Translation	$\$B.lessThan(m^{TR}, n^{TR})$

B.2.4 Arithmetic Less Equal Node

Event-B formula	$m \leq n$
Rodin tag name	*LE*
Node type	binary node
Node children	m, n
Translation	$\$B.lessEqual(m^{TR}, n^{TR})$

B.2.5 Arithmetic Greater Than Node

Event-B formula	$m > n$
Rodin tag name	*GT*
Node type	binary node
Node children	m, n
Translation	$\$B.greaterThan(m^{TR}, n^{TR})$

B.2.6 Arithmetic Greater Equal Node

Event-B formula	$m \geq n$
Rodin tag name	*GE*
Node type	binary node
Node children	m, n
Translation	$\$B.greaterEqual(m^{TR}, n^{TR})$

B.2.7 Set Membership Node

Event-B formula	$E \in S$
Rodin tag name	*IN*
Node type	binary node
Node children	E, S
Translation	$\$B.belong(E^{TR}, S^{TR})$

B.2.8 Not a Set Membership Node

Event-B formula	$E \notin S$
Rodin tag name	*NOTIN*
Node type	binary node
Node children	E, S
Translation	$\$B.notBelong(E^{TR}, S^{TR})$

B.2.9 Proper Subset Node

Event-B formula	$S \subset T$
Rodin tag name	*SUBSET*
Node type	binary node
Node children	S, T
Translation	$\$B.properSubset(S^{TR}, T^{TR})$

B.2.10 Not a Proper Subset Node

Event-B formula	$S \not\subset T$
Rodin tag name	*NOTSUBSET*
Node type	binary node
Node children	S, T
Translation	$\$B.notProperSubset(S^{TR}, T^{TR})$

B.2.11 Subset Node

Event-B formula	$S \subseteq T$
Rodin tag name	*SUBSETEQ*
Node type	binary node
Node children	S, T
Translation	$\$B.subset(S^{TR}, T^{TR})$

B.2.12 Not a Subset Node

Event-B formula	$S \not\subseteq T$
Rodin tag name	*NOTSUBSETEQ*
Node type	binary node
Node children	S, T
Translation	$\$B.notSubset(S^{TR}, T^{TR})$

B.3 Binary Predicate Nodes

B.3.1 Implication Node

Event-B formula	$P \Rightarrow Q$
Rodin tag name	*LIMP*
Node type	binary node
Node children	P, Q
Translation	$\$B.implication(P^{TR}, Q^{TR})$

B.3.2 Equivalence Node

Event-B formula	$P \Leftrightarrow Q$
Rodin tag name	*LEQV*
Node type	binary node
Node children	P, Q
Translation	$\$B.equivalence(P^{TR}, Q^{TR})$

B.4 Associative Predicate Nodes

B.4.1 Conjunction Node (Left Associative)

Event-B formula	$P_1 \land \ldots \land P_n \quad (n \geq 2)$
Rodin tag name	*LAND*
Node type	n-ary node
Node children	$P_1, \ldots, P_n$
Translation	$\$B.and(P_1^{TR}, \ldots, P_n^{TR})$

B.4.2 Disjunction Node (Left Associative)

Event-B formula	$P_1 \vee \ldots \vee P_n \quad (n \geq 2)$
Rodin tag name	*LOR*
Node type	n-ary node
Node children	$P_1, \ldots, P_n$
Translation	$\$B.or(P_1^{TR}, \ldots, P_n^{TR})$

B.5 Literal Predicate Nodes

B.5.1 True Predicate Node

Event-B formula	$\top$
Rodin tag name	*BTRUE*
Node type	leaf node
Node children	-
Translation	$\$B.bTrue()$

B.5.2 False Predicate Node

Event-B formula	$\bot$
Rodin tag name	*BFALSE*
Node type	leaf node
Node children	-
Translation	$\$B.bFalse()$

B.6 Simple Predicate Node

B.6.1 Finite Set Node

Event-B formula	$\text{finite}(S)$
Rodin tag name	*KFINITE*
Node type	unary node
Node children	S
Translation	$\$B.finite(S^{TR})$

B.7 Unary Predicate Node

B.7.1 Negation Node

Event-B formula	$\neg P$
Rodin tag name	*NOT*
Node type	unary node
Node children	P
Translation	$\$B.not(P^{TR})$

B.8 Quantified Predicate Nodes

B.8.1 Universal Quantification Node

Event-B formula	$\forall x_1,\ldots,x_n \cdot P \quad (n \geq 1)$
Rodin tag name	*FORALL*
Node type	n-ary node
Node children	$x_1,\ldots,x_n$, P
Translation	$\$B.forAll(P^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

The special treatment of P^{TR*} is defined by a wrapped function:

$P^{TR*} \mathrel{\widehat{=}}$ `function(` $x_1^{TR},\ldots,x_n^{TR}$ `) {`
    `return` P^{TR}`;`
`}`

The algorithm for the treatment of $[x_1^{TR*},\ldots,x_n^{TR*}]$ is defined by:

1. let $array = []$
2. for $i = 1$ to n do
   (a) try to find an explicit domain constraint set using the $\in$ operator in P for x_i
   (b) if found, add the translation of that set to *array*
   (c) else add the translation of the basic type set associated by the static checker to *array*
3. return *array*

For example, given a predicate $\forall x_1,x_2 \cdot x_1 \in S \wedge x_2 \in T \Rightarrow Q$, the result of translation $[x_1^{TR*},x_2^{TR*}]$ is defined as: $[x_1^{TR*},x_2^{TR*}] \mathrel{\widehat{=}} [S^{TR},T^{TR}]$.

B.8.2 Existential Quantification Node

Event-B formula	$\exists x_1,\ldots,x_n \cdot P \quad (n \geq 1)$
Rodin tag name	*EXISTS*
Node type	n-ary node
Node children	$x_1,\ldots,x_n$, P
Translation	$\$B.exists(P^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

The special treatments of P^{TR*} and $[x_1^{TR*}, \ldots, x_n^{TR*}]$ are same to the universal quantification node in Sect. B.8.1.

B.9 Multiple Predicate Node

B.9.1 Partition of a Set Node

Event-B formula	partition$(S_1, \ldots, S_n)$ $(n \geq 1)$
Rodin tag name	*KPARTITION*
Node type	n-ary node
Node children	S_1, …, S_n
Translation	$\$B.partition(S_1^{TR}, \ldots, S_n^{TR})$

B.10 Identifier Expression Nodes

B.10.1 Free Identifier Node

Event-B formula	χ
Rodin tag name	*FREE_IDENT*
Node type	leaf node
Node children	-
Translation	1) output $\$cst.\chi$ if χ is a constant defined in a context 2) output $\$var.\chi.value$ if χ is a variable defined in a machine 3) output $\$arg.\chi.value$ if χ is a parameter defined in an event

B.10.2 Bound Identifier Node

Event-B formula	χ
Rodin tag name	*BOUND_IDENT*
Node type	leaf node
Node children	-
Translation	1) output χ if χ is not a primed identifier 2) output $\$var.\chi.\_value$ if χ is a primed identifier

The second case occurs at the non-deterministic assignment with a before-after predicate node in Sect. B.18.3.

B.10.3 Bound Identifier Declaration Node

Event-B formula	-
Rodin tag name	*BOUND_IDENT_DECL*
Node type	leaf node
Node children	-
Translation	-

This kind of node is just internal used by the JeB translator to get the name of bound identifiers in a formula, e.g., in the formula

$$\forall x_1, x_2 \cdot x_1 \in \mathbb{N} \land x_2 \in \mathbb{N} \Rightarrow x_1 + x_2 \geq 0$$

the first occurrences of x_1 and x_2 are the nodes of bound identifier declarations, the other occurrences are the nodes of bound identifiers.

B.11 Integer Literal and Set Extension Expression Nodes

B.11.1 Integer Literal Node

Event-B formula	α
Rodin tag name	*INTLIT*
Node type	leaf node
Node children	-
Translation	$\$B('\alpha')$

For example,

Event-B	Translation
1000	$\$B('1000')$
12345678901234567890123456789	$\$B('12345678901234567890123456789')$

B.11.2 Set Extension Node

Event-B formula	$\{E_1, \ldots, E_n\} \quad (n \geq 1)$
Rodin tag name	*SETEXT*
Node type	n-ary node
Node children	$E_1, \ldots, E_n$
Translation	$\$B.SetExtension(E_1^{TR}, \ldots, E_n^{TR})$

B.12 Binary Expression Nodes

B.12.1 Ordered Pair Node

Event-B formula	$E \mapsto F$
Rodin tag name	*MAPSTO*
Node type	binary node
Node children	E, F
Translation	$\$B.Pair(E^{TR}, F^{TR})$

B.12.2 Relations Node

Event-B formula	$S \leftrightarrow T$
Rodin tag name	*REL*
Node type	binary node
Node children	S, T
Translation	$\$B.Relations(S^{TR}, T^{TR})$

B.12.3 Total Relations Node

Event-B formula	$S \nleftrightarrow T$
Rodin tag name	*TREL*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalRelations(S^{TR}, T^{TR})$

B.12.4 Surjective Relations Node

Event-B formula	$S \leftrightarrow\!\!\!\!\rightarrow T$
Rodin tag name	*SREL*
Node type	binary node
Node children	S, T
Translation	$\$B.SurjectiveRelations(S^{TR}, T^{TR})$

B.12.5 Total Surjective Relations Node

Event-B formula	$S \nleftrightarrow\!\!\!\!\rightarrow T$
Rodin tag name	*STREL*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalSurjectiveRelations(S^{TR}, T^{TR})$

B.12.6 Partial Functions Node

Event-B formula	$S \nrightarrow T$
Rodin tag name	*PFUN*
Node type	binary node
Node children	S, T
Translation	$\$B.PartialFunctions(S^{TR}, T^{TR})$

B.12.7 Total Functions Node

Event-B formula	$S \rightarrow T$
Rodin tag name	*TFUN*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalFunctions(S^{TR}, T^{TR})$

B.12.8 Partial Injections Node

Event-B formula	$S \rightarrowtail T$
Rodin tag name	*PINJ*
Node type	binary node
Node children	S, T
Translation	$\$B.PartialInjections(S^{TR}, T^{TR})$

B.12.9 Total Injections Node

Event-B formula	$S \rightarrowtail T$
Rodin tag name	*TINJ*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalInjections(S^{TR}, T^{TR})$

B.12.10 Partial Surjections Node

Event-B formula	$S \twoheadrightarrow T$
Rodin tag name	*PSUR*
Node type	binary node
Node children	S, T
Translation	$\$B.PartialSurjections(S^{TR}, T^{TR})$

B.12.11 Total Surjections Node

Event-B formula	$S \twoheadrightarrow T$
Rodin tag name	*TSUR*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalSurjections(S^{TR}, T^{TR})$

B.12.12 Total Bijections Node

Event-B formula	$S \rightarrowtail\!\!\!\!\!\rightarrow T$
Rodin tag name	*TBIJ*
Node type	binary node
Node children	S, T
Translation	$\$B.TotalBijections(S^{TR}, T^{TR})$

B.12.13 Set Difference Node

Event-B formula	$S \setminus T$
Rodin tag name	*SETMINUS*
Node type	binary node
Node children	S, T
Translation	$\$B.setMinus(S^{TR}, T^{TR})$

B.12.14 Cartesian Product Node

Event-B formula	$S \times T$
Rodin tag name	*CPROD*
Node type	binary node
Node children	S, T
Translation	$\$B.CartesianProduct(S^{TR}, T^{TR})$

B.12.15 Direct Product Node

Event-B formula	$r_1 \otimes r_2$
Rodin tag name	*DPROD*
Node type	binary node
Node children	r_1, r_2
Translation	$\$B.directProduct(r_1^{TR}, r_2^{TR})$

B.12.16 Parallel Product Node

Event-B formula	$r_1 \parallel r_2$
Rodin tag name	*PPROD*
Node type	binary node
Node children	r_1, r_2
Translation	$\$B.parallelProduct(r_1^{TR}, r_2^{TR})$

B.12.17 Domain Restriction Node

Event-B formula	$S \lhd r$
Rodin tag name	*DOMRES*
Node type	binary node
Node children	S, r
Translation	$\$B.domainRestriction(S^{TR}, r^{TR})$

B.12.18 Domain Subtraction Node

Event-B formula	$S \lhd r$
Rodin tag name	*DOMSUB*
Node type	binary node
Node children	S, r
Translation	$\$B.domainSubtraction(S^{TR}, r^{TR})$

B.12.19 Range Restriction Node

Event-B formula	$r \rhd T$
Rodin tag name	*RANRES*
Node type	binary node
Node children	r, T
Translation	$\$B.rangeRestriction(r^{TR}, T^{TR})$

B.12.20 Range Subtraction Node

Event-B formula	$r \rhd T$
Rodin tag name	*RANSUB*
Node type	binary node
Node children	r, T
Translation	$\$B.rangeSubtraction(r^{TR}, T^{TR})$

B.12.21 Interval Node

Event-B formula	$m \mathinner{..} n$
Rodin tag name	*UPTO*
Node type	binary node
Node children	m, n
Translation	$\$B.UpTo(m^{TR}, n^{TR})$

B.12.22 Arithmetic Subtraction Node

Event-B formula	$m - n$
Rodin tag name	*MINUS*
Node type	binary node
Node children	m, n
Translation	$\$B.minus(m^{TR}, n^{TR})$

B.12.23 Arithmetic Quotient Node

Event-B formula	$m \div n$
Rodin tag name	*DIV*
Node type	binary node
Node children	m, n
Translation	$\$B.divide(m^{TR}, n^{TR})$

B.12.24 Arithmetic Reminder Node

Event-B formula	m mod n
Rodin tag name	*MOD*
Node type	binary node
Node children	m, n
Translation	$\$B.mod(m^{TR}, n^{TR})$

B.12.25 Arithmetic Exponentiation Node

Event-B formula	$m \mathbin{\widehat{}} n$
Rodin tag name	*EXPN*
Node type	binary node
Node children	m, n
Translation	$\$B.pow(m^{TR}, n^{TR})$

B.12.26 Function Image Node

Event-B formula	$f(E)$
Rodin tag name	*FUNIMAGE*
Node type	binary node
Node children	f, E
Translation	$\$B.functionImage(f^{TR}, E^{TR})$

B.12.27 Relation Image Node

Event-B formula	$r[S]$
Rodin tag name	*RELIMAGE*
Node type	binary node
Node children	r, S
Translation	$\$B.relationImage(r^{TR}, S^{TR})$

B.13 Associative Expression Nodes

B.13.1 Set Union Node

Event-B formula	$S_1 \cup \ldots \cup S_n \quad (n \geq 2)$
Rodin tag name	*BUNION*
Node type	n-ary node
Node children	S_1, ..., S_n
Translation	$\$B.setUnion(S_1^{TR}, \ldots, S_n^{TR})$

B.13.2 Set Intersection Node

Event-B formula	$S_1 \cap \ldots \cap S_n \quad (n \geq 2)$
Rodin tag name	*BINTER*
Node type	n-ary node
Node children	S_1, ..., S_n
Translation	$\$B.setInter(S_1^{TR}, \ldots, S_n^{TR})$

B.13.3 Relational Forward Composition Node

Event-B formula	$r_1 ; \ldots ; r_n \quad (n \geq 2)$
Rodin tag name	*FCOMP*
Node type	n-ary node
Node children	r_1, ..., r_n
Translation	$\$B.forwardComposition(r_1^{TR}, \ldots, r_n^{TR})$

B.13.4 Relational Backward Composition Node

Event-B formula	$r_1 \circ \ldots \circ r_n \quad (n \geq 2)$
Rodin tag name	*BCOMP*
Node type	n-ary node
Node children	r_1, ..., r_n
Translation	$\$B.backwardComposition(r_1^{TR}, \ldots, r_n^{TR})$

B.13.5 Relational Overriding Node

Event-B formula	$r_1 \lhd \ldots \lhd r_n \quad (n \geq 2)$
Rodin tag name	*OVR*
Node type	n-ary node
Node children	r_1, ..., r_n
Translation	$\$B.override(r_1^{TR}, \ldots, r_n^{TR})$

B.13.6 Arithmetic Addition Node

Event-B formula	$m_1 + \ldots + m_n \quad (n \geq 2)$
Rodin tag name	*PLUS*
Node type	n-ary node
Node children	m_1, ..., m_n
Translation	$\$B.plus(m_1^{TR}, \ldots, m_n^{TR})$

B.13.7 Arithmetic Multiplication Node

Event-B formula	$m_1 * \ldots * m_n \quad (n \geq 2)$
Rodin tag name	*MUL*
Node type	n-ary node
Node children	m_1, ..., m_n
Translation	$\$B.multiply(m_1^{TR}, \ldots, m_n^{TR})$

B.14 Atomic Expression Nodes

B.14.1 Integer Numbers Node

Event-B formula	$\mathbb{Z}$
Rodin tag name	*INTEGER*
Node type	leaf node
Node children	-
Translation	$\$B.INTEGER$

B.14.2 Natural Numbers Node

Event-B formula	$\mathbb{N}$
Rodin tag name	*NATURAL*
Node type	leaf node
Node children	-
Translation	$\$B.NATURAL$

B.14.3 Positive Natural Numbers Node

Event-B formula	$\mathbb{N}_1$
Rodin tag name	*NATURAL1*
Node type	leaf node
Node children	-
Translation	$*B.NATURAL1*

B.14.4 Boolean Values Set Node

Event-B formula	BOOL
Rodin tag name	*BOOL*
Node type	leaf node
Node children	-
Translation	$*B.BOOL*

B.14.5 Boolean TRUE Node

Event-B formula	TRUE
Rodin tag name	*TRUE*
Node type	leaf node
Node children	-
Translation	$*B.TRUE*

B.14.6 Boolean FALSE Node

Event-B formula	FALSE
Rodin tag name	*FALSE*
Node type	leaf node
Node children	-
Translation	$*B.FALSE*

B.14.7 Empty Set Node

Event-B formula	$\varnothing$
Rodin tag name	*EMPTYSET*
Node type	leaf node
Node children	-
Translation	$*B.EmptySet*

B.14.8 Arithmetic Predecessor Node

Event-B formula	$\mathrm{pred}(m)$
Rodin tag name	*KPRED*
Node type	unary node
Node children	m
Translation	$\$B.pred(m^{TR})$

B.14.9 Arithmetic Successor Node

Event-B formula	$\mathrm{succ}(m)$
Rodin tag name	*KSUCC*
Node type	unary node
Node children	m
Translation	$\$B.succ(m^{TR})$

B.14.10 First Projection Node

Event-B formula	prj_1
Rodin tag name	*KPRJ1_GEN*
Node type	leaf node
Node children	-
Translation	$\$B.prj1$

B.14.11 Second Projection Node

Event-B formula	prj_2
Rodin tag name	*KPRJ2_GEN*
Node type	leaf node
Node children	-
Translation	$\$B.prj2$

B.14.12 Identity Node

Event-B formula	id
Rodin tag name	*KID_GEN*
Node type	leaf node
Node children	-
Translation	$\$B.id$

B.15 Bool Expression Node

B.15.1 Bool Node

Event-B formula	$\text{bool}(P)$
Rodin tag name	*KBOOL*
Node type	unary node
Node children	P
Translation	$\$B.bool(P^{TR})$

B.16 Unary Expression Nodes

B.16.1 Cardinality Node

Event-B formula	$\text{card}(S)$
Rodin tag name	*KCARD*
Node type	unary node
Node children	S
Translation	$\$B.card(S^{TR})$

B.16.2 Power Set Node

Event-B formula	$\mathbb{P}(S)$
Rodin tag name	*POW*
Node type	unary node
Node children	S
Translation	$\$B.PowerSet(S^{TR})$

B.16.3 Non-empty Subsets Node

Event-B formula	$\mathbb{P}_1(S)$
Rodin tag name	*POW1*
Node type	unary node
Node children	S
Translation	$\$B.PowerSet1(S^{TR})$

B.16.4 Generalized Union Node

Event-B formula	$\text{union}(U)$
Rodin tag name	*KUNION*
Node type	unary node
Node children	U
Translation	$\$B.union(U^{TR})$

B.16.5 Generalized Intersection Node

Event-B formula	$\mathrm{inter}(U)$
Rodin tag name	*KINTER*
Node type	unary node
Node children	U
Translation	$\$B.inter(U^{TR})$

B.16.6 Domain Node

Event-B formula	$\mathrm{dom}(r)$
Rodin tag name	*KDOM*
Node type	unary node
Node children	r
Translation	$\$B.dom(r^{TR})$

B.16.7 Range Node

Event-B formula	$\mathrm{ran}(r)$
Rodin tag name	*KRAN*
Node type	unary node
Node children	r
Translation	$\$B.ran(r^{TR})$

B.16.8 Arithmetic Minimum Node

Event-B formula	$\min(S)$
Rodin tag name	*KMIN*
Node type	unary node
Node children	S
Translation	$\$B.min(S^{TR})$

B.16.9 Arithmetic Maximum Node

Event-B formula	$\max(S)$
Rodin tag name	*KMAX*
Node type	unary node
Node children	S
Translation	$\$B.max(S^{TR})$

B.16.10 Inverse Relation Node

Event-B formula	$r^{\sim}$
Rodin tag name	*CONVERSE*
Node type	unary node
Node children	r
Translation	$\$B.converse(r^{TR})$

B.16.11 Arithmetic Unary Minus Node

Event-B formula	$-m$
Rodin tag name	*UNMINUS*
Node type	unary node
Node children	m
Translation	$\$B.unminus(m^{TR})$

B.17 Quantified Expression Nodes

B.17.1 Set Comprehension Node

Event-B formula	$\{x_1,\ldots,x_n \cdot P \mid E\} \quad (n \geq 1)$
	$\{E \mid P\}$ (short form)
Rodin tag name	*CSET*
Node children	$x_1,\ldots,x_n,P,E$
Translation	$\$B.SetComprehension(P^{TR*},E^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

The special treatments of P^{TR*} and $[x_1^{TR*},\ldots,x_n^{TR*}]$ are same to the universal quantification node in Sect. B.8.1. The treatment of E^{TR*} is defined by:

$E^{TR*} \mathrel{\widehat{=}}$ `function(` $x_1^{TR},\ldots,x_n^{TR}$ `) {`
    `return` E^{TR}`;`
`}`

B.17.2 Lambda Node

Event-B formula	$\lambda L \cdot P \mid E$
	$\lambda E \mid P$ (short form)
Rodin tag name	*CSET*
Node children	$x_1,\ldots,x_n,P,E$
	where $x_1,\ldots,x_n$ $(n \geq 1)$ are the identifiers in L
Translation	$\$B.Lambda(P^{TR*},E^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

In Rodin, the lambda expression is special form of the set comprehension, it uses the same tag with the set comprehension node. The treatments of P^{TR*} and $[x_1^{TR*},\ldots,x_n^{TR*}]$

are same to the universal quantification node in Sect. B.8.1. The treatments of E^{TR*} is same to the set comprehension node in Sect. B.17.1.

B.17.3 Quantified Union Node

Event-B formula	$\bigcup x_1,\ldots,x_n \cdot P \mid E \quad (n \geq 1)$ $\bigcup E \mid P$ (short form)
Rodin tag name	*QUNION*
Node children	$x_1,\ldots,x_n,P,E$
Translation	$\$B.quantifiedUnion(P^{TR*},E^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

The treatments of P^{TR*} and $[x_1^{TR*},\ldots,x_n^{TR*}]$ are same to the universal quantification node in Sect. B.8.1. The treatments of E^{TR*} is same to the set comprehension node in Sect. B.17.1.

B.17.4 Quantified Intersection Node

Event-B formula	$\bigcap x_1,\ldots,x_n \cdot P \mid E \quad (n \geq 1)$ $\bigcap E \mid P$ (short form)
Rodin tag name	*QINTER*
Node children	$x_1,\ldots,x_n,P,E$
Translation	$\$B.quantifiedInter(P^{TR*},E^{TR*},[x_1^{TR*},\ldots,x_n^{TR*}])$

The treatments of P^{TR*} and $[x_1^{TR*},\ldots,x_n^{TR*}]$ are same to the universal quantification node in Sect. B.8.1. The treatments of E^{TR*} is same to the set comprehension node in Sect. B.17.1.

B.18 Assignment Nodes

B.18.1 Deterministic Assignment Node

Event-B formula	$x_1,\ldots,x_n := E_1,\ldots,E_n \quad (n \geq 1)$
Rodin tag name	*BECOMES_EQUAL_TO*
Node type	n-ary node
Node children	$x_1,\ldots,x_n,E_1,\ldots,E_n$
Translation	$\$B.becomesEqualTo([x_1^{TR},\ldots,x_n^{TR}],[E_1^{TR},\ldots,E_n^{TR}])$

There exists a special assignment which uses a relational overriding: $x(E) := F$. In fact, this is just a shorthand for $x := x \leftarrow\!\!+ \{E \mapsto F\}$, therefore this form is unified to the above node.

B.18.2 Non-deterministic Assignment of a Set Member Node

Event-B formula	$x :\in E$
Rodin tag name	*BECOMES_MEMBER_OF*
Node type	binary node
Node children	x, E
Translation	$\$B.becomesMemberOf(x^{TR}, E^{TR})$

B.18.3 Non-deterministic Assignment with a Before-after-predicate Node

Event-B formula	$x_1, \ldots, x_n :\mid Q(x'_1, \ldots, x'_n) \quad (n \geq 1)$
Rodin tag name	*BECOMES_SUCH_THAT*
Node type	n-ary node
Node children	$x_1, \ldots, x_n, x'_1, \ldots, x'_n, Q$
Translation	$\$B.becomesSuchThat([x_1^{TR}, \ldots, x_n^{TR}], Q^{TR*}, [x'_1{}^{TR*}, \ldots, x'_n{}^{TR*}])$

In Rodin, Q is a before-after predicate quantified by the bounded identifiers $x'_1, \ldots, x'_n$ that reference the primed value of identifiers $x_1, \ldots, x_n$, respectively. The special treatment of Q^{TR*} is defined by a wrapped function:

```
Q^TR* ≙ function() {
    return Q^TR;
}
```

The special treatment of $[x'_1{}^{TR*}, \ldots, x'_n{}^{TR*}]$ is same to the universal quantification node in Sect. B.8.1.

Appendix C

JavaScript Library for Event-B

Contents

We use the following pattern to represent each API:

Notation	the corresponding Event-B mathematical notation
Syntax	the API syntax in JavaScript
Parameter	parameters, if present
Return	the returned value, if present
Description	the role
Example	some examples

The symbols and notations that we uses are described below:

Symbols	**Meaning**
`$B`	the namespace of the JavaScript library for Event-B
`$B()`	an alias of the integer constructor `$B.Integer()`
`.` (dot)	uses to access the properties and methods of a JavaScript object
`[...]`	1) denotes a JavaScript array
	2) denotes optional parameters in the syntax grammar
`'...'`	denotes a JavaScript string
`true, false`	denotes the JavaScript primitive boolean values
`undefined`	denotes the JavaScript primitive `undefined` value

C.1 Booleans

C.1.1 $B.TRUE

Notation	boolean TRUE
Syntax	`$B.TRUE`
Description	This constant has the JavaScript primitive boolean `true` value.
Example	`$B.TRUE` returns `true`

C.1.2 $B.FALSE

Notation	boolean FALSE
Syntax	`$B.FALSE`
Description	This constant has the JavaScript primitive boolean `false` value.
Example	`$B.FALSE` returns `false`

C.1.3 $B.bool

Notation	bool expression
Syntax	`$B.bool( P )`
Parameter	`P` - a predicate
Return	1) `true` if the evaluation of `P` is `true` 2) `false` if the evaluation of `P` is `false`
Description	This function converts a predicate to a boolean value.
Example	`$B.bool($B.bTrue())` returns `true` `$B.bool($B.bFalse())` returns `false`

C.2 Arithmetic

C.2.1 $B.Integer

Notation	integer literal
Syntax	`$B.Integer( value )`
Parameter	`value` - a decimal string enclosed in single quotes, it consists of an optional minus sign followed by a sequence of one or more decimal digits, i.e., it matches the regular expression `/^[-]?[0-9]+$/`
Return	an instance of `$B.Integer`
Description	This function is the integer object constructor which represents an arbitrary precision integer. `$B.Integer` objects should be considered immutable.
Example	`$B.Integer('-1000')` `$B.Integer('12345678901234567890123456789O')`

Remark: The namespace `$B` also references to `$B.Integer`, we can use `$B` as the integer object constructor, the above examples are equal to `$B('-1000')`, `$B('12345678901234567890123456789O')`, respectively.

To achieve a context-free lexical analysis, Event-B considers that integer literals are unsigned. For instance, the string -1 is thought as an unary minus operator followed by a number. To facilitate user input, we design that integer literals are signed in the library implementation.

C.2.2 $B.minus

Notation	arithmetic subtraction
Syntax	`$B.minus( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if both parameters are instances of `$B.Integer` 2) `undefined` if any parameter is not an instance of `$B.Integer`
Description	This function computes the subtraction of two input integers.
Example	`$B.minus($B('8'), $B('5'))` returns `$B('3')` `$B.minus($B('8'), undefined)` returns `undefined`

C.2.3 $B.divide

Notation	arithmetic quotient
Syntax	`$B.divide( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if both parameters are instances of `$B.Integer` and `n` is not equal to zero 2) `undefined` if any parameter is not an instance of `$B.Integer` or `n` is equal to zero
Description	This function computes the quotient of two input integers.

Example	`$B.divide($B('8'), $B('5'))` returns `$B('1')`
	`$B.divide($B('8'), $B('0'))` returns `undefined`
	`$B.divide($B('8'), undefined)` returns `undefined`

C.2.4 $B.mod

Notation	arithmetic reminder
Syntax	`$B.mod( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if both parameters are instances of `$B.Integer` and `m >=0` $\wedge$ `n > 0` 2) `undefined` if any parameter is not an instance of `$B.Integer` or `m < 0` or `n <= 0`
Description	This function computes the reminder of two input integers.

Example	`$B.mod($B('8'), $B('5'))` returns `$B('3')`
	`$B.mod($B('8'), $B('0'))` returns `undefined`
	`$B.mod($B('8'), undefined)` returns `undefined`

C.2.5 $B.pow

Notation	arithmetic exponentiation
Syntax	`$B.pow( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if both parameters are instances of `$B.Integer` and `m >=0` $\wedge$ `n >= 0` 2) `undefined` if any parameter is not an instance of `$B.Integer` or `m < 0` or `n < 0`
Description	This function computes the exponentiation of two input integers.
Example	`$B.pow($B('8'), $B('5'))` returns `$B('32768')` `$B.pow($B('8'), undefined)` returns `undefined`

C.2.6 $B.plus

Notation	arithmetic addition
Syntax	`$B.plus( m1, m2[, ...[, mN]] )`
Parameter	`m1, m2, ..., mN` - instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if all parameters are instances of `$B.Integer` 2) `undefined` if any parameter is not an instance of `$B.Integer`
Description	This function computes the addition of two or more input integers.
Example	`$B.plus($B('1'), $B('2'), $B('5'))` returns `$B('8')` `$B.plus($B('1'), undefined)` returns `undefined`

C.2.7 $B.multiply

Notation	arithmetic multiplication
Syntax	`$B.multiply( m1, m2[, ...[, mN]] )`
Parameter	`m1, m2, ..., mN` - instances of `$B.Integer`
Return	1) an instance of `$B.Integer` if all parameters are instances of `$B.Integer` 2) `undefined` if any parameter is not an instance of `$B.Integer`
Description	This function computes the multiplication of two or more input integers.
Example	`$B.multiply($B('1'), $B('2'), $B('5'))` returns `$B('10')` `$B.multiply($B('1'), undefined)` returns `undefined`

C.2.8 $B.pred

Notation	arithmetic predecessor
Syntax	`$B.pred( m )`
Parameter	`m` - an instance of `$B.Integer`
Return	1) an instance of `$B.Integer` if `m` is an instance of `$B.Integer` 2) `undefined` if `m` is not an instance of `$B.Integer`
Description	This function computes the predecessor of an input integer.
Example	`$B.pred($B('100'))` returns `$B('99')` `$B.pred(undefined)` returns `undefined`

C.2.9 $B.succ

Notation	arithmetic successor
Syntax	`$B.succ( m )`
Parameter	`m` - an instance of `$B.Integer`
Return	1) an instance of `$B.Integer` if `m` is an instance of `$B.Integer` 2) `undefined` if `m` is not an instance of `$B.Integer`
Description	This function computes the successor of an input integer.
Example	`$B.succ($B('100'))` returns `$B('101')` `$B.succ(undefined)` returns `undefined`

C.2.10 $B.unminus

Notation	arithmetic unary minus
Syntax	`$B.unminus( m )`
Parameter	`m` - an instance of `$B.Integer`
Return	1) an instance of `$B.Integer` if `m` is an instance of `$B.Integer` 2) `undefined` if `m` is not an instance of `$B.Integer`
Description	This function computes the negation of an input integer.
Example	`$B.unminus($B('100'))` returns `$B('-100')` `$B.unminus(undefined)` returns `undefined`

C.2.11 $B.lessThan

Notation	arithmetic less predicate
Syntax	`$B.lessThan( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	`true` if both parameters are instances of `$B.Integer` and `m` is less than `n`, otherwise `false`
Description	This function compares two input integers.

Example	`$B.lessThan($B('5'), $B('8'))` returns `true`
	`$B.lessThan($B('5'), $B('5'))` returns `false`
	`$B.lessThan($B('5'), undefined)` returns `false`

C.2.12 $B.lessEqual

Notation	arithmetic less or equal predicate
Syntax	`$B.lessEqual( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	`true` if both parameters are instances of `$B.Integer` and `m` is less than or equal to `n`, otherwise `false`
Description	This function compares two input integers.

Example	`$B.lessEqual($B('5'), $B('8'))` returns `true`
	`$B.lessEqual($B('5'), $B('5'))` returns `true`
	`$B.lessEqual($B('5'), undefined)` returns `false`

C.2.13 $B.greaterThan

Notation	arithmetic greater predicate
Syntax	`$B.greaterThan( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	`true` if both parameters are instances of `$B.Integer` and `m` is greater than `n`, otherwise `false`
Description	This function compares two input integers.

Example	`$B.greaterThan($B('8'), $B('5'))` returns `true`
	`$B.greaterThan($B('5'), $B('5'))` returns `false`
	`$B.greaterThan($B('5'), undefined)` returns `false`

C.2.14 $B.greaterEqual

Notation	arithmetic greater or equal predicate
Syntax	`$B.greaterEqual( m, n )`
Parameter	`m, n` - two instances of `$B.Integer`
Return	`true` if both parameters are instances of `$B.Integer` and `m` is greater than or equal to `n`, otherwise `false`
Description	This function compares two input integers.

Example	`$B.greaterEqual($B('8'), $B('5'))` returns `true`
	`$B.greaterEqual($B('5'), $B('5'))` returns `true`
	`$B.greaterEqual($B('5'), undefined)` returns `false`

C.3 Predicates

C.3.1 $B.bTrue

Notation	primitive true predicate
Syntax	`$B.bTrue()`
Parameter	-
Return	`true`
Description	This function represents the primitive true predicate.
Example	`$B.bTrue()` returns `true`

C.3.2 $B.bFalse

Notation	primitive false predicate
Syntax	`$B.bFalse()`
Parameter	-
Return	`false`
Description	This function represents the primitive false predicate.
Example	`$B.bFalse()` returns `false`

C.3.3 $B.implication

Notation	implication predicate
Syntax	`$B.implication( P, Q )`
Parameter	`P, Q` - two predicates
Return	if the evaluation of `P` is `true`, returns the evaluation of `Q`, if the evaluation of `P` is `false`, returns `true`
Description	This function evaluates two input predicates.

Example	`$B.implication($B.bTrue(), $B.bTrue())` returns `true`
	`$B.implication($B.bTrue(), $B.bFalse())` returns `false`
	`$B.implication($B.bFalse(), $B.bTrue())` returns `true`
	`$B.implication($B.bFalse(), $B.bFalse())` returns `true`

C.3.4 $B.equivalence

Notation	equivalence predicate
Syntax	`$B.equivalence( P, Q )`
Parameter	`P, Q` - two predicates
Return	1) `true` if the evaluation of `P` is equal to the evaluation of `Q` 2) `false` if the evaluation of `P` is not equal to the evaluation of `Q`
Description	This function evaluates two input predicates.

Example	`$B.equivalence($B.bTrue(), $B.bTrue())` returns `true`
	`$B.equivalence($B.bTrue(), $B.bFalse())` returns `false`
	`$B.equivalence($B.bFalse(), $B.bTrue())` returns `false`
	`$B.equivalence($B.bFalse(), $B.bFalse())` returns `true`

C.3.5 $B.and

Notation	conjunction predicate
Syntax	`$B.and( P1, P2[, ...[, Pn]] )`
Parameter	`P1, P2, ..., Pn` - predicates
Return	if the evaluation (from left to right) of any predicate is `false`, immediately returns `false`, otherwise returns the evaluation of `Pn`
Description	This function evaluates two or more input predicates.

Example	`$B.and($B.bTrue(), $B.bTrue())` returns `true`
	`$B.and($B.bTrue(), $B.bFalse())` returns `false`
	`$B.and($B.bFalse(), $B.bTrue())` returns `false`
	`$B.and($B.bFalse(), $B.bFalse())` returns `false`

C.3.6 $B.or

Notation	disjunction predicate
Syntax	`$B.or( P1, P2[, ...[, Pn]] )`
Parameter	`P1, P2, ..., Pn` - predicates
Return	if the evaluation (from left to right) of any predicate is `true`, immediately returns `true`, otherwise returns the evaluation of `Pn`
Description	This function evaluates two or more input predicates.

Example	`$B.or($B.bTrue(), $B.bTrue())` returns `true`
	`$B.or($B.bTrue(), $B.bFalse())` returns `true`
	`$B.or($B.bFalse(), $B.bTrue())` returns `true`
	`$B.or($B.bFalse(), $B.bFalse())` returns `false`

C.3.7 $B.not

Notation	negation predicate
Syntax	`$B.not( P )`
Parameter	`P` - a predicate
Return	1) `true` if the evaluation of `P` is `false` 2) `false` if the evaluation of `P` is `true`
Description	This function evaluates an input predicate.
Example	`$B.not($B.bTrue())` returns `false` `$B.not($B.bFalse())` returns `true`

C.3.8 $B.forAll

Notation	universal quantified predicate
Syntax	`$B.forAll( predicateFunction, domainArray )`
Parameter	`predicateFunction` - a function with the quantifiers as its parameters and the quantified predicate as its function body `domainArray` - an array, it contains the enumerated sets for each parameter in the function `predicateFunction`
Return	for each value of each set in `domainArray`, constructs a new array `argArray`, evaluates `predicateFunction` with `argArray` as its arguments until it returns `false`, otherwise finally returns `true`
Description	This function evaluates the universal quantification.
Example	sees below

Evaluating the universal quantification $\forall x, y \cdot x \in 1..2 \wedge y \in 3..4 \Rightarrow x + y >= 4$ yields `true`, where

```
predicateFunction ≙ function( x, y ) {
    return $B.implication( $B.and(
        $B.belong(x, $B.UpTo($B('1'), $B('2'))),
        $B.belong(y, $B.UpTo($B('3'), $B('4'))) ),
        $B.greaterEqual($B.plus(x, y), $B('4')) );
}

domainArray ≙ [$B.UpTo($B('1'), $B('2')), $B.UpTo($B('3'), $B('4'))]
```

Remark: If there exists an infinite set for any quantifiers, users should consider the range of enumerated values or their own implementation, by default, the enumerated values for $\mathbb{Z}, \mathbb{N}, \mathbb{N}_1$ are constrained by two configurable simulation parameters:

```
$B.MAX_ENUMERATED_VALUE
$B.MIN_ENUMERATED_VALUE
```

C.3.9 $B.exists

Notation	existential quantified predicate
Syntax	`$B.exists( predicateFunction, domainArray )`
Parameter	`predicateFunction` - a function with the quantifiers as its parameters and the quantified predicate as its function body `domainArray` - an array, it contains the enumerated sets for each parameter in the function `predicateFunction`
Return	for each value of each set in `domainArray`, constructs a new array `argArray`, evaluates `predicateFunction` with `argArray` as its arguments until it returns `true`, otherwise finally returns `false`
Description	This function evaluates the existential quantification.
Example	sees below

Evaluating the existential quantification $\exists x, y \cdot x \in 1..2 \land y \in 3..4 \Rightarrow x + y >= 5$ yields `true`, where

```
predicateFunction ≙ function( x, y ) {
    return $B.implication( $B.and(
        $B.belong(x, $B.UpTo($B('1'), $B('2'))),
        $B.belong(y, $B.UpTo($B('3'), $B('4'))) ),
        $B.greaterEqual($B.plus(x, y), $B('5')) );
}

domainArray ≙ [$B.UpTo($B('1'), $B('2')), $B.UpTo($B('3'), $B('4'))]
```

Enumerating an infinite set in `domainArray` sees the remark in Sect. C.3.8.

C.3.10 $B.equal

Notation	equality predicate
Syntax	`$B.equal( E, F )`
Parameter	`E, F` - two expressions, they should have the same type but can be any Event-B mathematical objects
Return	1) `true` if the evaluation of `E` is equal to the evaluation of `F` 2) `false` if the evaluation of `E` is not equal to the evaluation of `F`
Description	This function evaluates the equality of two input expressions.

Example	`$B.equal($B('1'), $B('1'))` returns `true`
	`$B.equal($B('1'), undefined)` returns `false`
	`$B.equal($B.Pair($B('1'), $B('2')), $B.Pair($B('1'), $B('2')))` returns `true`
	`$B.equal($B.UpTo($B('1'), $B('2')), $B.UpTo($B('1'), $B('2')))` returns `true`

C.3.11 $B.notEqual

Notation	inequality predicate
Syntax	`$B.notEqual( E, F )`
Parameter	`E, F` - two expressions, they should have the same type but can be any Event-B mathematical objects
Return	`!$B.equal( E, F )`
Description	This function evaluates the inequality of two input expressions.
Example	`$B.notEqual($B('1'), $B('2'))` returns `true` `$B.notEqual($B('1'), undefined)` returns `true`

C.4 Sets

C.4.1 $B.SetExtension

Notation	set extension
Syntax	`$B.SetExtension( [E1 [, ...[, En]]] )`
Parameter	`E1, ..., En` - the same type expressions
Return	an instance of `$B.SetExtension` object which contains elements `E1, ..., En`
Description	This function is the set object constructor with the extension form. Calling this function without parameters will return the empty set.

Example	`$B.SetExtension($B.TRUE, $B.FALSE)`
	`$B.SetExtension($B('1'), $B('2'), $B('5'))`
	`$B.SetExtension($B.Pair($B('1'), $B('2')),`
	`$B.Pair($B('3'), $B('5')))`

C.4.2 $B.EmptySet

Notation	empty set
Syntax	`$B.EmptySet`
Description	This represents the empty set. Indeed, it can be obtained by calling the function `$B.SetExtension` without any parameters.
Example	`$B.belong($B('1'), $B.EmptySet)` returns `false`

C.4.3 $B.BOOL

Notation	boolean values set
Syntax	`$B.BOOL`
Description	This represents the boolean values set that can be obtained by calling the function `$B.SetExtension` with arguments `$B.TRUE, $B.FALSE`.
Example	`$B.belong($B.TRUE, $B.BOOL)` returns `true`

C.4.4 $B.UpTo

Notation	arithmetic interval
Syntax	`$B.UpTo( low, high )`
Parameter	`low, high` - two instances of `$B.Integer`
Return	an instance of `$B.UpTo` object
Description	This function is the arithmetic interval object constructor. The instance of `$B.UpTo` has two properties `low` and `high`.
Example	`$B.UpTo($B('1'), $B('3'))` represents the set $\{1,2,3\}$ `$B.UpTo($B('3'), $B('1'))` is equal to the empty set

C.4.5 $B.INTEGER

Notation	integer numbers
Syntax	`$B.INTEGER`
Description	This object represents the set of integer numbers.
Example	`$B.belong($B('1'), $B.INTEGER)` returns `true` `$B.belong($B('-1'), $B.INTEGER)` returns `true`

C.4.6 $B.NATURAL

Notation	natural numbers
Syntax	`$B.NATURAL`
Description	This object represents the set of natural numbers.
Example	`$B.belong($B('0'), $B.NATURAL)` returns `true` `$B.belong($B('-1'), $B.NATURAL)` returns `false`

C.4.7 $B.NATURAL1

Notation	positive natural numbers
Syntax	`$B.NATURAL1`
Description	This object represents the set of positive natural numbers.
Example	`$B.belong($B('0'), $B.NATURAL1)` returns `false` `$B.belong($B('1'), $B.NATURAL1)` returns `true`

C.4.8 $B.SetComprehension

Notation	set comprehension
Syntax	`$B.SetComprehension( predicateFunction, expressionFunction, domainArray )`
Parameter	`predicateFunction` - a function with the quantifiers as its parameters and the quantified predicate as its function body `expressionFunction` - a function with the quantifiers as its parameters and the quantified expression as its function body `domainArray` - an array, it contains the enumerated sets for each quantifier
Return	an instance of `$B.SetComprehension` object
Description	This function is the set object constructor with the comprehension form. The instance of `$B.SetComprehension` first exists in an abstract form, it should be computed and transformed as a concrete set extension on-demand, e.g., in the set union operation.
Example	sees below

Evaluating the set comprehension $\{x, y \cdot x \in 1..2 \wedge y \in 3..4 \mid x+y\}$ yields the set $\{4,5,6\}$, where

```
predicateFunction ≙ function( x, y ) {
    return $B.and( $B.belong(x, $B.UpTo($B('1'), $B('2'))),
                   $B.belong(y, $B.UpTo($B('3'), $B('4'))) );
}

expressionFunction ≙ function( x, y ) {
    return $B.plus(x, y);
}

domainArray ≙ [$B.UpTo($B('1'), $B('2')), $B.UpTo($B('3'), $B('4'))]
```

Enumerating an infinite set in `domainArray` sees the remark in Sect. C.3.8.

C.4.9 $B.PowerSet

Notation	power set
Syntax	`$B.PowerSet( S )`
Parameter	`S` - a set
Return	an instance of `$B.PowerSet` object
Description	This function is designed as a constructor. The instance of `$B.PowerSet` first exists in an abstract form, it should be computed and transformed as a concrete set extension on-demand, e.g., in the set union operation.
Example	`$B.PowerSet($B.BOOL)` `$B.PowerSet($B.UpTo($B('1'), $B('2')))`

C.4.10 $B.PowerSet1

Notation	non-empty subsets
Syntax	`$B.PowerSet1( S )`
Parameter	`S` - a set
Return	an instance of `$B.PowerSet1` object
Description	This function is designed as a constructor. The instance of `$B.PowerSet1` first exists in an abstract form, it should be computed and transformed as a concrete set extension on-demand, e.g., in the set union operation.
Example	`$B.PowerSet1($B.BOOL)` `$B.PowerSet1($B.UpTo($B('1'), $B('2')))`

C.4.11 $B.setMinus

Notation	set difference
Syntax	`$B.setMinus( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.SetExtension` object
Description	This function computes the difference of two input sets. The result is defined as $\{x \mid x \in S \wedge x \notin T\}$. The computation should consider different forms of input sets.

Example	`$B.setMinus($B.SetExtension($B('1'), $B('2')),`
	`$B.SetExtension($B('1')))`
	returns `$B.SetExtension($B('2'))`

C.4.12 $B.setUnion

Notation	set union
Syntax	`$B.setUnion( S1, S2[, ...[, Sn]] )`
Parameter	`S1, S2, ..., Sn` - sets
Return	an instance of `$B.SetExtension` object
Description	This function computes the union of two or more input sets. The result is defined as $\{x \mid x \in S1 \vee \ldots \vee x \in Sn\}$. The computation should consider different forms of input sets.

Example	`$B.setUnion( $B.SetExtension($B('1'), $B('2')),`
	`$B.SetExtension($B('5')) )`
	returns `$B.SetExtension($B('1'), $B('2'), $B('5'))`

C.4.13 $B.setInter

Notation	set intersection
Syntax	`$B.setInter( S1, S2[, ...[, Sn]] )`
Parameter	`S1, S2, ..., Sn` - sets
Return	an instance of `$B.SetExtension` object
Description	This function computes the intersection of two or more input sets. The result is defined as $\{x \mid x \in S1 \vee \ldots \vee x \in Sn\}$. The computation should consider different forms of input sets.
Example	`$B.setInter( $B.SetExtension($B('1'), $B('2')), $B.SetExtension($B('2')) )` returns `$B.SetExtension($B('2'))`

C.4.14 $B.union

Notation	generalized union
Syntax	`$B.union( U )`
Parameter	`U` - a set of sets
Return	an instance of `$B.SetExtension` object
Description	This function computes the union of all elements of a set. The result is defined as $\{x \mid \exists s \cdot s \in U \wedge x \in s\}$.
Example	`$B.union($B.SetExtension( $B.SetExtension($B('1'), $B('2')), $B.SetExtension($B('5')) ))` returns `$B.SetExtension( $B('1'), $B('2'), $B('5') )`

C.4.15 $B.inter

Notation	generalized intersection
Syntax	`$B.inter( U )`
Parameter	`U` - a set of sets
Return	an instance of `$B.SetExtension` object
Description	This function computes the intersection of all elements of a set. The result is defined as $\{x \mid \forall s \cdot s \in U \Rightarrow x \in s\}$.
Example	`$B.inter($B.SetExtension( $B.SetExtension($B('1'), $B('2')), $B.SetExtension($B('1')) ))` returns `$B.SetExtension($B('1'))`

C.4.16 $B.quantifiedUnion

Notation	quantified union
Syntax	`$B.quantifiedUnion( predicateFunction, expressionFunction, domainArray )`
Parameter	The parameters definition is same to the set comprehension interface in Sect. C.4.8.
Return	an instance of `$B.SetExtension` object
Description	This function computes the quantified union. The result is defined as $\bigcup x_1,\ldots,x_n \cdot P \mid E = union(\{x_1,\ldots,x_n \cdot P \mid E\})$.
Example	sees below

Evaluating the quantified union $\bigcup x \cdot x \in 2\,..\,4 \mid \{5 * z \mid z \in 1\,..\,4 \wedge z \leq x\}$ yields the set $\{5, 10, 15, 20\}$, where

```
predicateFunction ≙ function( x ) {
    return $B.belong(x, $B.SetExtension($B('2'), $B('4')));
}

expressionFunction ≙ function( x ) {
    return $B.SetComprehension(
        function( z ) {
            return $B.and( $B.belong(z, $B.UpTo($B('1'), $B('4'))),
                           $B.lessEqual(z, x) );
        },
        function( z ) {
            return $B.multiply($B('5'), z);
        },
        [$B.UpTo($B('1'), $B('4'))]
    );
}

domainArray ≙ [$B.UpTo($B('2'), $B('4'))]
```

Enumerating an infinite set in `domainArray` sees the remark in Sect. C.3.8.

C.4.17 $B.quantifiedInter

Notation	quantified intersection
Syntax	`$B.quantifiedInter( predicateFunction, expressionFunction, domainArray )`
Parameter	The parameters definition is same to the set comprehension interface in Sect. C.4.8.
Return	an instance of `$B.SetExtension` object
Description	This function computes the quantified intersection. The result is defined as $\bigcap x_1,\ldots,x_n \cdot P \mid E = inter(\{x_1,\ldots,x_n \cdot P \mid E\})$.
Example	sees below

Evaluating the quantified intersection $\bigcap x \cdot x \in 2..4 \mid \{5 * z \mid z \in 1..4 \wedge z \leq x\}$ yields the set $\{5,10\}$, where the definitions of `predicateFunction`, `expressionFunction` and `domainArray` are same to the example in Sect. C.4.16.

Enumerating an infinite set in `domainArray` sees the remark in Sect. C.3.8.

C.4.18 $B.card

Notation	cardinality
Syntax	`$B.card( S )`
Parameter	`S` - a set
Return	1) an instance of `$B.Integer` if `S` is finite 2) `undefined` if `S` is infinite
Description	This function returns the cardinality of an input set.

Example	`$B.card($B.SetExtension($B('2'), $B('1'), $B('3')))` returns `$B('3')`
	`$B.card($B.EmptySet)` returns `$B('0')`
	`$B.card($B.INTEGER)` returns `undefined`

C.4.19 $B.min

Notation	arithmetic minimum
Syntax	`$B.min( S )`
Parameter	`S` - a set of integers
Return	1) an instance of `$B.Integer` if all member of `S` are instances of `$B.Integer` 2) `undefined` if `S` is not a set or `S` is an empty set or any member of `S` is not an instance of `$B.Integer`
Description	This function returns the smallest integer of an input set.

Example	`$B.min($B.SetExtension($B('2'), $B('1'), $B('3')))` returns `$B('1')`
	`$B.min($B.EmptySet)` returns `undefined`
	`$B.min(undefined)` returns `undefined`

C.4.20 $B.max

Notation	arithmetic maximum
Syntax	`$B.max( S )`
Parameter	`S` - a set of integers
Return	1) an instance of `$B.Integer` if all member of `S` are instances of `$B.Integer` 2) `undefined` if `S` is not a set or `S` is an empty set or any member of `S` is not an instance of `$B.Integer`
Description	This function returns the largest integer of an input set.

Example	`$B.max($B.SetExtension($B('2'), $B('1'), $B('3')))` returns `$B('3')`
	`$B.max($B.EmptySet)` returns `undefined`
	`$B.max(undefined)` returns `undefined`

C.4.21 $B.belong

Notation	set membership predicate
Syntax	`$B.belong( E, S )`
Parameter	`E` - an expression `S` - a set
Return	`true` if `E` is a member of `S`, otherwise `false`
Description	This function checks if an expression denotes an element of a set.
Example	`$B.belong($B('5'), $B.INTEGER)` returns `true` `$B.belong($B('5'), $B.EmptySet)` returns `false`

C.4.22 $B.notBelong

Notation	not a set membership predicate
Syntax	`$B.notBelong( E, S )`
Parameter	`E` - an expression `S` - a set
Return	`true` if `E` is not a member of `S`, otherwise `false`
Description	This function checks if an expression does not denote an element of a set.
Example	`$B.notBelong($B('5'), $B.INTEGER)` returns `false` `$B.notBelong($B('5'), $B.EmptySet)` returns `true`

C.4.23 $B.properSubset

Notation	proper subset predicate
Syntax	`$B.properSubset( S, T )`
Parameter	`S, T` - two sets
Return	`true` if `S` is a proper subset of `T`, otherwise `false`
Description	This function checks if a set denotes a proper subset of another set.
Example	`$B.properSubset($B.UpTo($B('1'), $B('3')), $B.INTEGER)` returns `true` `$B.properSubset($B.UpTo($B('1'), $B('3')), $B.UpTo($B('1'), $B('3')))` returns `false`

C.4.24 $B.notProperSubset

Notation	not a proper subset predicate
Syntax	`$B.notProperSubset( S, T )`
Parameter	`S, T` - two sets
Return	`true` if `S` is not a proper subset of `T`, otherwise `false`
Description	This function checks if a set does not denotes a proper subset of another set.
Example	`$B.notProperSubset($B.UpTo($B('1'), $B('3')), $B.INTEGER)` returns `false` `$B.notProperSubset($B.UpTo($B('1'), $B('3')), $B.UpTo($B('1'), $B('3')))` returns `true`

C.4.25 $B.subset

Notation	subset predicate
Syntax	`$B.subset( S, T )`
Parameter	`S, T` - two sets
Return	`true` if `S` is a subset of `T`, otherwise `false`
Description	This function checks if a set denotes a subset of another set.
Example	`$B.subset($B.UpTo($B('1'), $B('3')), $B.INTEGER)` returns `true` `$B.subset($B.UpTo($B('1'), $B('3')), $B.UpTo($B('1'), $B('3')))` returns `true`

C.4.26 $B.notSubset

Notation	not a subset predicate
Syntax	`$B.notSubset( S, T )`
Parameter	`S, T` - two sets
Return	`true` if `S` is not a subset of `T`, otherwise `false`
Description	This function checks if a set does not denotes a subset of another set.
Example	`$B.notSubset($B.UpTo($B('1'), $B('3')), $B.INTEGER)` returns `false` `$B.notSubset($B.UpTo($B('1'), $B('3')), $B.UpTo($B('1'), $B('3')))` returns `false`

C.4.27 $B.finite

Notation	finite set predicate
Syntax	`$B.finite( S )`
Parameter	`S` - a set
Return	`true` if `S` is finite, otherwise `false`
Description	This function checks if an input set denotes a finite set.

Example	`$B.finite($B.SetExtension($B('2'), $B('1')))` returns `true`
	`$B.finite($B.EmptySet)` returns `true`
	`$B.finite($B.INTEGER)` returns `false`

C.4.28 $B.partition

Notation	partition predicate
Syntax	`$B.partition( S1[, ...[, Sn]] )`
Parameter	`S1, ..., Sn` - sets
Return	`true` if the sets `S2, ..., Sn` constitute a partition of `S1`, otherwise `false`
Description	This function checks if some sets are a partition of a set. The partition is defined as $(S1 = S2 \cup \ldots \cup Sn) \wedge$ $(\forall i, j \cdot i \in 2..n \wedge j \in 2..n \wedge i \neq j \Rightarrow Si \cap Sj = \varnothing)$.
Example	`$B.partition( $B.SetExtension($B('10'), $B('20')), $B.SetExtension($B('10')), $B.SetExtension($B('20')) )` returns `true`

In Event-B, there are two special cases, the $partition(S1)$ is equivalent to $S1 = \varnothing$ and the $partition(S1, S2)$ is equivalent to $S1 = S2$.

C.5 Relations

C.5.1 $B.Pair

Notation	ordered pair
Syntax	`$B.Pair( left, right )`
Parameter	`left, right` - two expressions
Return	an instance of `$B.Pair` object
Description	This function is the pair object constructor. The instance of `$B.pair` has two properties `left` and `right`.
Example	`$B.Pair($B('1'), $B('3'))` `$B.Pair($B('5'), $B.TRUE)`

C.5.2 $B.CartesianProduct

Notation	Cartesian product
Syntax	`$B.CartesianProduct( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.CartesianProduct` object
Description	This function is the Cartesian product object constructor. The instance of `$B.CartesianProduct` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.CartesianProduct($B.INTEGER, $B.BOOL)` `$B.CartesianProduct($B.INTEGER, $B.INTEGER)`

C.5.3 $B.Relations

Notation	relations
Syntax	`$B.Relations( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.Relations` object
Description	This function is the set of relations constructor. The instance of `$B.Relations` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.Relations($B.INTEGER, $B.BOOL)` `$B.Relations($B.INTEGER, $B.INTEGER)`

C.5.4 $B.TotalRelations

Notation	total relations
Syntax	`$B.TotalRelations( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalRelations` object
Description	This function is the set of total relations constructor. The instance of `$B.TotalRelations` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalRelations($B.INTEGER, $B.BOOL)` `$B.TotalRelations($B.INTEGER, $B.INTEGER)`

C.5.5 $B.SurjectiveRelations

Notation	surjective relations
Syntax	`$B.SurjectiveRelations( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.SurjectiveRelations` object
Description	This function is the set of surjective relations constructor. The instance of `$B.SurjectiveRelations` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.SurjectiveRelations($B.INTEGER, $B.BOOL)` `$B.SurjectiveRelations($B.INTEGER, $B.INTEGER)`

C.5.6 $B.TotalSurjectiveRelations

Notation	total surjective relations
Syntax	`$B.TotalSurjectiveRelations( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalSurjectiveRelations` object
Description	This function is the set of total surjective relations constructor. The instance of `$B.TotalSurjectiveRelations` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalSurjectiveRelations($B.INTEGER, $B.BOOL)` `$B.TotalSurjectiveRelations($B.INTEGER, $B.INTEGER)`

C.5.7 $B.dom

Notation	domain
Syntax	`$B.dom( r )`
Parameter	`r` - a relation
Return	an instance of `$B.SetExtension` object
Description	This function returns the domain of an input relation. The result is defined as $\{x \mid \exists y \cdot x \mapsto y \in r\}$. The computation should consider different forms of the input relation.
Example	`$B.dom( $B.SetExtension($B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))) )` returns `$B.SetExtension($B('1'), $B('3'))`

C.5.8 $B.ran

Notation	range
Syntax	`$B.ran( r )`
Parameter	`r` - a relation
Return	an instance of `$B.SetExtension` object
Description	This function returns the range of an input relation. The result is defined as $\{y \mid \exists x \cdot x \mapsto y \in r\}$. The computation should consider different forms of the input relation.
Example	`$B.ran( $B.SetExtension($B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))) )` returns `$B.SetExtension($B('2'), $B('4'))`

C.5.9 $B.domainRestriction

Notation	domain restriction
Syntax	`$B.domainRestriction( S, r )`
Parameter	`S` - a set `r` - a relation
Return	an instance of `$B.SetExtension` object
Description	This function computes a subset of an input relation that is defined as $\{x \mapsto y \mid x \mapsto y \in r \wedge x \in S\}$. The computation should consider different forms of the input relation.
Example	`$B.domainRestriction( $B.SetExtension($B('1')), $B.SetExtension($B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))) )` returns `$B.SetExtension($B.Pair($B('1'), $B('2')))`

C.5.10 $B.domainSubtraction

Notation	domain subtraction
Syntax	`$B.domainSubtraction( S, r )`
Parameter	`S` - a set `r` - a relation
Return	an instance of `$B.SetExtension` object
Description	This function computes a subset of an input relation that is defined as $\{x \mapsto y \mid x \mapsto y \in r \wedge x \notin S\}$. The computation should consider different forms of the input relation.

Example	`$B.domainSubtraction( $B.SetExtension($B('1')),`
	`$B.SetExtension($B.Pair($B('1'), $B('2')),`
	`$B.Pair($B('3'), $B('4'))) )`
	returns `$B.SetExtension($B.Pair($B('3'), $B('4')))`

C.5.11 $B.rangeRestriction

Notation	range restriction
Syntax	`$B.rangeRestriction( r, T )`
Parameter	`r` - a relation `T` - a set
Return	an instance of `$B.SetExtension` object
Description	This function computes a subset of an input relation that is defined as $\{x \mapsto y \mid x \mapsto y \in r \wedge y \in T\}$. The computation should consider different forms of the input relation.

Example	`$B.rangeRestriction( $B.SetExtension(`
	`$B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))),`
	`$B.SetExtension($B('2')) )`
	returns `$B.SetExtension($B.Pair($B('1'), $B('2')))`

C.5.12 $B.rangeSubtraction

Notation	range subtraction
Syntax	`$B.rangeSubtraction( r, T )`
Parameter	`r` - a relation `T` - a set
Return	an instance of `$B.SetExtension` object
Description	This function computes a subset of an input relation that is defined as $\{x \mapsto y \mid x \mapsto y \in r \wedge y \notin T\}$. The computation should consider different forms of the input relation.

Example	`$B.rangeSubtraction( $B.SetExtension(`
	`$B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))),`
	`$B.SetExtension($B('2')) )`
	returns `$B.SetExtension($B.Pair($B('3'), $B('4')))`

C.5.13 $B.forwardComposition

Notation	relational forward composition
Syntax	`$B.forwardComposition( r1, r2[, ...[, rN]] )`
Parameter	`r1, r2, ..., rN` - relations
Return	an instance of `$B.SetExtension` object
Description	This function computes the forward composition of input relations that is defined as $\{x_1 \mapsto x_{n+1} \mid \exists x_2, \ldots, x_n \cdot x_1 \mapsto x_2 \in r_1 \wedge \ldots \wedge x_n \mapsto x_{n+1} \in r_n\}$. The computation should consider different forms of the input relation.
Example	`$B.forwardComposition( $B.SetExtension( $B.Pair($B('1'), $B('2'))), $B.SetExtension( $B.Pair($B('2'), $B('4'))) )` returns `$B.SetExtension($B.Pair($B('1'), $B('4')))`

C.5.14 $B.backwardComposition

Notation	relational backward composition
Syntax	`$B.backwardComposition( r1, r2[, ...[, rN]] )`
Parameter	`r1, r2, ..., rN` - relations
Return	an instance of `$B.SetExtension` object
Description	This function computes the backward composition of input relations that is defined as `$B.forwardComposition( rN, ..., r1 )`.
Example	`$B.backwardComposition( $B.SetExtension( $B.Pair($B('2'), $B('4'))), $B.SetExtension( $B.Pair($B('1'), $B('2'))) )` returns `$B.SetExtension($B.Pair($B('1'), $B('4')))`

C.5.15 $B.override

Notation	relational overriding
Syntax	`$B.override( r1, r2[, ...[, rN]] )`
Parameter	`r1, r2, ..., rN` - relations
Return	an instance of `$B.SetExtension` object
Description	This function computes the overriding of input relations from left to right. The overriding operation is defined as: $r_i \lhd r_{i+1} = r_{i+1} \cup (dom(r_{i+1}) \lhd r_i)$.
Example	`$B.override( $B.SetExtension( $B.Pair($B('1'), $B('2'))), $B.SetExtension( $B.Pair($B('1'), $B('5'))) )` returns `$B.SetExtension($B.Pair($B('1'), $B('5')))`

C.5.16 $B.directProduct

Notation	direct product
Syntax	`$B.directProduct( r1, r2 )`
Parameter	`r1, r2` - two relations
Return	an instance of `$B.SetExtension` object
Description	This function computes the direct product of two input relations that is defined as: $\{x_1 \mapsto (x_2 \mapsto x_3) \mid x_1 \mapsto x_2 \in r_1 \wedge x_1 \mapsto x_3 \in r_2\}$
Example	`$B.directProduct( $B.SetExtension( $B.Pair($B('1'), $B('2'))), $B.SetExtension( $B.Pair($B('1'), $B('4'))) )` returns `$B.SetExtension($B.Pair($B('1'), $B.Pair($B('2'), $B('4')))`

C.5.17 $B.parallelProduct

Notation	parallel product
Syntax	`$B.parallelProduct( r1, r2 )`
Parameter	`r1, r2` - two relations
Return	an instance of `$B.SetExtension` object
Description	This function computes the direct product of two input relations that is defined as: $\{(x_1 \mapsto x_3) \mapsto (x_2 \mapsto x_4) \mid x_1 \mapsto x_2 \in r_1 \wedge x_3 \mapsto x_4 \in r_2\}$
Example	`$B.parallelProduct( $B.SetExtension( $B.Pair($B('1'), $B('2'))), $B.SetExtension( $B.Pair($B('3'), $B('4'))) )` returns `$B.SetExtension( $B.Pair($B.Pair($B('1'), $B('3')), $B.Pair($B('2'), $B('4'))) )`

C.5.18 $B.converse

Notation	inverse relation
Syntax	`$B.converse( r )`
Parameter	`r` - a relation
Return	an instance of `$B.SetExtension` object
Description	This function computes the inverse of an input relation that is defined as $\{y \mapsto x \mid x \mapsto y \in r\}$
Example	`$B.converse( $B.SetExtension($B.Pair($B('1'), $B('2')), $B.Pair($B('3'), $B('4'))) )` returns `$B.SetExtension($B.Pair($B('2'), $B('1')), $B.Pair($B('4'), $B('3'))) )`

C.5.19 $B.relationImage

Notation	relation image
Syntax	`$B.relationImage( r, S )`
Parameter	`r` - a relation `S` - a set
Return	an instance of `$B.SetExtension` object
Description	This function computes the relational image of an input relation that is defined as $\{y \mid \exists x \cdot x \in S \wedge x \mapsto y \in r\}$

Example	`$B.relationImage( $B.SetExtension(`
	`$B.Pair($B('2'), $B('5')), $B.Pair($B('2'), $B('6'))),`
	`$B.SetExtension($B('2')) )`
	returns `$B.SetExtension($B('5'), $B('6'))`

C.6 Functions

C.6.1 $B.PartialFunctions

Notation	partial functions
Syntax	`$B.PartialFunctions( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.PartialFunctions` object
Description	This function is the set of partial functions constructor. The instance of `$B.PartialFunctions` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.PartialFunctions($B.INTEGER, $B.BOOL)` `$B.PartialFunctions($B.INTEGER, $B.INTEGER)`

C.6.2 $B.TotalFunctions

Notation	total functions
Syntax	`$B.TotalFunctions( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalFunctions` object
Description	This function is the set of total functions constructor. The instance of `$B.TotalFunctions` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalFunctions($B.INTEGER, $B.BOOL)` `$B.TotalFunctions($B.INTEGER, $B.INTEGER)`

C.6.3 $B.PartialInjections

Notation	partial injections
Syntax	`$B.PartialInjections( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.PartialInjections` object
Description	This function is the set of partial injections constructor. The instance of `$B.PartialInjections` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.PartialInjections($B.INTEGER, $B.BOOL)` `$B.PartialInjections($B.INTEGER, $B.INTEGER)`

C.6.4 $B.TotalInjections

Notation	total injections
Syntax	`$B.TotalInjections( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalInjections` object
Description	This function is the set of total injections constructor. The instance of `$B.TotalInjections` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalInjections($B.INTEGER, $B.BOOL)` `$B.TotalInjections($B.INTEGER, $B.INTEGER)`

C.6.5 $B.PartialSurjections

Notation	partial surjections
Syntax	`$B.PartialSurjections( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.PartialSurjections` object
Description	This function is the set of partial surjections constructor. The instance of `$B.PartialSurjections` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.PartialSurjections($B.INTEGER, $B.BOOL)` `$B.PartialSurjections($B.INTEGER, $B.INTEGER)`

C.6.6 $B.TotalSurjections

Notation	total surjections
Syntax	`$B.TotalSurjections( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalSurjections` object
Description	This function is the set of total surjections constructor. The instance of `$B.TotalSurjections` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalSurjections($B.INTEGER, $B.BOOL)` `$B.TotalSurjections($B.INTEGER, $B.INTEGER)`

C.6.7 $B.TotalBijections

Notation	total bijections
Syntax	`$B.TotalBijections( S, T )`
Parameter	`S, T` - two sets
Return	an instance of `$B.TotalBijections` object
Description	This function is the set of total surjections constructor. The instance of `$B.TotalBijections` has two properties `S` and `T`. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	`$B.TotalBijections($B.INTEGER, $B.BOOL)` `$B.TotalBijections($B.INTEGER, $B.INTEGER)`

C.6.8 $B.Lambda

Notation	lambda
Syntax	`$B.Lambda( predicateFunction,` `expressionFunction, domainArray )`
Parameter	The parameters definition is same to the set comprehension interface in Sect. C.4.8.
Return	an instance of `$B.Lambda` object
Description	This function is the lambda constructor. It normally exists in an abstract form, but has the capability to be transformed as a concrete set extension on-demand.
Example	sees below

Invoking the lambda $\lambda x \cdot x \in 1..10 \mid x + 10$ with 5 yields 15, with 11 yields `undefined`, where

```
predicateFunction ≙ function( x ) {
    return $B.belong(x, $B.SetExtension($B('1'), $B('10')));
}
```

```
expressionFunction ≙ function( x ) {
    return $B.Pair(x, $B.plus( x, $B('10'));
}

domainArray ≙ [$B.UpTo($B('1'), $B('10'))]
```

C.6.9 $B.functionImage

Notation	function image
Syntax	`$B.functionImage( f, E )`
Parameter	`f` - a function `E` - a expression
Return	1) `F` if `E` is in the domain of `f` and $E \mapsto F \in f$ 2) `undefined` if `E` is not in the domain of `f`
Description	`f` can denote a set extension of ordered pairs, a set comprehension or a lambda expression, hence the implementation must consider all above situations.

Example	`$B.functionImage( $B.SetExtension(`
	`$B.Pair($B('2'), $B('5')), $B.Pair($B('3'), $B('6'))),`
	`$B('3') )` returns `$B('6')`

C.6.10 $B.prj1

Notation	first projection
Syntax	`$B.prj1`
Parameter	-
Return	an instance of `B.SetComprehension`
Description	The first projection maps a pair to its first element. Its definition is generic by $\{(x \mapsto y) \mapsto x \mid \top\}$ which type is inferred from the environment.

Example	`$B.functionImage( $B.prj1, $B.Pair($B('2'), $B('5')) )`
	returns `$B('2')`
	`$B.setInter( $B.SetExtension(`
	`$B.Pair($B.Pair($B('1'), $B('2')), $B('1')),`
	`$B.Pair($B.Pair($B('2'), $B('4')), $B('3'))),`
	`$B.prj1 )`
	returns `$B.Pair($B.Pair($B('1'), $B('2')), $B('1'))`

When `$B.prj1` is used in the function application, the following definition can be used directly:

```
$B.prj1 = $B.SetComprehension(
    function( x, y ) { return true; },
    function( x, y ) { return $B.Pair( $B.Pair( x, y ), x ); },
    [$B.EmptySet, $B.EmptySet]
```

```
)
```

But when `$B.prj1` is used in the set operations, users should replace the `$B.EmptySet` by a concrete set.

C.6.11 $B.prj2

Notation	second projection
Syntax	`$B.prj2`
Parameter	-
Return	an instance of `B.SetComprehension`
Description	The second projection maps a pair to its second element. Its definition is generic by $\{(x \mapsto y) \mapsto y \mid \top\}$ which type is inferred from the environment.

Example	`$B.functionImage( $B.prj2, $B.Pair($B('2'), $B('5')) )` returns `$B('5')`
	`$B.setInter( $B.SetExtension(`
	`$B.Pair($B.Pair($B('1'), $B('2')), $B('2')),`
	`$B.Pair($B.Pair($B('2'), $B('4')), $B('3'))),`
	`$B.prj2 )`
	returns `$B.Pair($B.Pair($B('1'), $B('2')), $B('2'))`

When `$B.prj2` is used in the function application, the following definition can be used directly:

```
$B.prj2 = $B.SetComprehension(
    function( x, y ) { return true; },
    function( x, y ) { return $B.Pair( $B.Pair( x, y ), y ); },
    [$B.EmptySet, $B.EmptySet]
)
```

But when `$B.prj2` is used in the set operations, users should replace the `$B.EmptySet` by a concrete set.

C.6.12 $B.id

Notation	identity
Syntax	`$B.id`
Parameter	-
Return	an instance of `B.SetComprehension`
Description	The identity function maps every element to itself. Its definition is generic by $\{x \mapsto x \mid \top\}$ which type is inferred from the environment.

Example	`$B.functionImage( $B.id, $B('2') )` returns `$B('2')`
	`$B.setInter( $B.SetExtension($B.Pair($B('1'), $B('1'))`
	`$B.Pair($B('2'), $B('3'))), $B.id )`
	returns `$B.SetExtension($B.Pair($B('1'), $B('1')))`

When `$B.id` is used in the function application, the following definition can be used directly:

```
$B.id = $B.SetComprehension(
    function( x ) { return true; },
    function( x ) { return $B.Pair( x, x ); },
    [$B.EmptySet]
)
```

But when `$B.id` is used in the set operations, users should replace the `$B.EmptySet` by a concrete set.

C.7 Assignments

C.7.1 $B.becomesEqualTo

Notation	deterministic assignment
Syntax	`$B.becomesEqualTo( identifierArray, expressionArray )`
Parameter	`identifierArray` - an array, it contains assigned identifiers `expressionArray` - an array, it contains expressions
Return	-
Description	This function does the deterministic assignment. Let n be the length of the array `identifierArray`, for each identifier x_i in `identifierArray` and each expression E_i in `expressionArray`, it assigns the expression E_i to the identifier x_i primed value, with $i \in 0..n-1$.
Example	`$B.becomesEqualTo( [x], [$B('100')] )` `$B.becomesEqualTo( [x, y], [$B.TRUE, $B('10')] )`

There is a special assignment which uses a relational overriding: $x(E) := F$. In fact, this is just a shorthand for $x := x \Leftarrow \{E \mapsto F\}$, therefore this form is unified to the above form.

C.7.2 $B.becomesMemberOf

Notation	non-deterministic assignment of a set member
Syntax	`$B.becomesMemberOf( x, E )`
Parameter	`x` - an assigned identifier `E` - a set expression
Return	-
Description	This function assigns any value of the set expression `E` to the identifier `x` primed value.
Example	`$B.becomesMemberOf( x, $B.BOOL )` `$B.becomesMemberOf( x, $B.INTEGER )`

C.7.3 $B.becomesSuchThat

Notation	non-deterministic assignment with a before-after-predicate
Syntax	`$B.becomesSuchThat( identifierArray, predicateFunction, domainArray )`
Parameter	`identifierArray` - an array, it contains assigned identifiers `predicateFunction` - a function with the before-after-predicate as its function body `domainArray` - an array, it contains the enumerated sets for each primed identifier in the before-after-predicate
Return	-
Description	This function does the non-deterministic assignment. Let n be the length of the array `identifierArray`, for each identifier x_i in `identifierArray` and each expression E_i in `domainArray`, with $i \in 0..n-1$, it assigns any value of the set expression E_i to the identifier x_i primed value and evaluates the function `predicateFunction` until `predicateFunction` returns `true`, otherwise throws a message to remind that no enumerated values can satisfy the before-after predicate.
Example	sees below

Let `x._value, y._value` be the primed value of identifiers `x, y`, respectively, executing the assignment $x, y :\mid x' \in 1..100 \wedge y' \in 1..100 \wedge x' + y' > 100$ yields `x._value` has a value `1` and `y._value` has a value `100`, where

```
identifierArray ≙ [x, y]

predicateFunction ≙ function() {
    return $B.and(
        $B.belong(x._value, $B.UpTo($B('1'), $B('100'))),
        $B.belong(y._value, $B.UpTo($B('1'), $B('100'))),
        $B.greaterThan($B.plus(x._value, y._value), $B('100'))
    );
}

domainArray ≙ [$B.UpTo($B('1'), $B('100')), $B.UpTo($B('1'), $B('100'))]
```

Appendix D

1D Platooning Model in Event-B

Contents

A platoon is defined as a convoy of autonomous vehicles which follow exactly the same path and keep a very close distance between each other. We use a model of vehicle where the control can be decomposed into longitudinal (i.e, speed and acceleration) and lateral (i.e., curve and wheel orientation) laws. The 1D platooning model focuses on the longitudinal control.

Note: we have made small modifications to the labels of the original specification. The labels start with a prefix and are followed by an ordinal number.

D.1 Context0

CONTEXT context0
CONSTANTS
$VEHICLES$
$CRITICAL\_DISTANCE$
$initial\_xpos$
AXIOMS
axm1 : $VEHICLES \in \mathbb{N}_1$
axm2 : $VEHICLES \geq 2$
axm3 : $CRITICAL\_DISTANCE \in \mathbb{N}_1$
axm4 : $initial\_xpos \in 1 \mathinner{..} VEHICLES \rightarrow \mathbb{N}$

axm5 : $\forall v \cdot (v \in 1..VEHICLES \Rightarrow initial\_xpos(v) = (VEHICLES - v) * (CRITICAL\_DISTANCE + 1))$

END

D.2 Context1

CONTEXT context1
EXTENDS context0
END

D.3 Context2

CONTEXT context2
EXTENDS context1
CONSTANTS

MAX_SPEED
MIN_ACCEL
MAX_ACCEL
$initial\_speed$
new_speed
new_xpos
new_xpos_max
new_xpos_min

AXIOMS

axm1 : $MAX\_SPEED \in \mathbb{N}_1$

axm2 : $MAX\_ACCEL \in \mathbb{N}_1$

axm3 : $MIN\_ACCEL \in \mathbb{Z}$

axm4 : $MIN\_ACCEL < 0$

axm5 : $initial\_speed \in 1..VEHICLES \rightarrow 0..MAX\_SPEED$

axm6 : $\forall vehi0 \cdot (vehi0 \in 1..VEHICLES \Rightarrow (\exists speed0 \cdot (speed0 \in 0..MAX\_SPEED \wedge initial\_speed(vehi0) = speed0)))$

axm7 : $new\_speed \in (0..MAX\_SPEED \times MIN\_ACCEL..MAX\_ACCEL) \rightarrow \mathbb{Z}$

axm8 : $\forall speed1, accel1 \cdot (speed1 \in 0..MAX\_SPEED \wedge accel1 \in MIN\_ACCEL..MAX\_ACCEL \Rightarrow new\_speed(speed1 \mapsto accel1) = speed1 + accel1)$

axm9 : $new\_xpos \in (\mathbb{N} \times 0..MAX\_SPEED \times MIN\_ACCEL..MAX\_ACCEL) \rightarrow \mathbb{N}$

axm10 : $\forall xpos0, speed0, accel0 \cdot (xpos0 \in \mathbb{N} \wedge speed0 \in 0..MAX\_SPEED \wedge accel0 \in MIN\_ACCEL..MAX\_ACCEL \Rightarrow new\_xpos(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + speed0 + (accel0/2))$

axm11 : $new\_xpos\_max \in \mathbb{N} \times 0..MAX\_SPEED \times MIN\_ACCEL..MAX\_ACCEL \rightarrow \mathbb{N}$

axm12 : $\forall xpos0, speed0, accel0 \cdot (xpos0 \in \mathbb{N} \wedge speed0 \in 0..MAX\_SPEED \wedge accel0 \in MIN\_ACCEL..MAX\_ACCEL \Rightarrow ((accel0 = 0 \Rightarrow new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED) \wedge (accel0 \neq 0 \Rightarrow new\_xpos\_max(xpos0 \mapsto speed0 \mapsto accel0) = xpos0 + MAX\_SPEED - (((MAX\_SPEED - speed0) * (MAX\_SPEED - speed0))/(2 * accel0)))))$

axm13 : $new\_xpos\_min \in \mathbb{N} \times 0..MAX\_SPEED \times MIN\_ACCEL..MAX\_ACCEL \rightarrow \mathbb{N}$

axm14 : $\forall xpos0, speed0, accel0 \cdot ($
$xpos0 \in \mathbb{N} \land speed0 \in 0 \mathinner{..} MAX\_SPEED \land accel0 \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL \Rightarrow$
$($
$(accel0 = 0 \Rightarrow new\_xpos\_min(xpos0 \mapsto speed0 \mapsto accel0) = xpos0)$
$\land$
$(accel0 \neq 0 \Rightarrow new\_xpos\_min(xpos0 \mapsto speed0 \mapsto accel0) =$
$xpos0 - ((speed0 * speed0)/(2 * accel0)))$
$)$
$)$

END

D.4 Context3

CONTEXT context3
EXTENDS context2
CONSTANTS
initial_accel
AXIOMS
axm1 : $initial\_accel \in 1 \mathinner{..} VEHICLES \rightarrow MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
axm2 : $\forall vehi0 \cdot (vehi0 \in 1 \mathinner{..} VEHICLES \Rightarrow$
$\exists accel0 \cdot (accel0 \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL \land initial\_accel(vehi0) = accel0)$
$)$
END

D.5 Context4

CONTEXT context4
EXTENDS context3
CONSTANTS
IDEAL_SPEED
ideal_distance
new_accel
AXIOMS
axm1 : $IDEAL\_SPEED \in 0 \mathinner{..} MAX\_SPEED$
axm2 : $IDEAL\_SPEED < MAX\_SPEED$
axm3 : $ideal\_distance \in 0 \mathinner{..} MAX\_SPEED \rightarrow \mathbb{N}$
axm4 : $\forall speed0 \cdot (speed0 \in 0 \mathinner{..} MAX\_SPEED \Rightarrow ideal\_distance(speed0) = CRITICAL\_DISTANCE + speed0)$
axm5 : $new\_accel \in (\mathbb{Z} \times 0 \mathinner{..} MAX\_SPEED \times 0 \mathinner{..} MAX\_SPEED) \rightarrow \mathbb{Z}$
axm6 : $\forall p\_dist1, p\_speed1, p\_pre\_speed1 \cdot ($
$p\_dist1 \in \mathbb{Z} \land p\_speed1 \in 0 \mathinner{..} MAX\_SPEED \land p\_pre\_speed1 \in 0 \mathinner{..} MAX\_SPEED \Rightarrow$
$new\_accel(p\_dist1 \mapsto p\_speed1 \mapsto p\_pre\_speed1) = p\_dist1 -$
$ideal\_distance(p\_speed1) + p\_pre\_speed1 - p\_speed1$
$)$
thm1 : $\forall speed \cdot (speed \in 0 \mathinner{..} MAX\_SPEED \Rightarrow ideal\_distance(speed) \geq CRITICAL\_DISTANCE)$
END

D.6 Platoon0

MACHINE platoon0
SEES context0
VARIABLES
xpos0
INVARIANTS
inv1 : $xpos0 \in 1 \mathinner{..} VEHICLES \rightarrow \mathbb{N}$

inv2: $\forall v \cdot (v \in 2..VEHICLES \Rightarrow ((xpos0(v-1) - xpos0(v)) > CRITICAL\_DISTANCE))$

EVENTS

Initialisation

begin

act1: $xpos0 := initial\_xpos$

end

Event *all_moves* $\widehat{=}$

any

magic_xpos

where

grd1: $magic\_xpos \in 1..VEHICLES \rightarrow \mathbb{N}$

grd2: $\forall v \cdot (v \in 2..VEHICLES \Rightarrow ((magic\_xpos(v-1) - magic\_xpos(v)) > CRITICAL\_DISTANCE))$

then

act1: $xpos0 := magic\_xpos$

end

END

D.7 Platoon1

MACHINE platoon1

REFINES platoon0

SEES context1

VARIABLES

xpos0

vehicle

xpos

INVARIANTS

inv1: $xpos \in 1..VEHICLES \rightarrow \mathbb{N}$

inv2: $vehicle \in 1..VEHICLES + 1$

inv3: $\forall v \cdot (v \in 2..vehicle - 1 \Rightarrow (xpos(v-1) - xpos(v)) > CRITICAL\_DISTANCE)$

EVENTS

Initialisation

begin

act1: $xpos0 := initial\_xpos$

act2: $vehicle := 1$

act3: $xpos := initial\_xpos$

end

Event *move1* $\widehat{=}$

Status convergent

any

magic_xpos_vehicle

where

grd1: $vehicle = 1$

grd2: $magic\_xpos\_vehicle \in \mathbb{N}$

grd3: $magic\_xpos\_vehicle \geq xpos(vehicle)$

then

act1: $xpos(vehicle) := magic\_xpos\_vehicle$

act2: $vehicle := vehicle + 1$

end

Event *move* $\widehat{=}$

Status convergent

any

magic_xpos_vehicle

where

grd1 : $vehicle \in 2..VEHICLES$
grd2 : $magic\_xpos\_vehicle \in \mathbb{N}$
grd3 : $magic\_xpos\_vehicle \geq xpos(vehicle)$
grd4 : $xpos(vehicle-1) - magic\_xpos\_vehicle > CRITICAL\_DISTANCE$

then

act1 : $xpos(vehicle) := magic\_xpos\_vehicle$
act2 : $vehicle := vehicle + 1$

end

Event *all_moves* $\widehat{=}$
refines *all_moves*

when

grd1 : $vehicle = VEHICLES + 1$

with

magic_xpos : $magic\_xpos = xpos$

then

act1 : $xpos0 := xpos$
act2 : $vehicle := 1$

end

VARIANT
$(VEHICLES+1) - vehicle$
END

D.8 Platoon2

MACHINE platoon2
REFINES platoon1
SEES context2
VARIABLES
*xpos*0
vehicle
xpos
speed
INVARIANTS
inv1 : $speed \in 1..VEHICLES \rightarrow 0..MAX\_SPEED$
EVENTS
Initialisation

begin

act1 : $xpos0 := initial\_xpos$
act2 : $xpos := initial\_xpos$
act3 : $vehicle := 1$
act4 : $speed := initial\_speed$

end

Event *move1_normal* $\widehat{=}$
refines *move1*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle = 1$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$

grd4 : $nspeed \in 0..MAX\_SPEED$
grd5 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := nspeed$

end

Event *move1_max* $\widehat{=}$
refines *move1*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle = 1$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$
grd4 : $nspeed > MAX\_SPEED$
grd5 : $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := MAX\_SPEED$

end

Event *move1_reduce* $\widehat{=}$
refines *move1*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle = 1$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$
grd4 : $nspeed < 0$
grd5 : $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := 0$

end

Event *move_normal* $\widehat{=}$
refines *move*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle \in 2..VEHICLES$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$
grd4 : $nspeed \in 0..MAX\_SPEED$
grd5 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$
grd6 : $xpos(vehicle-1) - nxpos > CRITICAL\_DISTANCE$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := nspeed$

end

Event *move_max* $\widehat{=}$
refines *move*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle \in 2..VEHICLES$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$
grd4 : $nspeed > MAX\_SPEED$
grd5 : $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$
grd6 : $xpos(vehicle-1) - nxpos > CRITICAL\_DISTANCE$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := MAX\_SPEED$

end

Event *move_reduce* $\widehat{=}$
refines *move*

any

magic_accel
nspeed
nxpos

where

grd1 : $vehicle \in 2..VEHICLES$
grd2 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto magic\_accel)$
grd4 : $nspeed < 0$
grd5 : $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto magic\_accel)$
grd6 : $xpos(vehicle-1) - nxpos > CRITICAL\_DISTANCE$

with

magic_xpos_vehicle : $magic\_xpos\_vehicle = nxpos$

then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := 0$

end

Event *all_moves* $\widehat{=}$

refines *all_moves*
when
grd1 : $vehicle = VEHICLES + 1$
then
act1 : $xpos0 := xpos$
act2 : $vehicle := 1$
end
END

D.9 Platoon3

MACHINE platoon3
REFINES platoon2
SEES context3
VARIABLES
$xpos0$
$vehicle$
$xpos$
$speed$
$d\_vehicle$
$accel$
INVARIANTS
inv1 : $d\_vehicle \in 1..VEHICLES + 1$
inv2 : $accel \in 1..VEHICLES \rightarrow MIN\_ACCEL..MAX\_ACCEL$
inv3 : $(d\_vehicle = VEHICLES + 1) \lor$
$\forall v \cdot (v \in 2..d\_vehicle - 1 \Rightarrow$
$\exists f1, f2 \cdot (f1 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \land$
$f2 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \land$
$f1(xpos(v-1) \mapsto speed(v-1) \mapsto accel(v-1)) - f2(xpos(v) \mapsto speed(v) \mapsto accel(v)) >$
$CRITICAL\_DISTANCE$
)
)
EVENTS
Initialisation
begin
act1 : $xpos0 := initial\_xpos$
act2 : $xpos := initial\_xpos$
act3 : $vehicle := 1$
act4 : $speed := initial\_speed$
act5 : $d\_vehicle := 1$
act6 : $accel := initial\_accel$
end

Event *decide1* $\widehat{=}$
Status convergent
any
$magic\_accel$
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := magic\_accel$
end

Event *decide* $\widehat{=}$

Status convergent

any

$magic\_accel$

where

grd1 : $vehicle = 1$

grd2 : $d\_vehicle \in 2..VEHICLES$

grd3 : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$

grd4 : $\exists g1, g2 \cdot ($
$g1 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge$
$g2 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge$
$g1(xpos(d\_vehicle-1) \mapsto speed(d\_vehicle-1) \mapsto accel(d\_vehicle-1))$
$- g2(xpos(d\_vehicle) \mapsto speed(d\_vehicle) \mapsto magic\_accel) > CRITICAL\_DISTANCE$
$)$

then

act1 : $d\_vehicle := d\_vehicle + 1$

act2 : $accel(d\_vehicle) := magic\_accel$

end

Event $move1\_normal \widehat{=}$

refines $move1\_normal$

any

$nspeed$
$nxpos$

where

grd1 : $vehicle = 1$

grd2 : $d\_vehicle = VEHICLES + 1$

grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$

grd4 : $nspeed \in 0..MAX\_SPEED$

grd5 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$

then

act1 : $vehicle := vehicle + 1$

act2 : $xpos(vehicle) := nxpos$

act3 : $speed(vehicle) := nspeed$

end

Event $move1\_max \widehat{=}$

refines $move1\_max$

any

$nspeed$
$nxpos$

where

grd1 : $vehicle = 1$

grd2 : $d\_vehicle = VEHICLES + 1$

grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$

grd4 : $nspeed > MAX\_SPEED$

grd5 : $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$

then

act1 : $vehicle := vehicle + 1$

act2 : $xpos(vehicle) := nxpos$

act3 : $speed(vehicle) := MAX\_SPEED$

end

Event $move1\_reduce \widehat{=}$

refines $move1\_reduce$

any
$nspeed$
$nxpos$
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd4 : $nspeed < 0$
grd5 : $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
with
magic_accel : $magic\_accel = accel(vehicle)$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := 0$
end

Event $move\_normal \mathrel{\widehat{=}}$
refines $move\_normal$
any
$nspeed$
$nxpos$
where
grd1 : $vehicle \in 2..VEHICLES$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd4 : $nspeed \in 0..MAX\_SPEED$
grd5 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd6 : $xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE$
with
magic_accel : $magic\_accel = accel(vehicle)$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := nspeed$
end

Event $move\_max \mathrel{\widehat{=}}$
refines $move\_max$
any
$nspeed$
$nxpos$
where
grd1 : $vehicle \in 2..VEHICLES$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd4 : $nspeed > MAX\_SPEED$
grd5 : $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd6 : $xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE$
with
magic_accel : $magic\_accel = accel(vehicle)$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := MAX\_SPEED$
end

Event *move_reduce* $\widehat{=}$
refines *move_reduce*
any
nspeed
nxpos
where
grd1 : $vehicle \in 2..VEHICLES$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd4 : $nspeed < 0$
grd5 : $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd6 : $xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE$
with
magic_accel : $magic\_accel = accel(vehicle)$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := 0$
end

Event *all_moves* $\widehat{=}$
refines *all_moves*
when
grd1 : $vehicle = VEHICLES + 1$
grd2 : $d\_vehicle = VEHICLES + 1$
then
act1 : $xpos0 := xpos$
act2 : $vehicle := 1$
act3 : $d\_vehicle := 1$
end
VARIANT
$(VEHICLES + 1) - d\_vehicle$
END

D.10 Platoon4

MACHINE platoon4
REFINES platoon3
SEES context4
VARIABLES
*xpos*0
vehicle
xpos
speed
d_vehicle
accel
p_vehicle
p_speed
p_pre_speed
p_dist
INVARIANTS
inv1 : $p\_vehicle \in 1..VEHICLES + 1$
inv2 : $p\_speed \in 1..VEHICLES \rightarrow 0..MAX\_SPEED$
inv3 : $(p\_vehicle = VEHICLES + 1) \vee$
$\forall v \cdot (v \in 1..p\_vehicle - 1 \Rightarrow p\_speed(v) = speed(v))$
inv4 : $p\_dist \in 2..VEHICLES \rightarrow \mathbb{Z}$

inv5 : $(p\_vehicle = VEHICLES + 1) \lor$
$\forall v \cdot (v \in 2 \mathinner{..} p\_vehicle - 1 \Rightarrow p\_dist(v) = xpos(v-1) - xpos(v))$
inv6 : $p\_pre\_speed \in 2 \mathinner{..} VEHICLES \to 0 \mathinner{..} MAX\_SPEED$
inv7 : $(p\_vehicle = VEHICLES + 1) \lor$
$\forall v \cdot (v \in 2 \mathinner{..} p\_vehicle - 1 \Rightarrow p\_pre\_speed(v) = speed(v-1))$

EVENTS

Initialisation

begin

act1 : $xpos0 := initial\_xpos$
act2 : $xpos := initial\_xpos$
act3 : $vehicle := 1$
act4 : $speed := initial\_speed$
act5 : $d\_vehicle := 1$
act6 : $accel := initial\_accel$
act7 : $p\_vehicle := 1$
act8 : $p\_speed := initial\_speed$
act9 : $p\_pre\_speed := \{1\} \lhd initial\_speed$
act10 : $p\_dist := \{1\} \lhd initial\_xpos$

end

Event *perceive1* $\widehat{=}$
Status convergent

when

grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $p\_vehicle = 1$

then

act1 : $p\_speed(p\_vehicle) := speed(p\_vehicle)$
act2 : $p\_vehicle := p\_vehicle + 1$

end

Event *perceive* $\widehat{=}$
Status convergent

when

grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $p\_vehicle \in 2 \mathinner{..} VEHICLES$

then

act1 : $p\_speed(p\_vehicle) := speed(p\_vehicle)$
act2 : $p\_dist(p\_vehicle) := xpos(p\_vehicle - 1) - xpos(p\_vehicle)$
act3 : $p\_pre\_speed(p\_vehicle) := speed(p\_vehicle - 1)$
act4 : $p\_vehicle := p\_vehicle + 1$

end

Event *decide1_normal* $\widehat{=}$
refines *decide1*

any

naccel

where

grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = IDEAL\_SPEED - p\_speed(d\_vehicle)$
grd5 : $naccel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$

with

magic_accel : $magic\_accel = naccel$

then

act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := naccel$
end

Event *decide1_max* $\widehat{=}$
refines *decide1*
any
$naccel$
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = IDEAL\_SPEED - p\_speed(d\_vehicle)$
grd5 : $naccel > MAX\_ACCEL$
with
magic_accel : $magic\_accel = MAX\_ACCEL$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := MAX\_ACCEL$
end

Event *decide1_min* $\widehat{=}$
refines *decide1*
any
$naccel$
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = IDEAL\_SPEED - p\_speed(d\_vehicle)$
grd5 : $naccel < MIN\_ACCEL$
with
magic_accel : $magic\_accel = MIN\_ACCEL$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := MIN\_ACCEL$
end

Event *decide_normal* $\widehat{=}$
refines *decide*
any
$naccel$
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle \in 2\,..\,VEHICLES$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = new\_accel(p\_dist(d\_vehicle) \mapsto p\_speed(d\_vehicle) \mapsto p\_pre\_speed(d\_vehicle))$
grd5 : $naccel \in MIN\_ACCEL\,..\,MAX\_ACCEL$
grd6 : $\exists g1, g2 \cdot$
$(g1 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge g2 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge$
$(g1(xpos(d\_vehicle - 1) \mapsto speed(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1))$
$- g2(xpos(d\_vehicle) \mapsto speed(d\_vehicle) \mapsto naccel) > CRITICAL\_DISTANCE)$
$)$
with
magic_accel : $magic\_accel = naccel$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := naccel$

end

Event *decide_max* $\widehat{=}$
refines *decide*
any
naccel
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle \in 2..VEHICLES$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = new\_accel(p\_dist(d\_vehicle) \mapsto p\_speed(d\_vehicle) \mapsto p\_pre\_speed(d\_vehicle))$
grd5 : $naccel > MAX\_ACCEL$
grd6 : $\exists g1, g2 \cdot$
$(g1 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge g2 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge$
$(g1(xpos(d\_vehicle-1) \mapsto speed(d\_vehicle-1) \mapsto accel(d\_vehicle-1))$
$- g2(xpos(d\_vehicle) \mapsto speed(d\_vehicle) \mapsto MAX\_ACCEL) > CRITICAL\_DISTANCE)$
$)$
with
magic_accel : $magic\_accel = MAX\_ACCEL$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := MAX\_ACCEL$
end

Event *decide_min* $\widehat{=}$
refines *decide*
any
naccel
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle \in 2..VEHICLES$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $naccel = new\_accel(p\_dist(d\_vehicle) \mapsto p\_speed(d\_vehicle) \mapsto p\_pre\_speed(d\_vehicle))$
grd5 : $naccel < MIN\_ACCEL$
grd6 : $\exists g1, g2 \cdot$
$(g1 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge g2 \in \{new\_xpos, new\_xpos\_max, new\_xpos\_min\} \wedge$
$(g1(xpos(d\_vehicle-1) \mapsto speed(d\_vehicle-1) \mapsto accel(d\_vehicle-1))$
$- g2(xpos(d\_vehicle) \mapsto speed(d\_vehicle) \mapsto MIN\_ACCEL) > CRITICAL\_DISTANCE)$
$)$
with
magic_accel : $magic\_accel = MIN\_ACCEL$
then
act1 : $d\_vehicle := d\_vehicle + 1$
act2 : $accel(d\_vehicle) := MIN\_ACCEL$
end

Event *move1_normal* $\widehat{=}$
refines *move1_normal*
any
nspeed
nxpos
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5 : $nspeed \in 0..MAX\_SPEED$
grd6 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
then

act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := nspeed$

end

Event *move1_max* $\widehat{=}$
refines *move1_max*
any
nspeed
nxpos
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5 : $nspeed > MAX\_SPEED$
grd6 : $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := MAX\_SPEED$
end

Event *move1_reduce* $\widehat{=}$
refines *move1_reduce*
any
nspeed
nxpos
where
grd1 : $vehicle = 1$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5 : $nspeed < 0$
grd6 : $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$
act3 : $speed(vehicle) := 0$
end

Event *move_normal* $\widehat{=}$
refines *move_normal*
any
nspeed
nxpos
where
grd1 : $vehicle \in 2\,..\,VEHICLES$
grd2 : $d\_vehicle = VEHICLES + 1$
grd3 : $p\_vehicle = VEHICLES + 1$
grd4 : $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5 : $nspeed \in 0\,..\,MAX\_SPEED$
grd6 : $nxpos = new\_xpos(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd7 : $xpos(vehicle - 1) - nxpos > CRITICAL\_DISTANCE$
then
act1 : $vehicle := vehicle + 1$
act2 : $xpos(vehicle) := nxpos$

act3: $speed(vehicle) := nspeed$

end

Event *move_max* $\widehat{=}$
refines *move_max*
any
$nspeed$
$nxpos$
where
grd1: $vehicle \in 2..VEHICLES$
grd2: $d\_vehicle = VEHICLES + 1$
grd3: $p\_vehicle = VEHICLES + 1$
grd4: $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5: $nspeed > MAX\_SPEED$
grd6: $nxpos = new\_xpos\_max(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd7: $xpos(vehicle-1) - nxpos > CRITICAL\_DISTANCE$
then
act1: $vehicle := vehicle + 1$
act2: $xpos(vehicle) := nxpos$
act3: $speed(vehicle) := MAX\_SPEED$
end

Event *move_reduce* $\widehat{=}$
refines *move_reduce*
any
$nspeed$
$nxpos$
where
grd1: $vehicle \in 2..VEHICLES$
grd2: $d\_vehicle = VEHICLES + 1$
grd3: $p\_vehicle = VEHICLES + 1$
grd4: $nspeed = new\_speed(speed(vehicle) \mapsto accel(vehicle))$
grd5: $nspeed < 0$
grd6: $nxpos = new\_xpos\_min(xpos(vehicle) \mapsto speed(vehicle) \mapsto accel(vehicle))$
grd7: $xpos(vehicle-1) - nxpos > CRITICAL\_DISTANCE$
then
act1: $vehicle := vehicle + 1$
act2: $xpos(vehicle) := nxpos$
act3: $speed(vehicle) := 0$
end

Event *all_moves* $\widehat{=}$
refines *all_moves*
when
grd1: $vehicle = VEHICLES + 1$
grd2: $d\_vehicle = VEHICLES + 1$
grd3: $p\_vehicle = VEHICLES + 1$
then
act1: $xpos0 := xpos$
act2: $vehicle := 1$
act3: $d\_vehicle := 1$
act4: $p\_vehicle := 1$
end
VARIANT
$(VEHICLES + 1) - p\_vehicle$
END

Appendix E

2D Platooning Model in Event-B

Contents

The 2D platooning model kept the same structure as the 1D model, but added the lateral control, so autonomous vehicles are now moving on a plane (2D).

E.1 Context0

CONTEXT context0
SETS
Point
CONSTANTS
VEHICLES
CRITICAL_DISTANCE
Y_DISTANCE
initial_pos
initial_trajectory
dist
y_dist
nearest
AXIOMS
axm_VEHI1: $VEHICLES \in \mathbb{N}_1$
axm_VEHI2: $VEHICLES \geq 2$
axm_dist1: $dist \in Point \times Point \rightarrow \mathbb{N}$
axm_dist2: $\forall x0 \cdot (x0 \in Point \Rightarrow dist(x0 \mapsto x0) = 0)$
axm_dist3: $\forall x0, y0 \cdot (x0 \in Point \wedge y0 \in Point \Rightarrow dist(x0 \mapsto y0) = dist(y0 \mapsto x0))$

axm_dist4: $\forall x0, y0, z0 \cdot (x0 \in Point \land y0 \in Point \land z0 \in Point \Rightarrow dist(x0 \mapsto y0) + dist(y0 \mapsto z0) \geq dist(x0 \mapsto z0))$

axm_init_pos: $initial\_pos \in 1..VEHICLES \rightarrow Point$

axm_CRIT_DIST1: $CRITICAL\_DISTANCE \in \mathbb{N}_1$

axm_CRIT_DIST2: $\forall v0 \cdot (v0 \in 2..VEHICLES \Rightarrow dist(initial\_pos(v0-1) \mapsto initial\_pos(v0)) > CRITICAL\_DISTANCE)$

axm_y_dist1: $y\_dist \in Point \times Point \rightarrow \mathbb{N}$

axm_y_dist2: $\forall x0 \cdot (x0 \in Point \Rightarrow y\_dist(x0 \mapsto x0) = 0)$

axm_y_dist3: $\forall x0, y0 \cdot (x0 \in Point \land y0 \in Point \Rightarrow y\_dist(x0 \mapsto y0) = y\_dist(y0 \mapsto x0))$

axm_y_dist4: $\forall x0, y0, z0 \cdot (x0 \in Point \land y0 \in Point \land z0 \in Point \Rightarrow y\_dist(x0 \mapsto y0) + y\_dist(y0 \mapsto z0) \geq y\_dist(x0 \mapsto z0))$

axm_nearest: $nearest \in Point \times (\mathbb{N} \rightarrow (1..VEHICLES \rightarrow Point)) \rightarrow Point$

axm_init_traj: $initial\_trajectory \in 0..0 \rightarrow (1..VEHICLES \rightarrow Point)$

axm_Y_DIST1: $Y\_DISTANCE \in \mathbb{N}_1$

axm_Y_DIST2: $\forall v0 \cdot (v0 \in 2..VEHICLES \Rightarrow y\_dist(initial\_pos(v0) \mapsto nearest(initial\_pos(v0) \mapsto initial\_trajectory)) < Y\_DISTANCE)$

END

E.2 Context1

CONTEXT context1
EXTENDS context0
END

E.3 Context2

CONTEXT context2
EXTENDS context1
CONSTANTS

MAX_SPEED
MAX_ACCEL
MIN_ACCEL
MAX_κ
MAX_χ
x
y
$\gamma\theta$
$\sigma\theta$
v
κ
μ
β
μc
μs
Fc
Fs
new_point
$new\_pos\_v\_\kappa$
$new\_pos\_v\_\kappa max$
$new\_pos\_v\_\kappa min$
$new\_pos\_vmax\_\kappa$
$new\_pos\_vmax\_\kappa max$
$new\_pos\_vmax\_\kappa min$

new_pos_vmin_κ
new_pos_vmin_κmax
new_pos_vmin_κmin

AXIOMS

axm_ACCEL1 : $MAX\_ACCEL \in \mathbb{N}_1$
axm_ACCEL2 : $MIN\_ACCEL \in \mathbb{Z}$
axm_ACCEL3 : $MIN\_ACCEL < 0$
axm_χ : $MAX\_\chi \in \mathbb{N}_1$
axm_x : $x \in Point \to \mathbb{Z}$
axm_y : $y \in Point \to \mathbb{Z}$
axm_γθ : $\gamma\theta \in Point \to \mathbb{Z}$
axm_σθ : $\sigma\theta \in Point \to \mathbb{Z}$
axm_v1 : $MAX\_SPEED \in \mathbb{N}_1$
axm_v2 : $v \in Point \to 0\,..\,MAX\_SPEED$
axm_κ1 : $MAX\_\kappa \in \mathbb{N}_1$
axm_κ2 : $\kappa \in Point \to -MAX\_\kappa\,..\,MAX\_\kappa$
axm_β1 : $\beta \in Point \times MIN\_ACCEL\,..\,MAX\_ACCEL \times -MAX\_\chi\,..\,MAX\_\chi \to \mathbb{Z}$
axm_β2 : $\forall p,a,\chi \cdot ($
$p \in Point \land$
$a \in MIN\_ACCEL\,..\,MAX\_ACCEL \land$
$\chi \in -MAX\_\chi\,..\,MAX\_\chi$
$\Rightarrow$
$\beta(p \mapsto a \mapsto \chi) = v(p) * \kappa(p) + (a * \kappa(p) + v(p) * \chi)/2 + a * \chi/3$
$)$
axm_μ1 : $\mu \in \mathbb{N}_1$
axm_μ2 : $2 * \mu \neq 0$
axm_μ3 : $6 * \mu * \mu \neq 0$
axm_μ4 : $5040 * \mu \neq 0$
axm_μ5 : $3991680 * \mu * \mu \neq 0$
axm_μ6 : $\mu * \mu \neq 0$
axm_μ7 : $\mu \neq 0$
axm_μc1 : $\mu c \in Point \times MIN\_ACCEL\,..\,MAX\_ACCEL \times -MAX\_\chi\,..\,MAX\_\chi \to \mathbb{Z}$
axm_μc2 : $\forall p,a,\chi \cdot ($
$p \in Point \land$
$a \in MIN\_ACCEL\,..\,MAX\_ACCEL \land$
$\chi \in -MAX\_\chi\,..\,MAX\_\chi$
$\Rightarrow$
$\mu c(p \mapsto a \mapsto \chi) = \mu - (\beta(p \mapsto a \mapsto \chi) * \beta(p \mapsto a \mapsto \chi))/(2 * \mu)$
$)$
axm_μs1 : $\mu s \in Point \times MIN\_ACCEL\,..\,MAX\_ACCEL \times -MAX\_\chi\,..\,MAX\_\chi \to \mathbb{Z}$
axm_μs2 : $\forall p,a,\chi \cdot ($
$p \in Point \land$
$a \in MIN\_ACCEL\,..\,MAX\_ACCEL \land$
$\chi \in -MAX\_\chi\,..\,MAX\_\chi$
$\Rightarrow$
$\mu s(p \mapsto a \mapsto \chi) = \beta(p \mapsto a \mapsto \chi) - (\beta(p \mapsto a \mapsto \chi) * \beta(p \mapsto a \mapsto \chi) * \beta(p \mapsto a \mapsto \chi))/(6 * \mu * \mu)$
$)$
axm_Fc1 : $Fc \in Point \times MIN\_ACCEL\,..\,MAX\_ACCEL \times -MAX\_\chi\,..\,MAX\_\chi \to \mathbb{Z}$
axm_Fc2 : $\forall p,a,\chi \cdot ($
$p \in Point \land$
$a \in MIN\_ACCEL\,..\,MAX\_ACCEL \land$
$\chi \in -MAX\_\chi\,..\,MAX\_\chi$
$\Rightarrow$
$Fc(p \mapsto a \mapsto \chi) = v(p) * \mu + (a * \mu)/2$
$- (840 * v(p) * v(p) * v(p) * \kappa(p) * \kappa(p)$
$+ 630 * v(p) * v(p) * \kappa(p) * (2 * a * \kappa(p) + v(p) * \chi)$
$+ 42 * v(p) * (15 * a * a * \kappa(p) * \kappa(p) + 3 * v(p) * v(p) * \chi * \chi + 26 * v(p) * \kappa(p) * a * \chi)$
$+ 35 * a * (3 * a * a * \kappa(p) * \kappa(p) + 7 * v(p) * v(p) * \chi * \chi + 18 * v(p) * \kappa(p) * a * \chi)$
$+ 40 * a * a * \chi * (3 * a * \kappa(p) + 4 * v(p) * \chi)$
$+ 35 * a * a * a * \chi * \chi$
$)/(5040 * \mu)$
$)$

axm_Fs1: $Fs \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to \mathbb{Z}$

axm_Fs2: $\forall p,a,\chi \cdot ($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi$
$\Rightarrow$
$Fs(p \mapsto a \mapsto \chi) = (60*v(p)*v(p)*\kappa(p)+20*v(p)*(3*a*\kappa(p)+v(p)*\chi)$
$+5*a*(3*a*\kappa(p)+5*v(p)*\chi)+8*a*a*\chi)/120$
$-(166320*v(p)*v(p)*v(p)*v(p)*\kappa(p)*\kappa(p)*\kappa(p)$
$+66528*v(p)*v(p)*v(p)*\kappa(p)*\kappa(p)*(5*a*\kappa(p)+3*v(p)*\chi)$
$+27720*v(p)*v(p)*\kappa(p)*(9*a*a*\kappa(p)*\kappa(p)$
$+16*v(p)*\kappa(p)*a*\chi+3*v(p)*v(p)*\chi*\chi)$
$+11880*v(p)*(7*a*a*a*\kappa(p)*\kappa(p)*\kappa(p)+31*v(p)*\kappa(p)*\kappa(p)*a*a*\chi$
$+17*v(p)*v(p)*\kappa(p)*a*\chi*\chi+v(p)*v(p)*v(p)*\chi*\chi*\chi)$
$+3465*a*(3*a*a*a*a*\kappa(p)*\kappa(p)*\kappa(p)+39*v(p)*\kappa(p)*\kappa(p)*a*a*\chi$
$+53*v(p)*v(p)*\kappa(p)*a*\chi*\chi+9*v(p)*v(p)*v(p)*\chi*\chi*\chi)$
$+6160*a*a*a*\chi*(3*a*a*\kappa(p)*\kappa(p)+5*v(p)*v(p)*\chi*\chi+12*a*v(p)*\kappa(p)*\chi)$
$+1232*a*a*a*a*\chi*\chi*(9*a*\kappa(p)+11*v(p)*\chi)$
$+2240*a*a*a*a*a*\chi*\chi*\chi$
$)/(3991680*\mu*\mu)$
$)$

axm_new_point: $new\_point \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times 0..MAX\_SPEED \times -MAX\_\kappa..MAX\_\kappa \to Point$

axm_new_pos_v_κ1: $new\_pos\_v\_\kappa \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_v_κ2: $\forall p,a,\chi \cdot ($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi \wedge$
$v(p)+a \in 0..MAX\_SPEED \wedge$
$\kappa(p)+\chi \in -MAX\_\kappa..MAX\_\kappa$
$\Rightarrow$
$new\_pos\_v\_\kappa(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto a \mapsto \chi)-\sigma\theta(p)*Fs(p \mapsto a \mapsto \chi))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto a \mapsto \chi)+\sigma\theta(p)*Fc(p \mapsto a \mapsto \chi))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto a \mapsto \chi)*\gamma\theta(p)-\mu s(p \mapsto a \mapsto \chi)*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto a \mapsto \chi)*\sigma\theta(p)+\mu s(p \mapsto a \mapsto \chi)*\gamma\theta(p))/\mu) \mapsto$
$(v(p)+a) \mapsto$
$(\kappa(p)+\chi)$
$)$
$)$

axm_new_pos_v_κmax1: $new\_pos\_v\_\kappa max \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_v_κmax2: $\forall p,a,\chi \cdot ($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi \wedge$
$v(p)+a \in 0..MAX\_SPEED \wedge$
$\kappa(p)+\chi > MAX\_\kappa$
$\Rightarrow$
$new\_pos\_v\_\kappa max(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))-\sigma\theta(p)*Fs(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))+\sigma\theta(p)*Fc(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p)-\mu s(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p)+\mu s(p \mapsto a \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p))/\mu) \mapsto$
$(v(p)+a) \mapsto$
(MAX_κ)
$)$
$)$

axm_new_pos_v_κmin1: $new\_pos\_v\_\kappa min \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_v_κmin2 : $\forall p,a,\chi\cdot($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi \wedge$
$v(p)+a \in 0..MAX\_SPEED \wedge$
$\kappa(p)+\chi < -MAX\_\kappa$
$\Rightarrow$
$new\_pos\_v\_\kappa min(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))-\sigma\theta(p)*Fs(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))+\sigma\theta(p)*Fc(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))*\gamma\theta(p)-\mu s(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))*\sigma\theta(p)+\mu s(p \mapsto a \mapsto (-MAX\_\kappa-\kappa(p)))*\gamma\theta(p))/\mu) \mapsto$
$(v(p)+a) \mapsto$
$(-MAX\_\kappa)$
$)$
$)$

axm_new_pos_vmax_κ1 : $new\_pos\_vmax\_\kappa \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \rightarrow Point$

axm_new_pos_vmax_κ2 : $\forall p,a,\chi\cdot($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi \wedge$
$v(p)+a > MAX\_SPEED \wedge$
$\kappa(p)+\chi \in -MAX\_\kappa..MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmax\_\kappa(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)-\sigma\theta(p)*$
$Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)+\sigma\theta(p)*$
$Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)*\gamma\theta(p)-\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)*\sigma\theta(p)+\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto \chi)*\gamma\theta(p))/\mu) \mapsto$
$(MAX\_SPEED) \mapsto$
$(\kappa(p)+\chi)$
$)$
$)$

axm_new_pos_vmax_κmax1 : $new\_pos\_vmax\_\kappa max \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \rightarrow Point$

axm_new_pos_vmax_κmax2 : $\forall p,a,\chi\cdot($
$p \in Point \wedge$
$a \in MIN\_ACCEL..MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi..MAX\_\chi \wedge$
$v(p)+a > MAX\_SPEED \wedge$
$\kappa(p)+\chi > MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmax\_\kappa max(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))-\sigma\theta(p)*$
$Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))+\sigma\theta(p)*$
$Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p)-$
$\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p)+$
$\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p))/\mu) \mapsto$
$(MAX\_SPEED) \mapsto$
(MAX_κ)
$)$
$)$

axm_new_pos_vmax_κmin1 : $new\_pos\_vmax\_\kappa min \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \rightarrow Point$

axm_new_pos_vmax_κmin2 : $\forall p,a,\chi\cdot($
$p \in Point \land$
$a \in MIN\_ACCEL..MAX\_ACCEL \land$
$\chi \in -MAX\_\chi..MAX\_\chi \land$
$v(p)+a > MAX\_SPEED \land$
$\kappa(p)+\chi < -MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmax\_\kappa min(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))-\sigma\theta(p)*$
$Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))+\sigma\theta(p)*$
$Fc(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))*\gamma\theta(p)-$
$\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))*\sigma\theta(p)+$
$\mu s(p \mapsto (MAX\_SPEED-v(p)) \mapsto (-MAX\_\kappa-\kappa(p)))*\gamma\theta(p))/\mu) \mapsto$
$(MAX\_SPEED) \mapsto$
$(-MAX\_\kappa)$
$)$
$)$

axm_new_pos_vmin_κ1 : $new\_pos\_vmin\_\kappa \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_vmin_κ2 : $\forall p,a,\chi\cdot($
$p \in Point \land$
$a \in MIN\_ACCEL..MAX\_ACCEL \land$
$\chi \in -MAX\_\chi..MAX\_\chi \land$
$v(p)+a < 0 \land$
$\kappa(p)+\chi \in -MAX\_\kappa..MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmin\_\kappa(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto (-v(p)) \mapsto \chi)-\sigma\theta(p)*Fs(p \mapsto (-v(p)) \mapsto \chi))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto (-v(p)) \mapsto \chi)+\sigma\theta(p)*Fc(p \mapsto (-v(p)) \mapsto \chi))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto \chi)*\gamma\theta(p)-\mu s(p \mapsto (-v(p)) \mapsto \chi)*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto \chi)*\sigma\theta(p)+\mu s(p \mapsto (-v(p)) \mapsto \chi)*\gamma\theta(p))/\mu) \mapsto$
$(0) \mapsto$
$(\kappa(p)+\chi)$
$)$
$)$

axm_new_pos_vmin_κmax1 : $new\_pos\_vmin\_\kappa max \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_vmin_κmax2 : $\forall p,a,\chi\cdot($
$p \in Point \land$
$a \in MIN\_ACCEL..MAX\_ACCEL \land$
$\chi \in -MAX\_\chi..MAX\_\chi \land$
$v(p)+a < 0 \land$
$\kappa(p)+\chi > MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmin\_\kappa max(p \mapsto a \mapsto \chi) = new\_point($
$(x(p)+(\gamma\theta(p)*Fc(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))-\sigma\theta(p)*$
$Fs(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$(y(p)+(\gamma\theta(p)*Fs(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))+\sigma\theta(p)*$
$Fc(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p))))/(\mu*\mu)) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p)-\mu s(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\sigma\theta(p)+\mu s(p \mapsto (-v(p)) \mapsto (MAX\_\kappa-\kappa(p)))*\gamma\theta(p))/\mu) \mapsto$
$(0) \mapsto$
(MAX_κ)
$)$
$)$

axm_new_pos_vmin_κmin1 : $new\_pos\_vmin\_\kappa min \in Point \times MIN\_ACCEL..MAX\_ACCEL \times -MAX\_\chi..MAX\_\chi \to Point$

axm_new_pos_vmin_κmin2 : $\forall p, a, \chi \cdot ($
$p \in Point \wedge$
$a \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL \wedge$
$\chi \in -MAX\_\chi \mathinner{..} MAX\_\chi \wedge$
$v(p) + a < 0 \wedge$
$\kappa(p) + \chi < -MAX\_\kappa$
$\Rightarrow$
$new\_pos\_vmin\_\kappa min(p \mapsto a \mapsto \chi) = new\_point($
$(x(p) + (\gamma\theta(p) * Fc(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) - \sigma\theta(p) *$
$Fs(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))))/(\mu * \mu)) \mapsto$
$(y(p) + (\gamma\theta(p) * Fs(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) + \sigma\theta(p) *$
$Fc(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))))/(\mu * \mu)) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) * \gamma\theta(p) - \mu s(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) * \sigma\theta(p))/\mu) \mapsto$
$((\mu c(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) * \sigma\theta(p) + \mu s(p \mapsto (-v(p)) \mapsto (-MAX\_\kappa - \kappa(p))) * \gamma\theta(p))/\mu) \mapsto$
$(0) \mapsto$
$(-MAX\_\kappa)$
$)$
$)$
axm_init_pos : $\forall vehi \cdot (vehi \in 1 \mathinner{..} VEHICLES \Rightarrow$
$\exists x0, y0, \gamma\theta 0, \sigma\theta 0, v0, \kappa 0 \cdot ($
$x0 \in \mathbb{Z} \wedge y0 \in \mathbb{Z} \wedge \gamma\theta 0 \in \mathbb{Z} \wedge \sigma\theta 0 \in \mathbb{Z} \wedge v0 \in 0 \mathinner{..} MAX\_SPEED \wedge \kappa 0 \in -MAX\_\kappa \mathinner{..} MAX\_\kappa \wedge$
$initial\_pos(vehi) = new\_point(x0 \mapsto y0 \mapsto \gamma\theta 0 \mapsto \sigma\theta 0 \mapsto v0 \mapsto \kappa 0)$
$)$
$)$
END

E.4 Context3

CONTEXT context3
EXTENDS context2
CONSTANTS
initial_accel
initial_chi
AXIOMS
axm_init_accel1 : $initial\_accel \in 1 \mathinner{..} VEHICLES \rightarrow MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
axm_init_accel2 : $\forall v0 \cdot ($
$v0 \in 1 \mathinner{..} VEHICLES$
$\Rightarrow$
$(\exists a0 \cdot (a0 \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL \wedge initial\_accel(v0) = a0))$
$)$
axm_init_chi1 : $initial\_chi \in 1 \mathinner{..} VEHICLES \rightarrow -MAX\_\chi \mathinner{..} MAX\_\chi$
axm_init_chi2 : $\forall v0 \cdot ($
$v0 \in 1 \mathinner{..} VEHICLES$
$\Rightarrow$
$(\exists c0 \cdot (c0 \in -MAX\_\chi \mathinner{..} MAX\_\chi \wedge initial\_chi(v0) = c0))$
$)$
END

E.5 Context4

CONTEXT context4
EXTENDS context3
CONSTANTS
IDEAL_SPEED
IDEAL_κ
initial_v
initial_κ
initial_pre_v
h

k
$initial\_\Delta x$
$initial\_\Delta y$
$initial\_\Delta\gamma$
$initial\_\Delta\sigma$
new_accel
new_chi

AXIOMS

axm_IDEAL_SPEED1 : $IDEAL\_SPEED \in 0..MAX\_SPEED$
axm_IDEAL_SPEED2 : $IDEAL\_SPEED > 0$
axm_IDEAL_SPEED3 : $IDEAL\_SPEED < MAX\_SPEED$
axm_IDEAL_κ1 : $IDEAL\_\kappa \in -MAX\_\kappa..MAX\_\kappa$
axm_IDEAL_κ2 : $IDEAL\_\kappa < MAX\_\kappa$
axm_IDEAL_κ3 : $IDEAL\_\kappa > -MAX\_\kappa$
axm_initial_v1 : $initial\_v \in 1..VEHICLES \to 0..MAX\_SPEED$
axm_initial_v2 : $\forall vehi \cdot ($
$vehi \in 1..VEHICLES \Rightarrow initial\_v(vehi) = v(initial\_pos(vehi))$
$)$
axm_initial_κ1 : $initial\_\kappa \in 1..VEHICLES \to -MAX\_\kappa..MAX\_\kappa$
axm_initial_κ2 : $\forall vehi \cdot ($
$vehi \in 1..VEHICLES \Rightarrow initial\_\kappa(vehi) = \kappa(initial\_pos(vehi))$
$)$
axm_initial_pre_v : $initial\_pre\_v \in 2..VEHICLES \to 0..MAX\_SPEED$
axm_initial_pre_v2 : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES \Rightarrow initial\_pre\_v(vehi) = v(initial\_pos(vehi-1))$
$)$
axm_initial_Δx1 : $initial\_\Delta x \in 2..VEHICLES \to \mathbb{Z}$
axm_initial_Δx2 : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES$
$\Rightarrow$
$initial\_\Delta x(vehi) =$
$(x(initial\_pos(vehi-1)) - x(initial\_pos(vehi))) * \gamma\theta(initial\_pos(vehi)) +$
$(y(initial\_pos(vehi-1)) - y(initial\_pos(vehi))) * \sigma\theta(initial\_pos(vehi))$
$)$
axm_initial_Δy1 : $initial\_\Delta y \in 2..VEHICLES \to \mathbb{Z}$
axm_initial_Δy2 : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES$
$\Rightarrow$
$initial\_\Delta y(vehi) =$
$-(x(initial\_pos(vehi-1)) - x(initial\_pos(vehi))) * \sigma\theta(initial\_pos(vehi)) +$
$(y(initial\_pos(vehi-1)) - y(initial\_pos(vehi))) * \gamma\theta(initial\_pos(vehi))$
$)$
axm_initial_Δγ1 : $initial\_\Delta\gamma \in 2..VEHICLES \to \mathbb{Z}$
axm_initial_Δγ2 : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES$
$\Rightarrow$
$initial\_\Delta\gamma(vehi) = \gamma\theta(initial\_pos(vehi-1)) * \gamma\theta(initial\_pos(vehi))$
$+ \sigma\theta(initial\_pos(vehi-1)) * \sigma\theta(initial\_pos(vehi))$
$)$
axm_initial_Δσ1 : $initial\_\Delta\sigma \in 2..VEHICLES \to \mathbb{Z}$
axm_initial_Δσ2 : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES$
$\Rightarrow$
$initial\_\Delta\sigma(vehi) = \gamma\theta(initial\_pos(vehi-1)) * \sigma\theta(initial\_pos(vehi))$
$- \sigma\theta(initial\_pos(vehi-1)) * \gamma\theta(initial\_pos(vehi))$
$)$
axm_h : $h \in \mathbb{N}_1$
axm_k1 : $k \in 0..MAX\_SPEED \to \mathbb{Z}$
axm_k2 : $\forall v1 \cdot ($
$v1 \in 0..MAX\_SPEED \Rightarrow k(v1) = max(\{h, v1/MAX\_ACCEL\})$
$)$
axm_new_accel1 : $new\_accel \in 0..MAX\_SPEED \times 0..MAX\_SPEED \times \mathbb{Z} \to \mathbb{Z}$

axm_new_accel2 : $\forall pre\_v1, v1, \Delta x \cdot ($
$pre\_v1 \in 0..MAX\_SPEED \land v1 \in 0..MAX\_SPEED \land \Delta x \in \mathbb{Z}$
$\Rightarrow$
$new\_accel(pre\_v1 \mapsto v1 \mapsto \Delta x) =$
$(pre\_v1 - v1 + k(v1) * (\Delta x/\mu - h * v1 - CRITICAL\_DISTANCE))/h$
)

axm_new_chi1 : $new\_chi \in 0..MAX\_SPEED \times -MAX\_\kappa..MAX\_\kappa \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

axm_new_chi2 : $\forall v1, \kappa 1, \Delta x, \Delta y, \Delta\gamma, \Delta\sigma \cdot ($
$v1 \in 0..MAX\_SPEED \land \kappa 1 \in -MAX\_\kappa..MAX\_\kappa \land \Delta x \in \mathbb{Z} \land \Delta y \in \mathbb{Z} \land \Delta\gamma \in \mathbb{Z} \land \Delta\sigma \in \mathbb{Z} \land \Delta x * \Delta x * \Delta x * \Delta\gamma \neq 0$
$\Rightarrow$
$new\_chi(v1 \mapsto \kappa 1 \mapsto \Delta x \mapsto \Delta y \mapsto \Delta\gamma \mapsto \Delta\sigma) =$
$6 * v1 * ((4 * \mu * \mu * \Delta y * \Delta\gamma - \mu * \mu * \Delta x * \Delta\sigma - \mu * \kappa 1 * \Delta x * \Delta x * \Delta\gamma)/(\Delta x * \Delta x * \Delta x * \Delta\gamma))$
)

END

E.6 Platoon0

MACHINE platoon0
SEES context0
VARIABLES
$pos0$
$temp$
$trajectory$
INVARIANTS
inv_pos0 : $pos0 \in 1..VEHICLES \to Point$
inv_temp : $temp \in \mathbb{N}$
inv_traj : $trajectory \in 0..temp \to (1..VEHICLES \to Point)$
inv_CRIT_DIST : $\forall v0 \cdot ($
$v0 \in 2..VEHICLES \Rightarrow$
$dist(pos0(v0 - 1) \mapsto pos0(v0)) > CRITICAL\_DISTANCE$
)
inv_Y_DIST : $\forall v0 \cdot ($
$v0 \in 2..VEHICLES \Rightarrow$
$y\_dist(pos0(v0) \mapsto nearest(pos0(v0) \mapsto trajectory)) < Y\_DISTANCE$
)
thm_temp : $temp + 1 \in \mathbb{N}$
EVENTS
Initialisation
begin
act_pos0 : $pos0 := initial\_pos$
act_temp : $temp := 0$
act_traj : $trajectory := initial\_trajectory$
end

Event $all\_moves \mathrel{\widehat{=}}$
any
$magic\_pos$
where
grd_magic_pos : $magic\_pos \in 1..VEHICLES \to Point$
grd_CRIT_DIST : $\forall v0 \cdot ($
$v0 \in 2..VEHICLES \Rightarrow$
$dist(magic\_pos(v0 - 1) \mapsto magic\_pos(v0)) > CRITICAL\_DISTANCE$
)
grd_Y_DIST : $\forall v0 \cdot ($
$v0 \in 2..VEHICLES \Rightarrow$
$y\_dist(magic\_pos(v0) \mapsto nearest(magic\_pos(v0) \mapsto$
$trajectory \lhd\!\!+ \{temp + 1 \mapsto magic\_pos\}))) < Y\_DISTANCE$
)
then

act_pos0 : $pos0 := magic\_pos$
act_temp : $temp := temp + 1$
act_traj : $trajectory := trajectory \Leftarrow \{temp + 1 \mapsto magic\_pos\}$
end
END

E.7 Platoon1

MACHINE platoon1
REFINES platoon0
SEES context1
VARIABLES
$pos0$
$temp$
$trajectory$
$vehicle$
pos
INVARIANTS
inv_vehicle : $vehicle \in 1..VEHICLES + 1$
inv_pos : $pos \in 1..VEHICLES \rightarrow Point$
inv_CRIT_DIST : $\forall v0 \cdot ($
$v0 \in 2..vehicle - 1 \Rightarrow$
$dist(pos(v0 - 1) \mapsto pos(v0)) > CRITICAL\_DISTANCE$
$)$
inv_Y_DIST : $\forall v0 \cdot ($
$v0 \in 2..vehicle - 1 \Rightarrow$
$y\_dist(pos(v0) \mapsto nearest(pos(v0) \mapsto trajectory)) < Y\_DISTANCE$
$)$
thm_vehi : $VEHICLES + 1 - vehicle \in \mathbb{N}$
EVENTS
Initialisation
begin
act_pos0 : $pos0 := initial\_pos$
act_temp : $temp := 0$
act_traj : $trajectory := initial\_trajectory$
act_vehi : $vehicle := 1$
act_pos : $pos := initial\_pos$
end

Event *move1* $\widehat{=}$
Status convergent
any
magic_pos_vehicle
where
grd_vehi : $vehicle = 1$
grd_magic_pos : $magic\_pos\_vehicle \in Point$
then
act_pos : $pos(vehicle) := magic\_pos\_vehicle$
act_vehi : $vehicle := vehicle + 1$
end

Event *move* $\widehat{=}$
Status convergent
any
magic_pos_vehicle
where
grd_vehi : $vehicle \in 2..VEHICLES$

grd_magic_pos : $magic\_pos\_vehicle \in Point$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto magic\_pos\_vehicle) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(magic\_pos\_vehicle \mapsto nearest(magic\_pos\_vehicle \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := magic\_pos\_vehicle$
act_vehi : $vehicle := vehicle + 1$

end

Event $all\_moves \mathrel{\widehat{=}}$
refines all_moves

when

grd_vehi : $vehicle = VEHICLES + 1$
grd_Y_DIST : $\forall v0 \cdot (v0 \in 2..VEHICLES \Rightarrow y\_dist(pos(v0) \mapsto nearest(pos(v0) \mapsto trajectory \Leftarrow \{temp+1 \mapsto pos\})) < Y\_DISTANCE)$

with

magic_pos : $magic\_pos = pos$

then

act_pos0 : $pos0 := pos$
act_temp : $temp := temp + 1$
act_traj : $trajectory := trajectory \Leftarrow \{temp+1 \mapsto pos\}$
act_vehi : $vehicle := 1$

end

VARIANT
$(VEHICLES+1) - vehicle$
END

E.8 Platoon2

MACHINE platoon2
REFINES platoon1
SEES context2
VARIABLES
$pos0$
$temp$
$trajectory$
$vehicle$
pos
INVARIANTS
thm_dom_pos1 : $1 \in dom(pos)$
thm_dom_pos2 : $vehicle \in 2..VEHICLES \Rightarrow vehicle \in dom(pos)$
thm_dom_pos3 : $vehicle \in 2..VEHICLES \Rightarrow vehicle - 1 \in dom(pos)$
thm_dom_v1 : $pos(1) \in dom(v)$
thm_dom_v2 : $vehicle \in 2..VEHICLES \Rightarrow pos(vehicle) \in dom(v)$
thm_dom_κ1 : $pos(1) \in dom(\kappa)$
thm_dom_κ2 : $vehicle \in 2..VEHICLES \Rightarrow pos(vehicle) \in dom(\kappa)$
EVENTS
Initialisation
begin

act_pos0 : $pos0 := initial\_pos$
act_temp : $temp := 0$
act_traj : $trajectory := initial\_trajectory$
act_vehi : $vehicle := 1$
act_pos : $pos := initial\_pos$

end

Event *move1_v_κ* $\widehat{=}$
Status convergent
refines *move1*
any
magic_accel
magic_chi
nspeed
nkappa
npos
where
`grd_vehi:` $vehicle = 1$
`grd_magic_accel:` $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
`grd_nspeed1:` $nspeed = v(pos(vehicle)) + magic\_accel$
`grd_nspeed2:` $nspeed \in 0..MAX\_SPEED$
`grd_magic_chi:` $magic\_chi \in -MAX\_\chi..MAX\_\chi$
`grd_nkappa1:` $nkappa = \kappa(pos(vehicle)) + magic\_chi$
`grd_nkappa2:` $nkappa \in -MAX\_\kappa..MAX\_\kappa$
`grd_dom_new_pos:` $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa)$
`grd_npos:` $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
with
`magic_pos_vehicle:` $magic\_pos\_vehicle = npos$
then
`act_pos:` $pos(vehicle) := npos$
`act_vehi:` $vehicle := vehicle + 1$
end

Event *move1_v_κmax* $\widehat{=}$
Status convergent
refines *move1*
any
magic_accel
magic_chi
nspeed
nkappa
npos
where
`grd_vehi:` $vehicle = 1$
`grd_magic_accel:` $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
`grd_nspeed1:` $nspeed = v(pos(vehicle)) + magic\_accel$
`grd_nspeed2:` $nspeed \in 0..MAX\_SPEED$
`grd_magic_chi:` $magic\_chi \in -MAX\_\chi..MAX\_\chi$
`grd_nkappa1:` $nkappa = \kappa(pos(vehicle)) + magic\_chi$
`grd_nkappa2:` $nkappa > MAX\_\kappa$
`grd_dom_new_pos:` $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa max)$
`grd_npos:` $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
with
`magic_pos_vehicle:` $magic\_pos\_vehicle = npos$
then
`act_pos:` $pos(vehicle) := npos$
`act_vehi:` $vehicle := vehicle + 1$
end

Event *move1_v_κmin* $\widehat{=}$
Status convergent
refines *move1*
any

any
magic_accel
magic_chi
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move1_vmax_κ* $\widehat{=}$
Status convergent
refines *move1*
any
magic_accel
magic_chi
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move1_vmax_κmax* $\widehat{=}$
Status convergent
refines *move1*
any
magic_accel
magic_chi
nspeed
nkappa
npos
where

grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmax_κmin* $\widehat{=}$
Status convergent
refines *move1*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa min)$
grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κ* $\widehat{=}$
Status convergent
refines *move1*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$

grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κmax* $\widehat{=}$
Status convergent
refines *move1*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi \mathinner{..} MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κmin* $\widehat{=}$
Status convergent
refines *move1*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi \mathinner{..} MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa min)$
grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_v\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines $move$
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa)$
grd_npos : $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_v\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines $move$
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa max)$
grd_npos : $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_v\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines *move*
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_vmax\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines *move*
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_vmax\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines *move*
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_vmax\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines *move*
any
$magic\_accel$
$magic\_chi$
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmax\_\kappa min)$
grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_pos_vehicle : $magic\_pos\_vehicle = npos$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move\_vmin\_\kappa \mathrel{\widehat{=}}$
Status convergent

refines *move*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmin_κmax* $\widehat{=}$
Status convergent
refines *move*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmin_κmin* $\widehat{=}$
Status convergent
refines *move*

any

magic_accel
magic_chi
nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_magic_accel : $magic\_accel \in MIN\_ACCEL..MAX\_ACCEL$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + magic\_accel$
grd_nspeed2 : $nspeed < 0$
grd_magic_chi : $magic\_chi \in -MAX\_\chi..MAX\_\chi$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + magic\_chi$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi \in dom(new\_pos\_vmin\_\kappa min)$
grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto magic\_accel \mapsto magic\_chi)$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_pos_vehicle : $magic\_pos\_vehicle = npos$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *all_moves* $\widehat{=}$
refines *all_moves*

when

grd_vehi : $vehicle = VEHICLES + 1$
grd_Y_DIST : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES \Rightarrow$
$y\_dist(pos(vehi) \mapsto nearest(pos(vehi) \mapsto$
$trajectory \Leftarrow \{temp+1 \mapsto pos\})) < Y\_DISTANCE$
$)$

then

act_pos0 : $pos0 := pos$
act_temp : $temp := temp + 1$
act_traj : $trajectory := trajectory \Leftarrow \{temp+1 \mapsto pos\}$
act_vehi : $vehicle := 1$

end

END

E.9 Platoon3

MACHINE platoon3
REFINES platoon2
SEES context3
VARIABLES

pos0
temp
trajectory
vehicle
pos
d_vehicle
accel
chi

INVARIANTS

inv_d_vehi : $d\_vehicle \in 1..VEHICLES + 1$

inv_accel : $accel \in 1..VEHICLES \to MIN\_ACCEL..MAX\_ACCEL$

inv_chi : $chi \in 1..VEHICLES \to -MAX\_\chi..MAX\_\chi$

inv_CRIT_DIST : $(d\_vehicle = VEHICLES + 1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 2..d\_vehicle - 1$
$\Rightarrow$
$(\exists g1, g2 \cdot ($
$g1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \land$
$g2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \land$
$(dist(g1(pos(vehi-1) \mapsto accel(vehi-1) \mapsto chi(vehi-1)) \mapsto$
$g2(pos(vehi) \mapsto accel(vehi) \mapsto chi(vehi))) > CRITICAL\_DISTANCE)$
$)$
$)$
$)$
$)$

inv_Y_DIST : $(d\_vehicle = VEHICLES + 1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 2..d\_vehicle - 1$
$\Rightarrow$
$(\exists g1 \cdot ($
$g1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \land$
$(y\_dist(g1(pos(vehi) \mapsto accel(vehi) \mapsto chi(vehi)) \mapsto$
$nearest(g1(pos(vehi) \mapsto accel(vehi) \mapsto chi(vehi)) \mapsto trajectory)) < Y\_DISTANCE)$
$)$
$)$
$)$
$)$

thm_dom_accel1 : $1 \in dom(accel)$

thm_dom_accel2 : $vehicle \in 2..VEHICLES \Rightarrow vehicle \in dom(accel)$

thm_dom_chi1 : $1 \in dom(chi)$

thm_dom_chi2 : $vehicle \in 2..VEHICLES \Rightarrow vehicle \in dom(chi)$

thm_accel1 : $accel(1) \in MIN\_ACCEL..MAX\_ACCEL$

thm_accel2 : $vehicle \in 2..VEHICLES \Rightarrow accel(vehicle) \in MIN\_ACCEL..MAX\_ACCEL$

thm_chi1 : $chi(1) \in -MAX\_\chi..MAX\_\chi$

thm_chi2 : $vehicle \in 2..VEHICLES \Rightarrow chi(vehicle) \in -MAX\_\chi..MAX\_\chi$

thm_d_vehi : $VEHICLES + 1 - d\_vehicle \in \mathbb{N}$

EVENTS

Initialisation

begin

act_pos0 : $pos0 := initial\_pos$

act_temp : $temp := 0$

act_traj : $trajectory := initial\_trajectory$

act_vehi : $vehicle := 1$

act_pos : $pos := initial\_pos$

act_d_vehi : $d\_vehicle := 1$

act_accel : $accel := initial\_accel$

act_chi : $chi := initial\_chi$

end

Event $decide1 \mathrel{\widehat{=}}$

Status convergent

any

$magic\_accel$

magic_chi

where

grd_vehi : $vehicle = 1$

grd_d_vehi : $d\_vehicle = 1$

grd_magic_accel : $magic\_accel \in MIN\_ACCEL\,..\,MAX\_ACCEL$

grd_magic_chi : $magic\_chi \in -MAX\_\chi\,..\,MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := magic\_accel$

act_chi : $chi(d\_vehicle) := magic\_chi$

act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *decide* $\widehat{=}$

Status convergent

any

magic_accel

magic_chi

where

grd_vehi : $vehicle = 1$

grd_d_vehi : $d\_vehicle \in 2\,..\,VEHICLES$

grd_magic_accel : $magic\_accel \in MIN\_ACCEL\,..\,MAX\_ACCEL$

grd_magic_chi : $magic\_chi \in -MAX\_\chi\,..\,MAX\_\chi$

grd_CRIT_DIST : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto magic\_accel \mapsto magic\_chi)) > CRITICAL\_DISTANCE)$
$)$

grd_Y_DIST : $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto magic\_accel \mapsto magic\_chi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto magic\_accel \mapsto magic\_chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

then

act_accel : $accel(d\_vehicle) := magic\_accel$

act_chi : $chi(d\_vehicle) := magic\_chi$

act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *move1_v_κ* $\widehat{=}$

Status convergent

refines *move1_v_κ*

any

nspeed

nkappa

npos

where

grd_vehi : $vehicle = 1$

grd_d_vehi : $d\_vehicle = VEHICLES + 1$

grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$

grd_nspeed2 : $nspeed \in 0\,..\,MAX\_SPEED$

grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$

grd_nkappa2 : $nkappa \in -MAX\_\kappa\,..\,MAX\_\kappa$

grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa)$
grd_npos : $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move1\_v\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines $move1\_v\_\kappa max$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0 \mathinner{..} MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa max)$
grd_npos : $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move1\_v\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines $move1\_v\_\kappa min$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0 \mathinner{..} MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move1\_vmax\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines $move1\_vmax\_\kappa$

any
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
with
magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move1\_vmax\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines $move1\_vmax\_\kappa max$

any
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
with
magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $move1\_vmax\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines $move1\_vmax\_\kappa min$

any
$nspeed$
$nkappa$
$npos$
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$

grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa min)$
grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κ* $\widehat{=}$
Status convergent
refines *move1_vmin_κ*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa .. MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κmax* $\widehat{=}$
Status convergent
refines *move1_vmin_κmax*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move1_vmin_κmin* $\widehat{=}$
Status convergent
refines *move1_vmin_κmin*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa min)$
grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
with
magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move_v_κ* $\widehat{=}$
Status convergent
refines *move_v_κ*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa)$
grd_npos : $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle - 1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
with
magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move_v_κmax* $\widehat{=}$
Status convergent
refines *move_v_κmax*
any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa max)$
grd_npos : $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_v_κmin* $\widehat{=}$
Status convergent
refines *move_v_κmin*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmax_κ* $\widehat{=}$
Status convergent
refines *move_vmax_κ*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$

grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmax_κmax* $\widehat{=}$
Status convergent
refines *move_vmax_κmax*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2 \mathinner{..} VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmax_κmin* $\widehat{=}$
Status convergent
refines *move_vmax_κmin*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2 \mathinner{..} VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa min)$

grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_vmin\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines $move\_vmin\_\kappa$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$
magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_vmin\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines $move\_vmin\_\kappa max$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$

magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$

act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmin_κmin* $\widehat{=}$

Status convergent

refines *move_vmin_κmin*

any

$nspeed$

$nkappa$

$npos$

where

grd_vehi : $vehicle \in 2..VEHICLES$

grd_d_vehi : $d\_vehicle = VEHICLES + 1$

grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$

grd_nspeed2 : $nspeed < 0$

grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$

grd_nkappa2 : $nkappa < -MAX\_\kappa$

grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa min)$

grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$

grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

with

magic_accel : $magic\_accel = accel(vehicle)$

magic_chi : $magic\_chi = chi(vehicle)$

then

act_pos : $pos(vehicle) := npos$

act_vehi : $vehicle := vehicle + 1$

end

Event *all_moves* $\widehat{=}$

refines *all_moves*

when

grd_vehi : $vehicle = VEHICLES + 1$

grd_d_vehi : $d\_vehicle = VEHICLES + 1$

grd_Y_DIST : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES \Rightarrow$
$y\_dist(pos(vehi) \mapsto nearest(pos(vehi) \mapsto$
$trajectory \lhd\!\!+ \{temp+1 \mapsto pos\})) < Y\_DISTANCE$
$)$

then

act_pos0 : $pos0 := pos$

act_temp : $temp := temp + 1$

act_traj : $trajectory := trajectory \lhd\!\!+ \{temp+1 \mapsto pos\}$

act_vehi : $vehicle := 1$

act_d_vehi : $d\_vehicle := 1$

end

VARIANT

$(VEHICLES+1) - d\_vehicle$

END

E.10 Platoon4

MACHINE platoon4

REFINES platoon3

SEES context4

VARIABLES

*pos*0
temp
trajectory
vehicle
pos
d_vehicle
accel
chi
p_vehicle
p_v
p_κ
p_pre_v
$p\_\Delta x$
$p\_\Delta y$
$p\_\Delta\gamma$
$p\_\Delta\sigma$

INVARIANTS

inv_p_vehi : $p\_vehicle \in 1..VEHICLES+1$

inv_p_v1 : $p\_v \in 1..VEHICLES \rightarrow 0..MAX\_SPEED$

inv_p_v2 : $(p\_vehicle = VEHICLES+1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 1..p\_vehicle-1 \Rightarrow p\_v(vehi) = v(pos(vehi))$
$))$

inv_p_κ1 : $p\_\kappa \in 1..VEHICLES \rightarrow -MAX\_\kappa..MAX\_\kappa$

inv_p_κ2 : $(p\_vehicle = VEHICLES+1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 1..p\_vehicle-1 \Rightarrow p\_\kappa(vehi) = \kappa(pos(vehi))$
$))$

inv_p_pre_v1 : $p\_pre\_v \in 2..VEHICLES \rightarrow 0..MAX\_SPEED$

inv_p_pre_v2 : $(p\_vehicle = VEHICLES+1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 2..p\_vehicle-1 \Rightarrow p\_pre\_v(vehi) = v(pos(vehi-1))$
$))$

inv_p_Δx1 : $p\_\Delta x \in 2..VEHICLES \rightarrow \mathbb{Z}$

inv_p_Δx2 : $(p\_vehicle = VEHICLES+1)$
$\lor$
$(\forall vehi \cdot ($
$vehi \in 2..p\_vehicle-1$
$\Rightarrow$
$p\_\Delta x(vehi) = (x(pos(vehi-1)) - x(pos(vehi))) * \gamma\theta(pos(vehi))$
$+ (y(pos(vehi-1)) - y(pos(vehi))) * \sigma\theta(pos(vehi))$
$))$

inv_p_Δy1 : $p\_\Delta y \in 2..VEHICLES \rightarrow \mathbb{Z}$

inv_p_Δy2 : $(p\_vehicle = VEHICLES + 1)$
$\vee$
$(\forall vehi \cdot ($
$vehi \in 2 \mathinner{..} p\_vehicle - 1$
$\Rightarrow$
$p\_\Delta y(vehi) = -(x(pos(vehi-1)) - x(pos(vehi))) * \sigma\theta(pos(vehi))$
$+ (y(pos(vehi-1)) - y(pos(vehi))) * \gamma\theta(pos(vehi))$
$)$
$)$

inv_p_Δγ1 : $p\_\Delta\gamma \in 2 \mathinner{..} VEHICLES \rightarrow \mathbb{Z}$

inv_p_Δγ2 : $(p\_vehicle = VEHICLES + 1)$
$\vee$
$(\forall vehi \cdot ($
$vehi \in 2 \mathinner{..} p\_vehicle - 1$
$\Rightarrow$
$p\_\Delta\gamma(vehi) = \gamma\theta(pos(vehi-1)) * \gamma\theta(pos(vehi))$
$+ \sigma\theta(pos(vehi-1)) * \sigma\theta(pos(vehi))$
$)$
$)$

inv_p_Δσ1 : $p\_\Delta\sigma \in 2 \mathinner{..} VEHICLES \rightarrow \mathbb{Z}$

inv_p_Δσ2 : $(p\_vehicle = VEHICLES + 1)$
$\vee$
$(\forall vehi \cdot ($
$vehi \in 2 \mathinner{..} p\_vehicle - 1$
$\Rightarrow$
$p\_\Delta\sigma(vehi) = \gamma\theta(pos(vehi-1)) * \sigma\theta(pos(vehi))$
$- \sigma\theta(pos(vehi-1)) * \gamma\theta(pos(vehi))$
$)$
$)$

thm_p_vehi : $VEHICLES + 1 - p\_vehicle \in \mathbb{N}$

thm_dom_pos1 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow p\_vehicle - 1 \in dom(pos)$

thm_dom_pos2 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow p\_vehicle \in dom(pos)$

thm_dom_x1 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle - 1) \in dom(x)$

thm_dom_x2 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle) \in dom(x)$

thm_dom_y1 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle - 1) \in dom(y)$

thm_dom_y2 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle) \in dom(y)$

thm_dom_γθ1 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle - 1) \in dom(\gamma\theta)$

thm_dom_γθ2 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle) \in dom(\gamma\theta)$

thm_dom_σθ1 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle - 1) \in dom(\sigma\theta)$

thm_dom_σθ2 : $p\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow pos(p\_vehicle) \in dom(\sigma\theta)$

thm_MAX_χ1 : $MAX\_\chi \in -MAX\_\chi \mathinner{..} MAX\_\chi$

thm_MAX_χ2 : $-MAX\_\chi \in -MAX\_\chi \mathinner{..} MAX\_\chi$

thm_MAX_ACCEL : $MAX\_ACCEL \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$

thm_MIN_ACCE : $MIN\_ACCEL \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$

thm_dom_p_v : $1 \in dom(p\_v)$

thm_dom_p_v2 : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_v)$

thm_dom_p_pre_v : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_pre\_v)$

thm_dom_p_κ : $1 \in dom(p\_\kappa)$

thm_dom_p_κ2 : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_\kappa)$

thm_dom_p_Δx : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_\Delta x)$

thm_dom_p_Δy : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_\Delta y)$

thm_dom_p_Δγ : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_\Delta\gamma)$

thm_dom_p_Δσ : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(p\_\Delta\sigma)$

thm_dom_pos3 : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle - 1 \in dom(pos)$

thm_dom_pos4 : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle \in dom(pos)$

thm_dom_accel : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle - 1 \in dom(accel)$

thm_dom_chi : $d\_vehicle \in 2 \mathinner{..} VEHICLES \Rightarrow d\_vehicle - 1 \in dom(chi)$

EVENTS

Initialisation

begin

act_pos0 : $pos0 := initial\_pos$

act_temp : $temp := 0$
act_traj : $trajectory := initial\_trajectory$
act_vehi : $vehicle := 1$
act_pos : $pos := initial\_pos$
act_d_vehi : $d\_vehicle := 1$
act_accel : $accel := initial\_accel$
act_chi : $chi := initial\_chi$
act_p_vehi : $p\_vehicle := 1$
act_p_v : $p\_v := initial\_v$
act_p_pre_v : $p\_pre\_v := initial\_pre\_v$
act_p_κ : $p\_\kappa := initial\_\kappa$
act_p_Δx : $p\_\Delta x := initial\_\Delta x$
act_p_Δy : $p\_\Delta y := initial\_\Delta y$
act_p_Δγ : $p\_\Delta\gamma := initial\_\Delta\gamma$
act_p_Δσ : $p\_\Delta\sigma := initial\_\Delta\sigma$

end

Event *perceive1* $\widehat{=}$
Status convergent
when
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = 1$
then
act_p_v : $p\_v(p\_vehicle) := v(pos(p\_vehicle))$
act_p_κ : $p\_\kappa(p\_vehicle) := \kappa(pos(p\_vehicle))$
act_p_vehi : $p\_vehicle := p\_vehicle + 1$
end

Event *perceive* $\widehat{=}$
Status convergent
when
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle \in 2..VEHICLES$
then
act_p_v : $p\_v(p\_vehicle) := v(pos(p\_vehicle))$
act_p_κ : $p\_\kappa(p\_vehicle) := \kappa(pos(p\_vehicle))$
act_p_pre_v : $p\_pre\_v(p\_vehicle) := v(pos(p\_vehicle-1))$
act_p_Δx : $p\_\Delta x(p\_vehicle) := (x(pos(p\_vehicle-1)) - x(pos(p\_vehicle))) * \gamma\theta(pos(p\_vehicle)) + (y(pos(p\_vehicle-1)) - y(pos(p\_vehicle))) * \sigma\theta(pos(p\_vehicle))$
act_p_Δy : $p\_\Delta y(p\_vehicle) := -(x(pos(p\_vehicle-1)) - x(pos(p\_vehicle))) * \sigma\theta(pos(p\_vehicle)) + (y(pos(p\_vehicle-1)) - y(pos(p\_vehicle))) * \gamma\theta(pos(p\_vehicle))$
act_p_Δγ : $p\_\Delta\gamma(p\_vehicle) := \gamma\theta(pos(p\_vehicle-1)) * \gamma\theta(pos(p\_vehicle)) + \sigma\theta(pos(p\_vehicle-1)) * \sigma\theta(pos(p\_vehicle))$
act_p_Δσ : $p\_\Delta\sigma(p\_vehicle) := \gamma\theta(pos(p\_vehicle-1)) * \sigma\theta(pos(p\_vehicle)) - \sigma\theta(pos(p\_vehicle-1)) * \gamma\theta(pos(p\_vehicle))$
act_p_vehi : $p\_vehicle := p\_vehicle + 1$
end

Event *decide1_a_χ* $\widehat{=}$
Status convergent
refines *decide1*
any
naccel
nchi
where
grd_vehi : $vehicle = 1$

grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi \in -MAX\_\chi \mathinner{..} MAX\_\chi$

with

magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = nchi$

then

act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := nchi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event $decide1\_a\_\chi max \mathrel{\widehat{=}}$
Status convergent
refines *decide1*

any

$naccel$
$nchi$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi > MAX\_\chi$

with

magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event $decide1\_a\_\chi min \mathrel{\widehat{=}}$
Status convergent
refines *decide1*

any

$naccel$
$nchi$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel \in MIN\_ACCEL \mathinner{..} MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi < -MAX\_\chi$

with

magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = -MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := -MAX\_\chi$

act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *decide1_amax_χ* $\widehat{=}$
Status convergent
refines *decide1*

any
naccel
nchi
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel > MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi \in -MAX\_\chi \mathinner{..} MAX\_\chi$
with
magic_accel : $magic\_accel = MAX\_ACCEL$
magic_chi : $magic\_chi = nchi$
then
act_accel : $accel(d\_vehicle) := MAX\_ACCEL$
act_chi : $chi(d\_vehicle) := nchi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$
end

Event *decide1_amax_χmax* $\widehat{=}$
Status convergent
refines *decide1*

any
naccel
nchi
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel > MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi > MAX\_\chi$
with
magic_accel : $magic\_accel = MAX\_ACCEL$
magic_chi : $magic\_chi = MAX\_\chi$
then
act_accel : $accel(d\_vehicle) := MAX\_ACCEL$
act_chi : $chi(d\_vehicle) := MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$
end

Event *decide1_amax_χmin* $\widehat{=}$
Status convergent
refines *decide1*

any
naccel
nchi
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$

grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel > MAX\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi < -MAX\_\chi$

with

magic_accel : $magic\_accel = MAX\_ACCEL$
magic_chi : $magic\_chi = -MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := MAX\_ACCEL$
act_chi : $chi(d\_vehicle) := -MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *decide1_amin_*$\chi \mathrel{\widehat{=}}$
Status convergent
refines *decide1*

any

naccel
nchi

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel < MIN\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi \in -MAX\_\chi \mathinner{..} MAX\_\chi$

with

magic_accel : $magic\_accel = MIN\_ACCEL$
magic_chi : $magic\_chi = nchi$

then

act_accel : $accel(d\_vehicle) := MIN\_ACCEL$
act_chi : $chi(d\_vehicle) := nchi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *decide1_amin_*$\chi max \mathrel{\widehat{=}}$
Status convergent
refines *decide1*

any

naccel
nchi

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel < MIN\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi > MAX\_\chi$

with

magic_accel : $magic\_accel = MIN\_ACCEL$
magic_chi : $magic\_chi = MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := MIN\_ACCEL$
act_chi : $chi(d\_vehicle) := MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event *decide1_amin_χmin* $\widehat{=}$
Status convergent
refines *decide1*
any
naccel
nchi
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_naccel1 : $naccel = IDEAL\_SPEED - p\_v(d\_vehicle)$
grd_naccel2 : $naccel < MIN\_ACCEL$
grd_nchi1 : $nchi = IDEAL\_\kappa - p\_\kappa(d\_vehicle)$
grd_nchi2 : $nchi < -MAX\_\chi$
with
magic_accel : $magic\_accel = MIN\_ACCEL$
magic_chi : $magic\_chi = -MAX\_\chi$
then
act_accel : $accel(d\_vehicle) := MIN\_ACCEL$
act_chi : $chi(d\_vehicle) := -MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$
end

Event *decide_a_χ* $\widehat{=}$
Status convergent
refines *decide*
any
naccel
nchi
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle \in 2..VEHICLES$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel : $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1 : $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2 : $naccel \in MIN\_ACCEL..MAX\_ACCEL$
grd_dom_new_chi : $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1 : $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2 : $nchi \in -MAX\_\chi..MAX\_\chi$
grd_CRIT_DIST : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto f2(pos(d\_vehicle) \mapsto naccel \mapsto nchi)) > CRITICAL\_DISTANCE)$
$)$
grd_Y_DIST : $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto naccel \mapsto nchi) \mapsto nearest(f1(pos(d\_vehicle) \mapsto naccel \mapsto nchi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$
with

magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = nchi$

then

act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := nchi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event $decide\_a\_\chi max \mathrel{\widehat{=}}$
Status convergent
refines $decide$

any

$naccel$
$nchi$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle \in 2..VEHICLES$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel : $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1 : $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2 : $naccel \in MIN\_ACCEL..MAX\_ACCEL$
grd_dom_new_chi : $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1 : $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2 : $nchi > MAX\_\chi$
grd_CRIT_DIST : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto f2(pos(d\_vehicle) \mapsto naccel \mapsto MAX\_\chi)) > CRITICAL\_DISTANCE)$
$)$
grd_Y_DIST : $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto naccel \mapsto MAX\_\chi) \mapsto nearest(f1(pos(d\_vehicle) \mapsto naccel \mapsto MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

with

magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = MAX\_\chi$

then

act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$

end

Event $decide\_a\_\chi min \mathrel{\widehat{=}}$
Status convergent
refines $decide$

any

$naccel$
$nchi$

where

grd_vehi : $vehicle = 1$

grd_d_vehi : $d\_vehicle \in 2..VEHICLES$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel : $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1 : $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2 : $naccel \in MIN\_ACCEL..MAX\_ACCEL$
grd_dom_new_chi : $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1 : $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2 : $nchi < -MAX\_\chi$
grd_CRIT_DIST : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto naccel \mapsto -MAX\_\chi)) > CRITICAL\_DISTANCE)$
)
grd_Y_DIST : $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto naccel \mapsto -MAX\_\chi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto naccel \mapsto -MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
)

with
magic_accel : $magic\_accel = naccel$
magic_chi : $magic\_chi = -MAX\_\chi$
then
act_accel : $accel(d\_vehicle) := naccel$
act_chi : $chi(d\_vehicle) := -MAX\_\chi$
act_d_vehi : $d\_vehicle := d\_vehicle + 1$
end

Event $decide\_amax\_\chi \mathrel{\widehat{=}}$
Status convergent
refines $decide$
any
$naccel$
$nchi$
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle \in 2..VEHICLES$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel : $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1 : $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2 : $naccel > MAX\_ACCEL$
grd_dom_new_chi : $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1 : $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2 : $nchi \in -MAX\_\chi..MAX\_\chi$
grd_CRIT_DIST : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto nchi)) > CRITICAL\_DISTANCE)$
)

grd_Y_DIST: $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto nchi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto nchi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

with
magic_accel: $magic\_accel = MAX\_ACCEL$
magic_chi: $magic\_chi = nchi$
then
act_accel: $accel(d\_vehicle) := MAX\_ACCEL$
act_chi: $chi(d\_vehicle) := nchi$
act_d_vehi: $d\_vehicle := d\_vehicle + 1$
end

Event *decide_amax_χmax* $\widehat{=}$
Status convergent
refines *decide*
any
naccel
nchi
where
grd_vehi: $vehicle = 1$
grd_d_vehi: $d\_vehicle \in 2..VEHICLES$
grd_p_vehi: $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel: $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1: $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2: $naccel > MAX\_ACCEL$
grd_dom_new_chi: $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1: $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2: $nchi > MAX\_\chi$
grd_CRIT_DIST: $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle-1) \mapsto accel(d\_vehicle-1) \mapsto chi(d\_vehicle-1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto MAX\_\chi)) > CRITICAL\_DISTANCE)$
$)$
grd_Y_DIST: $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto MAX\_\chi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

with
magic_accel: $magic\_accel = MAX\_ACCEL$
magic_chi: $magic\_chi = MAX\_\chi$
then
act_accel: $accel(d\_vehicle) := MAX\_ACCEL$
act_chi: $chi(d\_vehicle) := MAX\_\chi$
act_d_vehi: $d\_vehicle := d\_vehicle + 1$
end

Event $decide\_amax\_\chi min \mathrel{\widehat{=}}$
Status convergent
refines *decide*
any
$naccel$
$nchi$
where
grd_vehi: $vehicle = 1$
grd_d_vehi: $d\_vehicle \in 2..VEHICLES$
grd_p_vehi: $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel: $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1: $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2: $naccel > MAX\_ACCEL$
grd_dom_new_chi: $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1: $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
grd_nchi2: $nchi < -MAX\_\chi$
grd_CRIT_DIST: $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto f2(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto -MAX\_\chi)) > CRITICAL\_DISTANCE)$
$)$
grd_Y_DIST: $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto -MAX\_\chi) \mapsto nearest(f1(pos(d\_vehicle) \mapsto MAX\_ACCEL \mapsto -MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$
with
magic_accel: $magic\_accel = MAX\_ACCEL$
magic_chi: $magic\_chi = -MAX\_\chi$
then
act_accel: $accel(d\_vehicle) := MAX\_ACCEL$
act_chi: $chi(d\_vehicle) := -MAX\_\chi$
act_d_vehi: $d\_vehicle := d\_vehicle + 1$
end

Event $decide\_amin\_\chi \mathrel{\widehat{=}}$
Status convergent
refines *decide*
any
$naccel$
$nchi$
where
grd_vehi: $vehicle = 1$
grd_d_vehi: $d\_vehicle \in 2..VEHICLES$
grd_p_vehi: $p\_vehicle = VEHICLES + 1$
grd_dom_new_accel: $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
grd_naccel1: $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
grd_naccel2: $naccel < MIN\_ACCEL$
grd_dom_new_chi: $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
grd_nchi1: $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$

grd_nchi2: $nchi \in -MAX\_\chi \mathinner{..} MAX\_\chi$

grd_CRIT_DIST: $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle-1) \mapsto accel(d\_vehicle-1) \mapsto chi(d\_vehicle-1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto nchi)) > CRITICAL\_DISTANCE)$
$)$

grd_Y_DIST: $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto nchi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto nchi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

with

magic_accel: $magic\_accel = MIN\_ACCEL$

magic_chi: $magic\_chi = nchi$

then

act_accel: $accel(d\_vehicle) := MIN\_ACCEL$

act_chi: $chi(d\_vehicle) := nchi$

act_d_vehi: $d\_vehicle := d\_vehicle + 1$

end

Event $decide\_amin\_\chi max \mathrel{\widehat{=}}$
Status convergent
refines $decide$

any

$naccel$
$nchi$

where

grd_vehi: $vehicle = 1$

grd_d_vehi: $d\_vehicle \in 2 \mathinner{..} VEHICLES$

grd_p_vehi: $p\_vehicle = VEHICLES + 1$

grd_dom_new_accel: $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$

grd_naccel1: $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$

grd_naccel2: $naccel < MIN\_ACCEL$

grd_dom_new_chi: $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$

grd_nchi1: $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle)$
$\mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$

grd_nchi2: $nchi > MAX\_\chi$

grd_CRIT_DIST: $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle-1) \mapsto accel(d\_vehicle-1) \mapsto chi(d\_vehicle-1)) \mapsto$
$f2(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto MAX\_\chi)) > CRITICAL\_DISTANCE)$
$)$

grd_Y_DIST: $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min,$
$new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min,$
$new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto MAX\_\chi) \mapsto$
$nearest(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$

with
`magic_accel` : $magic\_accel = MIN\_ACCEL$
`magic_chi` : $magic\_chi = MAX\_\chi$
then
`act_accel` : $accel(d\_vehicle) := MIN\_ACCEL$
`act_chi` : $chi(d\_vehicle) := MAX\_\chi$
`act_d_vehi` : $d\_vehicle := d\_vehicle + 1$
end

Event $decide\_amin\_\chi min \mathrel{\widehat{=}}$
Status convergent
refines *decide*
any
$naccel$
$nchi$
where
`grd_vehi` : $vehicle = 1$
`grd_d_vehi` : $d\_vehicle \in 2..VEHICLES$
`grd_p_vehi` : $p\_vehicle = VEHICLES + 1$
`grd_dom_new_accel` : $p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \in dom(new\_accel)$
`grd_naccel1` : $naccel = new\_accel(p\_pre\_v(d\_vehicle) \mapsto p\_v(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle))$
`grd_naccel2` : $naccel < MIN\_ACCEL$
`grd_dom_new_chi` : $p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle) \in dom(new\_chi)$
`grd_nchi1` : $nchi = new\_chi(p\_v(d\_vehicle) \mapsto p\_\kappa(d\_vehicle) \mapsto p\_\Delta x(d\_vehicle) \mapsto p\_\Delta y(d\_vehicle) \mapsto p\_\Delta\gamma(d\_vehicle) \mapsto p\_\Delta\sigma(d\_vehicle))$
`grd_nchi2` : $nchi < -MAX\_\chi$
`grd_CRIT_DIST` : $\exists f1, f2 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$f2 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(dist(f1(pos(d\_vehicle - 1) \mapsto accel(d\_vehicle - 1) \mapsto chi(d\_vehicle - 1)) \mapsto f2(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto -MAX\_\chi)) > CRITICAL\_DISTANCE)$
$)$
`grd_Y_DIST` : $\exists f1 \cdot ($
$f1 \in \{new\_pos\_v\_\kappa, new\_pos\_v\_\kappa max, new\_pos\_v\_\kappa min, new\_pos\_vmax\_\kappa, new\_pos\_vmax\_\kappa max, new\_pos\_vmax\_\kappa min, new\_pos\_vmin\_\kappa, new\_pos\_vmin\_\kappa max, new\_pos\_vmin\_\kappa min\} \wedge$
$(y\_dist(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto -MAX\_\chi) \mapsto nearest(f1(pos(d\_vehicle) \mapsto MIN\_ACCEL \mapsto -MAX\_\chi) \mapsto trajectory)) < Y\_DISTANCE)$
$)$
with
`magic_accel` : $magic\_accel = MIN\_ACCEL$
`magic_chi` : $magic\_chi = -MAX\_\chi$
then
`act_accel` : $accel(d\_vehicle) := MIN\_ACCEL$
`act_chi` : $chi(d\_vehicle) := -MAX\_\chi$
`act_d_vehi` : $d\_vehicle := d\_vehicle + 1$
end

Event $move1\_v\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines *move1_v_κ*
any
$nspeed$
$nkappa$
$npos$
where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa)$
grd_npos : $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move1\_v\_\kappa max \mathrel{\widehat{=}}$
Status convergent
refines $move1\_v\_\kappa max$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa max)$
grd_npos : $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move1\_v\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines $move1\_v\_\kappa min$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmax_κ* $\widehat{=}$
Status convergent
refines *move1_vmax_κ*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move1_vmax_κmax* $\widehat{=}$
Status convergent
refines *move1_vmax_κmax*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move1_vmax_κmin* $\widehat{=}$
Status convergent
refines *move1_vmax_κmin*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$

grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa min)$
grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κ* $\widehat{=}$
Status convergent
refines *move1_vmin_κ*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κmax* $\widehat{=}$
Status convergent
refines *move1_vmin_κmax*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move1_vmin_κmin* $\widehat{=}$
Status convergent
refines *move1_vmin_κmin*

any

nspeed

nkappa
npos

where

grd_vehi : $vehicle = 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa min)$
grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_v_κ* $\widehat{=}$
Status convergent
refines *move_v_κ*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa)$
grd_npos : $npos = new\_pos\_v\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle - 1) \mapsto npos) > CRITICAL\_DISTANCE$

grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_v_κmax* $\widehat{=}$
Status convergent
refines *move_v_κmax*

any

nspeed
nkappa
npos

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa max)$

grd_npos : $npos = new\_pos\_v\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_v\_\kappa min \mathrel{\widehat{=}}$
Status convergent
refines $move\_v\_\kappa min$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed \in 0..MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_v\_\kappa min)$
grd_npos : $npos = new\_pos\_v\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_vmax\_\kappa \mathrel{\widehat{=}}$
Status convergent
refines $move\_vmax\_\kappa$

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa..MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa)$
grd_npos : $npos = new\_pos\_vmax\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event $move\_vmax\_\kappa max \mathrel{\widehat{=}}$

Status convergent
refines *move_vmax_κmax*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa max)$
grd_npos : $npos = new\_pos\_vmax\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move_vmax_κmin* $\widehat{=}$
Status convergent
refines *move_vmax_κmin*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed > MAX\_SPEED$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmax\_\kappa min)$
grd_npos : $npos = new\_pos\_vmax\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$
then
act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event *move_vmin_κ* $\widehat{=}$
Status convergent
refines *move_vmin_κ*
any
nspeed
nkappa
npos
where
grd_vehi : $vehicle \in 2..VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$

grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa \in -MAX\_\kappa \mathinner{..} MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa)$
grd_npos : $npos = new\_pos\_vmin\_\kappa(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmin_κmax* $\widehat{=}$
Status convergent
refines *move_vmin_κmax*

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2 \mathinner{..} VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa > MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa max)$
grd_npos : $npos = new\_pos\_vmin\_\kappa max(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$

end

Event *move_vmin_κmin* $\widehat{=}$
Status convergent
refines *move_vmin_κmin*

any

$nspeed$
$nkappa$
$npos$

where

grd_vehi : $vehicle \in 2 \mathinner{..} VEHICLES$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_nspeed1 : $nspeed = v(pos(vehicle)) + accel(vehicle)$
grd_nspeed2 : $nspeed < 0$
grd_nkappa1 : $nkappa = \kappa(pos(vehicle)) + chi(vehicle)$
grd_nkappa2 : $nkappa < -MAX\_\kappa$
grd_dom_new_pos : $pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle) \in dom(new\_pos\_vmin\_\kappa min)$
grd_npos : $npos = new\_pos\_vmin\_\kappa min(pos(vehicle) \mapsto accel(vehicle) \mapsto chi(vehicle))$
grd_CRIT_DIST : $dist(pos(vehicle-1) \mapsto npos) > CRITICAL\_DISTANCE$
grd_Y_DIST : $y\_dist(npos \mapsto nearest(npos \mapsto trajectory)) < Y\_DISTANCE$

then

act_pos : $pos(vehicle) := npos$
act_vehi : $vehicle := vehicle + 1$
end

Event $all\_moves \mathrel{\widehat{=}}$
refines all_moves
when
grd_vehi : $vehicle = VEHICLES + 1$
grd_d_vehi : $d\_vehicle = VEHICLES + 1$
grd_p_vehi : $p\_vehicle = VEHICLES + 1$
grd_Y_DIST : $\forall vehi \cdot ($
$vehi \in 2..VEHICLES \Rightarrow$
$y\_dist(pos(vehi) \mapsto nearest(pos(vehi) \mapsto$
$trajectory \lhd\!\!+ \{temp + 1 \mapsto pos\})) < Y\_DISTANCE$
$)$
then
act_pos0 : $pos0 := pos$
act_temp : $temp := temp + 1$
act_traj : $trajectory := trajectory \lhd\!\!+ \{temp + 1 \mapsto pos\}$
act_vehi : $vehicle := 1$
act_d_vehi : $d\_vehicle := 1$
act_p_vehi : $p\_vehicle := 1$
end
VARIANT
$(VEHICLES + 1) - p\_vehicle$
END

Bibliography

[Abrial 1996] J.-R. Abrial. The B book. Cambridge University Press, 1996. 15, 100

[Abrial 1999] J.-R. Abrial. *Event Driven System Construction*. Rapport technique, Clearsy, 1999. 15

[Abrial 2006] Jean-Raymond Abrial. *Formal methods in industry: achievements, problems, future*. In Proceedings of the 28th international conference on Software engineering, pages 761–768. ACM, 2006. 111

[Abrial 2010] J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010. 15, 32, 40

[Abrial 2012] Jean-Raymond Abrial, Wen Su and Huibiao Zhu. *Formalizing Hybrid Systems with Event-B*. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves and Elvinia Riccobene, editeurs, Abstract State Machines, Alloy, B, VDM, and Z, volume 7316 of *Lecture Notes in Computer Science*, pages 178–193. Springer Berlin Heidelberg, 2012. 49

[Adrion 1982] W. Richards Adrion, Martha A. Branstad and John C. Cherniavsky. *Validation, Verification, and Testing of Computer Software*. ACM Comput. Surv., vol. 14, no. 2, pages 159–192, 1982. 10

[Ait Ameur 2010] Yamine Ait Ameur, Frédéric Boniol and Virginie Wiels. *Toward a wider use of formal methods for aerospace systems design and verification*. International Journal on Software Tools for Technology Transfer, vol. 12, no. 1, pages 1–7, 2010. 13

[Ait-Sadoune 2009] Idir Ait-Sadoune and Yamine Ait-Ameur. *Animating Event B Models by Formal Data Models*. In Tiziana Margaria and Bernhard Steffen, editeurs, Leveraging Applications of Formal Methods, Verification and Validation, volume 17 of *Communications in Computer and Information Science*, pages 37–55. Springer Berlin Heidelberg, 2009. 20

[Andriole 1986] Stephen J. Andriole. Software Validation, Verification, Testing and Documentation: A Source Book. Petrocelli Books, Inc., Princeton, NJ, USA, 1986. 10

[Ashenden 2002] P.J. Ashenden. The designer's guide to vhdl. Electronics & Electrical. Morgan Kaufmann, 2002. 20

[B-Core(UK) Ltd 1996] B-Core(UK) Ltd. *B-Toolkit User's Manual, Release 3.2*, 1996. 15

[Baille 1999] Gérard Baille, Philippe Garnier, Hervé Mathieu and Pissard-Gibollet Roger. *Le cycab de l'INRIA Rhônes-Alpes*. Technical Report RT-0229, INRIA – Rhônes-Alpes, 1999. 23

[Balzer 1981] Robert M. Balzer and Neil M. Goldman. *Principles of Good Software Specification and their Implications for Specification Languages*. In Proceedings of AFIPS '81, National Computer Conference, pages 393–400, New York, USA, 1981. ACM. 9

[Balzer 1982] Robert M. Balzer, Neil M. Goldman and David S. Wile. *Operational Specification as the Basis for Rapid Prototyping*. SIGSOFT Softw. Eng. Notes, vol. 7, no. 5, pages 3–16, 1982. 58

[Barner 2005] Jörg Barner. *A Lightweight Formal Method for the Prediction of Non-Functional System Properties*. PhD thesis, Friedrich-Alexander-Universität, Arbeitsberichte des Instituts für Informatik Bd. 38, Nr. 4, 2005. 13

[Beck 2001] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland and Dave Thomas. *Manifesto for Agile Software Development*, 2001. http://agilemanifesto.org/. 8

[Behm 1999] P. Behm, P. Benoit and J. M. Meynadier. *METEOR: A Successful Application of B in a Large Project*. In FM, volume 1708 of *LNCS*, pages 369–387. Springer Verlag, 1999. 15

[Bellegarde 2002] Françoise Bellegarde, Samir Chouali and Jacques Julliand. *Verification of Dynamic Constraints for B Event Systems under Fairness Assumptions*. In Didier Bert, JonathanP. Bowen, MartinC. Henson and Ken Robinson, editeurs, ZB 2002:Formal Specification and Development in Z and B, volume 2272 of *Lecture Notes in Computer Science*, pages 477–496. Springer Berlin Heidelberg, 2002. 51

[Bendisposto 2008] J. Bendisposto, M. Leuschel, O. Ligot and M. Samia. *La validation de modèles Event-B avec le plug-in ProB pour RODIN*. Technique et Science Informatiques, vol. 27, no. 8, pages 1065–1084, 2008. 20

[Bendisposto 2010] Jens Bendisposto, Fabian Fritz and Michael Leuschel. *Developing Camille, a Text Editor for Rodin*. In Proc. Workshop on Tool Building in Formal Methods, colocated with ABZ Conference – Orford – Canada, 2010. 19, 52

[Bert 2003] D. Bert, S. Boulmé, M.-L. Potet, A. Requet and L. Voisin. *Adaptable Translator of B Specifications to Embedded C Programs*. In Integrated Formal Method, IFM'03, volume 2805 of *LNCS*, pages 94–113. Springer Verlag, 2003. 14

[Bjesse 2005] Per Bjesse. *What is formal verification?* SIGDA Newsletter, vol. 35, no. 24, 2005. 14

[Boehm 1988] Barry W. Boehm. *A spiral model of software development and enhancement*. Computer, vol. 21, no. 5, pages 61–72, 1988. 12

[Bom 2005] J. Bom, B. Thuilot, F. Marmoiton and P. Martinet. *Nonlinar Control for Urban Vehicle Platooning, Relying upon a Unique Kinematic GPS*. In International Conference on Robotics and Automation. IEEE, 2005. 23

[Bowen 1995a] Jonathan P. Bowen and Michael G. Hinchey. *Seven More Myths of Formal Methods*. IEEE Softw., vol. 12, no. 4, pages 34–41, 1995. 13

[Bowen 1995b] Jonathan P. Bowen and Michael G. Hinchey. *Ten Commandments of Formal Methods*. Computer, vol. 28, pages 56–63, 1995. 13

[Bowen 2006] Jonathan P. Bowen and Michael G. Hinchey. *Ten Commandments of Formal Methods ...Ten Years Later*. Computer, vol. 39, pages 40–48, 2006. 13

[Brooks 1987] Jr. Frederick P. Brooks. *No Silver Bullet Essence and Accidents of Software Engineering*. Computer, vol. 20, no. 4, pages 10–19, 1987. 8

[Burdy 2012] Lilian Burdy, Jean-Louis Dufour and Thierry Lecomte1. *The B method takes up floating-point numbers*. In Embedded Real Time Software and Systems, 2012. 49

[Butler 2002] Michael Butler. *A System-Based Approach to the Formal Development of Embedded Controllers for a Railway*. Design Automation for Embedded Systems, vol. 6, pages 355–366, 2002. 13

[Clarke 1996] Edmund M. Clarke and Jeannette M. Wing. *Formal methods: state of the art and future directions*. ACM Comput. Surv., vol. 28, no. 4, pages 626–643, 1996. 13

[Cle 2009a] Clearsy. *Atelier B–Interactive Prover Reference Manual, version 4.0*, 2009. 19, 52

[Cle 2009b] Clearsy. *Atelier B 4.0 User manual*, 2009. 15

[Colin 2008] S. Colin, A. Lanoix, O. Kouchnarenko and J. Souquières. *Towards Validating a Platoon of Cristal Vehicles using CSP||B*. In J. Meseguer and G. Rosu, editeurs, 12th International Conference on Algebraic Methodology and Software Technology (AMAST 2008), numéro 5140 de LNCS, pages 139–144. Springer-Verlag, July 2008. 25

[Daviet 1996] P. Daviet and M. Parent. *Longitudinal and Lateral Servoing of Vehicles in a Platoon*. In Proceeding of the IEEE Intelligent Vehicles Symposium, pages 41–46, 1996. 24, 45

[Déharbe 2012] David Déharbe, Pascal Fontaine, Yoann Guyot and Laurent Voisin. *SMT Solvers for Rodin*. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves and Elvinia Riccobene, editeurs, Abstract State Machines, Alloy, B, VDM, and Z, volume 7316 of *Lecture Notes in Computer Science*, pages 194–207. Springer Berlin Heidelberg, 2012. 19

[Edmunds 2012] Andrew Edmunds, Abdolbaghi Rezazadeh and Michael Butler. *Formal Modelling for Ada Implementations: Tasking Event-B*. In Mats Brorsson and LuísMiguel Pinho, editeurs, Reliable Software Technologies – Ada-Europe 2012, volume 7308 of *Lecture Notes in Computer Science*, pages 119–132. Springer Berlin Heidelberg, 2012. 20

[Ferber 1996] J. Ferber and J. P. Muller. *Influences and Reaction : a Model of Situated Multiagent Systems*. In 2nd Int. Conf. on Multi-agent Systems, pages 72–79, 1996. 24, 46

[Ferber 1999] J. Ferber. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Professional, 1999. 24

[Forsberg 1995] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. Center for Systems Management, 1995. 12

[Fuchs 1992] Norbert E. Fuchs. *Specifications are (preferably) executable*. Software Engineering Journal, vol. 7, pages 323–334, September 1992. 58

[Gerhart 1994] S. Gerhart, D. Craigen and T. Ralston. *Case study: Paris Metro Signaling System*. IEEE Software, vol. 11, no. 1, pages 28–32, 1994. 15

[Gosling 2005] James Gosling, Bill Joy, Guy Steele and Gilad Bracha. Java language specification, the 3rd edition. Addison-Wesley Professional, 2005. 8

[Hallerstede 2010] Stefan Hallerstede, Michael Leuschel and Daniel Plagge. *Refinement-Animation for Event-B - Towards a Method of Validation*. In Proceedings ABZ'2010, volume 5977 of *Lecture Notes in Computer Science*, pages 287–301. Springer-Verlag, 2010. 23

[Hallerstede 2011] Stefan Hallerstede and Michael Leuschel. *Constraint-Based Deadlock Checking of High-Level Specifications*. Theory and Practice of Logic Programming, vol. 11, no. 4–5, pages 767–782, 2011. 39

[Hallerstede 2013] Stefan Hallerstede, Michael Leuschel and Daniel Plagge. *Validation of Formal Models by Refinement Animation*. Science of Computer Programming, vol. 78, no. 3, pages 272–292, 2013. 23

[Idir 2010] Ait-Sadoune Idir. *Modélisation et Vérification Formelles de Compositions de Services. Une Approche Fondée sur le Raffinement et la Preuve.* PhD thesis, École Nationale Supérieure de Mécanique et Aérotechnique (ENSMA), Poitiers, Décembre 2010. 41

[IEEE 2004] Institute of Electrical IEEE and Electronics Engineers. *IEEE Standard for Software Verification and Validation*, 2004. 10

[Jackson 2001] Daniel Jackson. *Lightweight Formal Methods*. In Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity, FME '01, pages 1–, London, UK, UK, 2001. Springer-Verlag. 13

[Ladenberger 2009] Lukas Ladenberger, Jens Bendisposto and Michael Leuschel. *Visualising Event-B Models with B-Motion Studio*. In María Alpuente, Byron Cook and Christophe Joubert, editeurs, Formal Methods for Industrial Critical Systems, volume 5825 of *Lecture Notes in Computer Science*, pages 202–204. Springer Berlin Heidelberg, 2009. 22

[Lamport 1977] L. Lamport. *Proving the Correctness of Multiprocess Programs*. IEEE Transactions on Software Engineering, vol. 3, no. 2, pages 125–143, 1977. 50

[Lamsweerde 2000] Axel van Lamsweerde. *Formal specification: a roadmap*. In ICSE'00: Proceedings of the Conference on The Future of Software Engineering, pages 147–159, New York, USA, 2000. ACM. 13

[Lanoix 2008] Arnaud Lanoix. *Event-B Specification of a Situated Multi-Agent System: Study of a Platoon of Vehicles*. In 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2008), pages 297–304, France, 2008. 4, 24, 90

[Larman 2003] Craig Larman and Victor R Basili. *Iterative and incremental developments. A brief history*. Computer, vol. 36, no. 6, pages 47–56, 2003. 12

[Leino 2010] K.RustanM. Leino. *Dafny: An Automatic Program Verifier for Functional Correctness*. In EdmundM. Clarke and Andrei Voronkov, editeurs, Logic for Programming, Artificial Intelligence, and Reasoning, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer Berlin Heidelberg, 2010. 20

[Leroy 2009a] Xavier Leroy. *Formal verification of a realistic compiler*. Commun. ACM, vol. 52, no. 7, pages 107–115, July 2009. 112

[Leroy 2009b] Xavier Leroy. *A Formally Verified Compiler Back-end*. J. Autom. Reason., vol. 43, no. 4, pages 363–446, December 2009. 112

[Leuschel 2001] M. Leuschel, L. Adhianto, M. Butler, C. Ferreira and L. Mikhailov. *Animation and Model Checking of CSP and B using Prolog Technology*. In Proceedings of the ACM Sigplan Workshop on Verification and Computational Logic VCL'2001, pages 97–109, 2001. 20

[Leuschel 2003] M. Leuschel and M. Butler. *ProB: A Model Checker for B*. In Keijiro Araki, Stefania Gnesi and Dino Mandrioli, editeurs, FME 2003: Formal Methods, LNCS 2805, pages 855–874. Springer-Verlag, 2003. 19

[Leuschel 2008] M. Leuschel and M. Butler. *ProB: An Automated Analysis Toolset for the B Method*. Journal Software Tools for Technology Transfer, vol. 10, no. 2, pages 185–203, 2008. 19, 39, 53

[Leuschel 2011] Michael Leuschel, Jérôme Falampin, Fabian Fritz and Daniel Plagge. *Automated property verification for large scale B models with ProB*. Formal Aspects of Computing, pages 1–27, 2011. 10.1007/s00165-010-0172-1. 44

[Ligot 2007] Olivier Ligot, Jens Bendisposto and Michael Leuschel. *Debug Event-B Models using the ProB Disprover Plugin*. In Approches Formelles dans l'Assistance au Développement de Logiciels(AFADL'07), Namur, Belgium, 2007. 19

[Maibaum 2007] Tom Maibaum. *Challenges in software certification*. In ICFEM'07: Proceedings of the 9th international conference on Formal methods and software engineering, pages 4–18. Springer-Verlag, 2007. 10

[Mashkoor 2009a] Atif Mashkoor, Jean-Pierre Jacquot and Jeanine Souquières. *B Evénementiel pour la Modélisation du Domaine: Application au Transport*. In Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'09), Toulouse, France, 2009. 4

[Mashkoor 2009b] Atif Mashkoor, Jean-Pierre Jacquot and Jeanine Souquières. *Transformation Heuristics for Formal Requirements Validation by Animation*. In 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems (SafeCert'09), York, UK, 2009. 21, 58, 93, 95

[Mashkoor 2010] Atif Mashkoor and Jean-Pierre Jacquot. *Domain Engineering with Event-B: Some Lessons We Learned*. In 18th International Requirements Engineering Conference - RE'10, pages 252–261, Sydney Australie, 2010. IEEE. 53

[Méry 2011] D. Méry and N.K. Singh. *Automatic Code Generation from Event-B Models*. In Proc. Symposium on Information and Communication Technology, Hanoi, Vietnam, 2011. ACM. 20

[Métayer 2012] C. Métayer. *B model animator*. Website, 2012. http://www.animb.org. 19

[Parnas 1972] David Lorge Parnas. *On the criteria to be used in decomposing systems into modules*. Communications of the ACM, vol. 15, no. 12, pages 1053–1058, 1972. 8

[Patin 2006] F. Patin, G. Pouzancre, D. Sabatier, S. Hauvespre and P. Sauvage. *Utilisation de la méthode formelle B pour un système SIL3 : la commande des portes palière sur la ligne 13 du métro Parisien*. In Conférence Lamda MU 15, 2006. 15

[Rehm 2009] Joris Rehm. *Gestion du temps par le raffinement*. PhD thesis, Université Henri Poincaré, Nancy 1, 2009. 50

[RODIN 2012] RODIN. *Rigorous Open Development Environment for Complex Systems*. Website, consulted November 2012. http://www.event-b.org. 15, 18

[Romanovsky 2013] Alexander Romanovsky. Industrial deployment of system engineering methods. Springer, 2013. 112

[Royce 1970] Winston W. Royce. *Managing the development of large software systems*. In proceedings of IEEE WESCON, volume 26. Los Angeles, 1970. 12

[Rushby 1993] J. Rushby. *Formal Methods and the Certification of Critical Systems*. Technical Report CLS-93-7, Computer Science Laboratory – SRI International, December 1993. 13

[Savicks 2009] Vitaly Savicks, Colin Snook and Michael Butler. *Animation of UML-B State-machines*. Rapport technique, University of Southampton, 2009. 19

[Schenck 1994] Douglas A. Schenck and Peter R. Wilson. Information Modeling the EXPRESS Way. Oxford University Press, USA, 1994. 22

[Scheuer 2009] Alexis Scheuer, Olivier Simonin and François Charpillet. *Safe longitudinal platoons of vehicles without communication*. In ICRA'09: Proceedings of IEEE international conference on Robotics and Automation, pages 2835–2840, Piscataway, NJ, USA, 2009. IEEE Press. 23, 38

[Schmalz 2011] Matthias Schmalz. *Term Rewriting in Logics of Partial Functions*. In Shengchao Qin and Zongyan Qiu, editeurs, Formal Methods and Software Engineering, volume 6991 of *Lecture Notes in Computer Science*, pages 633–650. Springer Berlin Heidelberg, 2011. 19

[Schmid 2000] Reto Schmid, Johannes Ryser, Stefan Berner, Martin Glinz, Ralf Reutemann and Erwin Fahr. *A Survey of Simulation Tools for Requirements Engineering*. Rapport technique 2000.06, University of Zurich, 2000. 20

[Schwartz 1986] J. T. Schwartz, R. B. Dewar, E. Schonberg and E. Dubinsky. Programming with sets; an introduction to SETL. Springer-Verlag, New York, USA, 1986. 105

[Servat 2007] Thierry Servat. *BRAMA: A New Graphic Animation Tool for B Models*. In B 2007: Formal Specification and Development in B, pages 274–276. Springer-Verlag, 2007. 19, 39, 53

[Siddiqi 1997] Jawed I. Siddiqi, Ian C. Morrey, Chris R. Roast and Mehmet B. Ozcan. *Towards quality requirements via animated formal specifications*. Ann. Softw. Eng., vol. 3, pages 131–155, 1997. 20

[Simonin 2007] O. Simonin, A. Lanoix, S. Colin, A. Scheuer and F. Charpillet. *Generic Expression in B of the Influence/Reaction Model: Specifying and Verifying Situated Multi-Agent Systems*. INRIA Research Report 6304, INRIA, September 2007. 24, 25

[Stamatis 2003] D.H. Stamatis. Failure mode and effect analysis: Fmea from theory to execution. American Society for Quality, 2003. 13

[Van 2004] Hung Tran Van, Axel van Lamsweerde, Philippe Massonet and Christophe Ponsard. *Goal-Oriented Requirements Animation*. In RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International, pages 218–228, Washington, DC, USA, 2004. IEEE Computer Society. 20

[Wing 1990] Jeannette M. Wing. *A Specifier's Introduction to Formal Methods*. Computer, vol. 23, no. 9, pages 8–23, 1990. 13

[Wright 2009a] Steve Wright. *Automatic Generation of C from Event-B*. In IM_FMT, Workshop on Integration of Model-based Formal Methods and Tools, Dusseldorf, Germany, 2009. 20

[Wright 2009b] Steve Wright. *MIDAS Machine Specification*. Rapport technique, Department of Computer Science, University of Bristol, March 2009. 4, 52, 81, 91

[Wright 2010] Steve Wright and Kerstin Eder. *Using Event-B to construct Instruction Set Architectures*. Formal Aspects of Computing, vol. 23, no. 1, pages 73–89, January 2010. 4, 81

[Yang 2011] Faqing Yang and Jean-Pierre Jacquot. *Scaling Up with Event-B: A Case Study*. In The 3rd NASA Formal Methods Symposium (NFM'11), volume 6617 of *LNCS*, pages 438–452, California, USA, 2011. Springer Berlin / Heidelberg. 4, 25

[Yang 2012] Faqing Yang, Jean-Pierre Jacquot and Jeanine Souquières. *Traduction de B événementiel en C pour la validation par la simulation*. In Approches Formelles dans l'Assistance au Développement de Logiciels - AFADL, Grenoble, France, janvier 2012. 61

Abstract

This thesis aims at the specification, verification and validation of safety-critical systems with formal methods, in particular, with Event-B. We assessed the usability of Event-B by the development of platooning control algorithms, specially how it scaled up from a simplified 1D version to a more realistic 2D version. The critical analysis of the 1D platooning model uncovered some anomalous behaviors. The difficulty of expressing the deadlock-freeness theorems in Event-B motivated us to develop a tool, the generator of deadlock-freeness theorems, to automatically construct these theorems. Our assessment confirmed that the mathematical proofs are not sufficient to assure the correctness of a formal specification: a formal specification should also be validated. We believe that the validation activities, like the verification activities, should be associated with each refinement during the development. To do that, we need better validation tools. The state-of-the-art tools which can execute Event-B models failed in highly non-deterministic models. Therefore we designed and implemented a new execution tool, JeB, which is a JavaScript simulation framework for Event-B. JeB allows users to safely insert hand-coded pieces of code to supply deterministic computations where the automatic translation fails. To achieve this goal, we have defined a set of proof-obligations which, when discharged, guarantee the correctness of the simulations with respect to the model.

Keywords: Specification, Verification, Validation, Formal methods, Event-B, Simulation, JavaScript

Résumé

Cette thèse porte sur la spécification, la vérification et la validation de systèmes critiques à l'aide de méthodes formelles, en particulier, B événementiel. Nous avons travaillé sur l'utilisation de B événementiel pour étudier des algorithmes de contrôle du *platooning*, à partir d'une version 1D simplifiée vers une version 2D plus réaliste. L'analyse critique du modèle du *platooning* en 1D a découvert certaines anomalies. La difficulté d'exprimer les théorèmes de *deadlock-freeness* dans B événementiel nous a motivé pour développer un outil, le générateur de théorèmes de *deadlock-freeness*, pour construire automatiquement ces théorèmes. Notre évaluation a confirmé que les preuves mathématiques ne sont pas suffisantes pour vérifier la correction d'une spécification formelle : une spécification formelle doit aussi être validée. Nous pensons que les activités de validation, comme les activités de vérification, doivent être associées à chaque raffinement. Pour ce faire, nous avons besoin de meilleurs outils de validation. Certains outils d'exécution existants échouent pour certains modèles non-déterministes exprimés en B événementiel. Nous avons donc conçu et implanté un nouvel outil d'exécution, JeB, un environnement de simulation en JavaScript pour B événementiel. JeB permet aux utilisateurs d'insérer du code sûr à la main pour fournir des calculs déterministes lorsque la traduction automatique échoue. Pour atteindre cet objectif, nous avons défini des obligations de preuve qui garantissent la correction de simulations par rapport au modèle formel.

Mots-clés : Spécification, Vérification, Validation, Méthodes formelles, B événementiel, Simulation, JavaScript