Model Based System Consistency Checking Using Event-B

MODEL BASED SYSTEM CONSISTENCY CHECKING USING EVENT-B

BY

HAO XU, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2011) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Model Based System Consistency Checking Using Event-B
AUTHOR:	Hao Xu B.Sc.
SUPERVISOR:	Dr. Tom Maibaum
NUMBER OF PAGES:	xii, 132

To my family.

Abstract

Formal methods such as Event-B are a widely used approach for developing critical systems. This thesis demonstrates that creating models and proving the consistency of the models at the requirements level during software (system) development is an effective way to reduce the occurrence of faults and errors in a practical application. An insulin infusion pump (IIP) is a complicated and time critical system. This thesis uses Event-B to specify models for an IIP, based on a draft requirements document developed by the US Food and Drug Administration (FDA). Consequently it demonstrates Event-B can be used effectively to detect the missing properties, the missing quantities, the faults and the errors at the requirements level of a system development. The IIP is an active and reactive time control system. To achieve the goal of handling timing issues in the IIP system, we made extensions of an existing time pattern specified using Event-B to enrich the semantics of the Event-B language. We created several sets to model the activation times of different events and the union of these time sets defines a global time activation set. The tick of global time is specified as a progress tick event. All the actions in an event are triggered only when the global time in the time tick event matches the time specified in the event. Time is deleted from the corresponding time set, but not the corresponding global time set while the event is triggered. A time point is deleted from the global time set only when there

are no pending actions for that time point. Through discharging proof obligations using Event-B, we achieved our goal of improving the requirements document.

Acknowledgements

First of all, I offer my sincerest gratitude to my supervisor, Dr. Thomas Maibaum, who has supported me throughout my thesis with his patience and knowledge. Through two years of study, I learned the patience and the preciseness of the academic research. Without him this thesis would not have been completed or written.

This thesis came about as a result of the author participating in a project at the McMaster Software Certification Centre in year 2011. This project includes a group of two professors and four graduate students. I am grateful to the following members of this project: Dr. Alan Wassyng, Dr. Mark Lawford, John Stribbell, Linna Pang, Grant Whinton and Esteban Bucio. To work together with such a group of people with passion and inspiration is my honor.

I am very grateful to the committee members: Dr. Mark Lawford, Dr. Emil Sekerinski and Dr. Thomas Maibaum. Thanks for the valuable comments and corrections.

Finally, I am deeply grateful to Dr. Thomas Maibaum and Dr. Chris George for their help in careful proofreading the thesis.

Contents

Abstract **iv**

Acknowledgements **vi**

1 Introduction **1**

1.1 Motivation 1

1.2 Why Event-B 3

1.3 Overview of the Chapters 4

1.3.1 *Chapter 1: Introduction* 4

1.3.2 *Chapter 2: The Event-B Language* 4

1.3.3 *Chapter 3: The Insulin Infusion Pump* 5

1.3.4 *Chapter 4: Modelling the IIP* 5

1.3.5 *Chapter 5: Future Work* 6

1.3.6 *Chapter 5: Conclusion* 7

2 The Event-B Language **8**

2.1 Mathematical Languages 9

2.1.1 Sequent Calculus 9

2.1.2 Inference Rules 11

2.1.3 Set Theory 15
2.2 Event-B Notation 17
2.2.1 Components and Relations 17
2.2.2 Context and Machine Notation 18
2.3 Proof Obligation Rules 21
2.3.1 Invariant Preservation Rule 22
2.3.2 Feasibility Rule 22
2.3.3 Guard Strengthening Rule 23
2.3.4 Simulation Rule 24
2.3.5 Witness Feasibility Rule 24
2.3.6 Well-definedness Rule 25

3 The Insulin Infusion Pump **26**
3.1 Components 27
3.2 Commands and Actions 28
3.3 Interaction Behaviour 31
3.4 Requirements of IIP 34

4 Modelling the IIP **35**
4.1 Refinement Strategies 35
4.2 System Structure Review 40
4.3 Initial Model 43
4.3.1 Model Description 43
4.3.2 Formalizing the States 44
4.3.3 Formalizing the Events 45

4.3.4 Summary of the Initial Model 46
4.4 First Refinement: Refining Phases 47
4.4.1 Model Description 47
4.4.2 Refining the State 47
4.4.3 Refining the Events 49
4.4.4 Summary of the First Refinement 51
4.5 Second Refinement: Basal Profile Setting 52
4.5.1 Model Description 52
4.5.2 Refining the State 52
4.5.3 Refining the Events 54
4.5.4 Summary of the Second Refinement 56
4.6 Time Constraint Patterns 56
4.6.1 An Existing Time Pattern 56
4.6.2 Extension of Time Pattern 58
4.6.3 A Time Pattern with Unpredictable Events 62
4.6.4 Time Pattern for Classifying the Events 64
4.6.5 Discussions of Zeno Behaviour 67
4.6.6 Comparison with Other Approaches 69
4.7 Third Refinement: Introducing the Time Pattern 70
4.7.1 Model Description 70
4.7.2 Refining the States 71
4.7.3 Refining the Events 76
4.7.4 Summary of the Third Refinement 96
4.8 Summary of the IIP Model 97

4.8.1 Summary of Refinement Stage 97
4.8.2 Summary of Inconsistencies 99

5 Future Work **101**

5.1 Future Work on the IIP Project 101
5.2 Future Work on Technical Research 102
5.2.1 No Support for Real Numbers 103
5.2.2 Refinement Consistency Proof Obligation Rules 104
5.2.3 Completeness of the Model 105

6 Conclusion **108**

A IIP Requirements **111**

A.1 Infusion Control 111
A.2 Basal Programming and Administration 111
A.3 Bolus Calculation and Administration 112
A.4 Drug Reservoir 113
A.5 Pump Suspension 113
A.6 Alerts and Alarms 114

B Model Summary **115**

B.1 Initial Model 115
B.1.1 Model 115
B.1.2 Proof Obligations 116
B.2 Refinement One 116
B.2.1 Model 116

B.2.2 Proof Obligations 118
B.3 Refinement Two 119
B.3.1 Model 119
B.3.2 Proof Obligations 120
B.4 Refinement Three 120
B.4.1 Model 120

List of Figures

2.1 Contexts and Machines Relation Example 18
3.1 IIP 27
4.1 Structure of the IIP 41
4.2 Events Trail 67
4.3 Time Line with Events 68
4.4 Proof Tree in Rodin 77
4.5 Proof Tree Structure 77
4.6 Proof Tree of INFU_START_NORMAL/inv32/INV 81
4.7 Proof Tree of INFU_START_EXBO/inv32/INV 82
4.8 Proof Tree of BO_PROC/inv6/INV 93
4.9 Proof Tree of BO_PROC/inv9/INV 94
4.10 Refinement Relations 99

Chapter 1

Introduction

To develop robust and correct software has been the intention of some software developers for a long time. To design faultless software, people have to totally understand the required behaviour of the system and the cooperation between each part of the system. This presents a problem, which is how to make the requirements and the definitions of the systems accurate and faultless. For safety critical systems, it is necessary to find a formal way to detect and correct the limitations or faults in the documentation. This is one reason why formal methods play a significant role in the cycle of software engineering.

1.1 Motivation

The Insulin Infusion Pump (IIP) is a semi-automatic, patient controlled medical device whose purpose is to control the blood glucose (BG) level of diabetics by continuously or intermittently infusing insulin from an insulin reservoir into the patient. An IIP simulates the behaviour of the pancreas, which accurately delivers (using a

biological feedback mechanism) insulin to the body during normal activities of human beings.

The necessary accuracy and the complexity of the simulation of the pancreas behaviours by the IIP make it a high risk system. The faults and errors are arise from the system itself and because of the actions of the patients. A recent report from U.S. Food and Drug Administration (FDA) indicated that over 5000 adverse events for the Insulin Pump were reported during 2008 [YZPLJRJ10]. To avoid the occurrence of the faults and errors, designing a robust system with low errors and high fault tolerance becomes necessary.

Considering the situation mentioned above, in 2010 the FDA proposed a draft of some documentation for the Generic Insulin Infusion Pump (GIIP) [FDA10]. This document includes the requirements for the GIIP, with accompanying definitions and the hazards analysis for it as well.

To judge the correctness of a model is far easier than making the same judgement about the corresponding program [Abr10]. A project at McMaster University's McMaster Centre for Software Certification (McSCert) is trying to create models which contain all the features described in the document and formally prove the models are faultless, and then implement a practical infusion pump based on the formal models. Because the documents describing the GIIP are a draft, to find a method to make the requirements of the system complete and faultless is necessary and useful. This article describes the approach which we are using to identify the faults and errors in the requirements documents by creating some formal models and then developing the associated proofs about the models in the context of the formal specification language Event-B.

1.2 Why Event-B

Event-B is an extension of the B method [Abr96], which was developed by Jean-Raymond Abrial several years ago [Abr10]. Its notation is based on propositional logic and Zermelo-Fraenkel set theory [CM08]. Event-B uses refinement to relate different stages of the system description and uses the underlying mathematical languages to prove the consistency of every refinement (with respect to its predecessor) and invariant preservation for each event. The mathematical languages used in Event-B include sequent calculus, predicate logic, equality language, the boolean and arithmetic language and some advanced data structures [Abr10].

Event-B is a mature formal method which has been widely used in a number of industry projects in a number of domains, such as automotive, transportation, space, business information, medical device and so on. An FP7 project supported by the European Commission, called the Deploy Project [Dep08], is using Event-B to improve system dependability, to handle system complexity and to reduce the capital spent on the testing and debugging stages of software development. A recent technical report [MS10] written by Mery described a formal development of a Cardiac Pace Maker using Event-B, which shows the feasibility and validity of using Event-B during the development cycle of a medical device.

Another reason for using Event-B is that it is a tool-supported formal specification language. The Rodin platform [rod09, ABH$^+$10], an Eclipse-based IDE for Event-B, provides a user friendly user interface to create, refine and mathematically prove properties of models. Proof obligations are automatically generated after models have been created. The most powerful aspect of the Rodin platform is that a semi-automatic theorem prover is embedded in the tool, which saves lots of time compared

to creating the proofs manually. The Rodin platform is open source, so it supports a large number of plug-in tools, such as the animation tools AnimB[CM09], ProB[LB08] and model decomposition tool [SPHB10], The introduction of these plug-in tools makes modelling and proof construction more flexible.

1.3 Overview of the Chapters

We give a brief outline for each chapter in this thesis.

1.3.1 *Chapter 1: Introduction*

This chapter presents the motivation for the research topic and the reasons for choosing Event-B as our modelling language. Although our intention is to make the FDA requirements documents faultless and correct, the creation of models and the correctness proofs related to the models are only on an abstract level. We can fix the bugs and errors in the documents, but on the implementation level, bugs may still exist. Our intention in this thesis is to formally improve the requirements document, which is key for producing safe software by manufacturers.

1.3.2 *Chapter 2: The Event-B Language*

Event-B is a mature and widely used formal specification languages. The intent of this chapter is to give a brief introduction to Event-B. Event-B supports a refinement based formal method; in other words, starting with each model, the designer refines the so called abstract model to a more concrete model. Refinement, the concept of mathematical proof and proof obligations and corresponding rules are introduced in

this chapter. More details about Event-B can be found in J.R. Arial's book [Abr10].

1.3.3 *Chapter 3: The Insulin Infusion Pump*

In this chapter, we present a detailed informal description of the IIP, that is system behaviour and requirements description in natural language. The version of the document [FDA10] used as our starting point is an incomplete one. Some inconsistencies of the IIP requirements exist in this document, but the reader should note that these faults and errors are detected and fixed in subsequent chapters. The IIP is a complex system; our interests are mainly focused on the pumping process and the constraints for basal, bolus and extended bolus injections, so the requirements for these parts are described in detail in this chapter.

1.3.4 *Chapter 4: Modelling the IIP*

Modelling the IIP follows the typical approach to modelling in Event-B [Abr10] by refinement. This chapter provides a formal description to the properties of the IIP using the Axioms, Context and the Invariant parts of the Machine encapsulating the model.

Although Event-B describes systems using discrete events, with the help of some existing time patterns, it can handle timing issues as well. This thesis makes an extension of a time constraint pattern put forward by Mery [CMR07]. A sub-section in chapter 4 defines an extension of the time pattern and illustrates the usage of the extended time pattern on the IIP example. We create several sets to model the activation times of different events and the union of these time sets defines a global time activation set. All the actions in an event are triggered only when the global

time matches the time specified in the event. When the action is activated, the time is deleted from the corresponding time set, but not the corresponding global time set. A time point is deleted from the global time set only when there are no pending actions for that time point.

During the process of modelling the IIP and proving relevant proof obligations, several inconsistencies and missing properties of the draft FDA document are identified.

1.3.5 *Chapter 5: Future Work*

This chapter describes some future work that should be undertaken both on the IIP project and for the Event-B formal method.

The development of the IIP is a long term project. The work that has been done in this thesis focuses on the correctness checking of the requirement documents by creating a model and then checking relevant properties. Future work such as adding new features of the IIP, introducing environmental variables and even code generation are important issues which we have to address in the future.

Through the development of the IIP, some disadvantages of Event-B emerged, which forces us to reconsider the formal method. This chapter proposes several issues related to the disadvantages of the Event-B language, such as Real Number support, checking of model completeness. Moreover, the new time pattern in this thesis results in the creation of an obstruction to using refinement in the development of the model when using Event-B. We provide some general discussion and propose some partial solutions to the above issues.

1.3.6 *Chapter 5: Conclusion*

This chapter provides some conclusions about the formal method we applied in the project. The main contributions of this thesis are presented in this chapter:

We created a safety critical system model by using Event-B;

The correctness of the requirements document is (indirectly) checked by discharging the POs generated from the IIP model;

To handle the timing issues encountered during the development of the IIP model, we created several time patterns and applied them to the model.

Chapter 2

The Event-B Language

Event-B is an event based formal specification language. This chapter illustrates the Event-B language through the following perspectives: the mathematical languages that Event-B is based on, the Event-B notations (the structure of each component of the model) and the Proof Obligation (PO) rules. Because Event-B is a rich formal specification language, it is impossible for us to illustrate all the details of Event-B in this thesis. We choose some fundamental features of Event-B from [Abr10] and present them in this chapter. The descriptions in this chapter are intended to help readers to understand the IIP models specified using Event-B in the following chapters. Detail knowledge of Event-B and the semantics of logic expressions used in Event-B can be found in [Abr10, Abr96].

2.1 Mathematical Languages

2.1.1 Sequent Calculus

The definitions of some terms (*sequent, sequent rules, theory*) shall be presented before we describe the sequent calculus.

A *sequent* is constructed from two parts: the *hypotheses*, which includes several predicates, and the *goal*, which has only one predicate. We write a sequent in the following format:

$$H \vdash G$$

The above sequent means that the *goal* G holds when a set of *hypotheses* H hold.

An *inference rule* is is a relation between the antecedent and the consequent used when we construct a sequent based proof. An *inference rule* is made up of two parts: the *antecedent* part, which includes a finite set of sequents (the antecedent part can be empty), and the *consequent* part, which includes one sequent. We write a sequent rule in the following format:

$$\frac{A}{C}\ \mathbf{R}$$

The A stands for the antecedent part; the C stands for the consequent part; and the R stands for the name of the inference rule. This inference rule says that the inference rule **R** yields a proof of sequent C when we have the proofs of each sequent in A when A *is not empty.* When antecedent A is empty, we say the inference rule R yields a proof of sequent C.

A *theory* is a set of inference rules.

Through the definitions of *sequent*, *inference rule* and *theory* we see when we want

to prove a sequent in a theory we can construct a proof tree. Each node in the tree is represented by a tuple (S, **R**), in which S and **R** stand for the consequent of a sequent rule and the name of the relevant inference rule, respectively. The antecedent part of a node in the proof tree is the collection of all the consequent parts of the subtrees of this node. The leaves of a tree are those inference rules which have an empty antecedent part.

Below is an example to illustrate the construction of a proof tree. We have a theory including inference rules $\mathbf{R_1}$ to $\mathbf{R_6}$ and sequents from S_1 to S_6, which are written as follows:

$$\frac{S_2 \quad S_3}{S_1}\ \mathbf{R_5} \qquad \frac{S_4 \quad S_5}{S_2}\ \mathbf{R_2} \qquad \frac{}{S_4}\ \mathbf{R_1} \qquad \frac{S_6}{S_5}\ \mathbf{R_3} \qquad \frac{}{S_6}\ \mathbf{R_4} \qquad \frac{}{S_3}\ \mathbf{R_6}$$

To prove the sequent S_1, we can create the following proof tree:

$$S_1\ \mathbf{R_5} \begin{cases} S_2\ \mathbf{R_2} \begin{cases} S_4\ \mathbf{R_1} \\ S_5\ \mathbf{R_3} \quad S_6\ \mathbf{R_4} \end{cases} \\ S_3\ \mathbf{R_6} \end{cases}$$

The root $(S_1, \mathbf{R_5})$, whose antecedent parts are the consequent part of its offspring nodes $(S_2, \mathbf{R_2})$ (parent node of $(S_4, \mathbf{R_1})$ and $(S_5, \mathbf{R_3})$) and $(S_3, \mathbf{R_6})$ (antecedent part is empty), is the sequent that we intend to prove. The node $(S_5, \mathbf{R_3})$ only has one child node $(S_6, \mathbf{R_4})$. The tree has three leaves $(S_4, \mathbf{R_1})$, $(S_6, \mathbf{R_4})$ and $(S_3, \mathbf{R_6})$. All the leaves are sequent rules with empty antecedent.

2.1.2 Inference Rules

The theory of underlying Event-B is based on some initial inference rules. This theory is refined or extended by propositional logic and predicate logic. We summarize in Table 2.1 (the initial theory), Table 2.2 (the propositional logic extension), Table 2.3 (the predicate logic extension) the inference rules in Event-B and their definitions. These inference rules form the core of the approach to discharging proof obligations[1] in Event-B. We can construct proof trees for each proof obligation through the sequent calculus and discharge them by using the following inference rules.

Initial Theory

The basic rules are **HYP, MON** and **CUT**:

$$\frac{}{H, P \vdash P}\ \textbf{HYP} \qquad \frac{H \vdash Q}{H, P \vdash Q}\ \textbf{MON} \qquad \frac{H \vdash P \quad H, P \vdash Q}{H \vdash Q}\ \textbf{CUT}$$

Rules	Definitions
HYP	If goal P of a sequent is in the hypotheses, then the sequent is proved.
MON	To prove a sequent, we only need to prove another sequent with fewer hypotheses (without P) but with the same goal Q.
CUT	If we proved a goal P under hypotheses H, then P can be added to hypotheses H to prove another goal Q.

Table 2.1: Initial Inference Rules

[1]Proof obligations and proof obligation rules are explained in detail in Section 2.3.

Propositional Logic Extension

The propositional logic extends the initial inference rules by adding *falsity, negation, conjunction, disjunction and implication*. Each of the rules has two corresponding rules labelled with **_L** (the hypotheses part of the consequent) and labelled with **_R** (the goal part of the consequent).

$$\frac{}{H, \bot \vdash P}\ \textbf{FALSE\_L} \qquad \frac{H, \vdash P \quad H \vdash \neg P}{H \vdash \bot}\ \textbf{FALSE\_R}$$

$$\frac{H \vdash P}{H, \top \vdash P}\ \textbf{TRUE\_L} \qquad \frac{}{H \vdash \top}\ \textbf{TRUE\_R}$$

$$\frac{H, \neg Q \vdash P}{H, \neg P \vdash Q}\ \textbf{NOT\_L} \qquad \frac{H, P \vdash \bot}{H \vdash \neg P}\ \textbf{NOT\_R}$$

$$\frac{H, P, Q \vdash R}{H, P \wedge Q \vdash R}\ \textbf{AND\_L} \qquad \frac{H \vdash P \quad H \vdash Q}{H \vdash P \wedge Q}\ \textbf{AND\_R}$$

$$\frac{H, P \vdash R \quad H, Q \vdash R}{H, P \vee Q \vdash R}\ \textbf{OR\_L} \qquad \frac{H, \neg P \vdash Q}{H \vdash P \vee Q}\ \textbf{OR\_R}$$

$$\frac{H, P, Q \vdash R}{H, P, P \implies Q \vdash R}\ \textbf{IMP\_L} \qquad \frac{H, P \vdash Q}{H \vdash P \implies Q}\ \textbf{IMP\_R}$$

$$\frac{H, Q \vdash P \quad H, \neg Q \vdash P}{H \vdash P}\ \textbf{CASE}$$

Rules	Definitions
FALSE_L	If the hypotheses of a sequent included false assumption, then the sequent is proved.
FALSE_R	If a goal P and its negation $\neg P$ are both proved under hypotheses H, then H is false.
TRUE_L	This rule can be proved by using **MON** (substitute P by $\top$).
TRUE_R	This rule can be proved by using **HYP_L** and **AND_L**.
NOT_L	To prove a sequent which says the goal Q of the sequent is proved under the hypotheses H and $\neg P$, we only need to prove the sequent with hypotheses H and $\neg Q$, and with goal P.
NOT_R	If we succeed in proving the goal $\bot$ under hypotheses H and P, then the the sequent with goal of $\neg P$ can be proved.
AND_L	The relation between hypotheses H and some predicate P is conjunction.
AND_R	To prove a sequent with goal $P \wedge Q$, we should individually prove the sequent with the same hypotheses H, but with goals P and Q, respectively.
OR_L	To prove a sequent with hypotheses include $P \vee Q$, we have to prove the sequent with the same goal R, but with hypotheses P and Q, individually.
OR_R	If we succeed in proving the goal Q under hypotheses H and $\neg P$ and goal Q, then the new sequent with new goal $P \vee Q$ can be proved.
IMP_L	This rule is a derived rule from **OR_L** and **FALSE_R** (substitute $P \implies Q$ by $\neg P \vee Q$).
IMP_R	This rule is the same as rule **OR_R** (substitute $P \implies Q$ by $\neg P \vee Q$).
CASE	To prove a sequent with goal P, we should prove the sequent with hypotheses Q and the sequent with hypotheses $\neg Q$.

Table 2.2: Propositional Logic Extension Inference Rules

Predicate Logic Extension

The predicate logic extends the previous inference rules by adding *variables, expressions, quantified predicates, equality*. The recursive definition of an expression is: an expression is either a variable or an expression such as $A \mapsto B$, where A and B are expressions. The quantified predicate is written in the form of $\square x \cdot P$, where $\square$ stands for the universal quantifier $\forall$ or the existential quantifier $\exists$, x stands for *a non-empty list of variables*, P stands for a *predicate*. The notation $[x := E]P$ stands for the instantiation of the quantified variable x by expression E in the predicate P.

$$\frac{H, \forall x \cdot P, [x := E]P \vdash Q}{H, \forall x \cdot P \vdash Q}\ \textbf{ALL\_L} \qquad \frac{H \vdash P}{H \vdash \forall x \cdot P}\ \textbf{ALL\_R}\ (x\ not\ free\ in\ H)$$

$$\frac{H, P \vdash Q}{H, \exists x \cdot P \vdash Q}\ \textbf{XST\_L}\ (x\ not\ free\ in\ H\ and\ Q) \qquad \frac{H \vdash [x := E]P}{H \vdash \exists x \cdot P}\ \textbf{XST\_R}$$

$$\frac{H \vdash \exists x \cdot Q \quad H, Q \vdash P}{H \vdash \exists x \cdot P}\ \textbf{CUT\_XST}\ (x\ not\ free\ in\ H)$$

$$\frac{[x := F]H, E = F \vdash [x := F]P}{[x := E]H, E = F \vdash [x := E]P}\ \textbf{EQ\_LR} \qquad \frac{[x := E]H, E = F \vdash [x := E]P}{[x := F]H, E = F \vdash [x := F]P}\ \textbf{EQ\_RL}$$

Rules	Definitions
ALL_L	If we can prove a sequent with a quantified predicate $\forall x \cdot P$ and an assumption $[x := E]P$ in the hypotheses, then the sequent without the assumption $[x := E]P$ is proved.
ALL_R	To prove a sequent with a quantified predicate $\forall x \cdot P$ as its goal, it is enough to prove a similar sequent without quantifier "$\forall$" in the goal
XST_L	If we can prove a sequent with a predicate P in the hypotheses, then we can prove a sequent with an instantiation of variable x in P.
XST_R	If a goal of a sequent is obtained by an instantiation of a quantified variable x in predicate P, then we say the sequent with a goal which says $\exists x \cdot P$ is proved.
CUT_XST	This is a derived rule from **CUT, XST_L** and **XST_R**.
EQ_LR	This inference rule applies the equality assumption from left to right in the remaining hypotheses and goal.
EQ_RL	This inference rule applies the equality assumption from right to left in the remaining hypotheses and goal.

Table 2.3: Predicate Logic Extension Inference Rules

2.1.3 Set Theory

Event-B uses first order logic and set-theoretical notation to define the constants, relations and data structures used in models. The notation of Event-B follows the classic set-theoretical notation such as *cartesian product* : $E \mapsto F \in S \times T$, *power set* : $E \in \mathbb{P}(S)$, *set comprehension* : $E \in \{x \cdot P|F\}$, *set equality* : $S = T$, in which E and F stand for expressions; S and T stand for sets; P stands for a predicate. The set comprehension says that E is the set of elements in F that satisfy the predicate P.

We list some of the binary relation operators and their definitions in Table 2.4. The function syntax is defined in Table 2.5.

Name	Syntax	Definition
All binary relations	$r \in S \leftrightarrow T$	$r \subseteq S \times T$
Domain	$E \in dom(r)$	$\exists y \cdot E \mapsto y \in r$
Range	$E \in ran(r)$	$\exists x \cdot x \mapsto E \in r$
All total relations	$r \in S \nleftrightarrow T$	$r \in S \leftrightarrow T \wedge dom(r) = S$
All surjective relations	$r \in S \leftrightarrow\!\!\!\!\rightarrow T$	$r \in S \leftrightarrow T \wedge ran(r) = T$
All surjective and total relations	$r \in S \nleftrightarrow\!\!\!\!\rightarrow T$	$r \in S \leftrightarrow\!\!\!\!\rightarrow T \wedge r \in S \nleftrightarrow T$
Converse	$E \mapsto F \in r^{-1}$	$F \mapsto E \in r$
Domain restriction	$E \mapsto F \in S \lhd r$	$E \in S \wedge E \mapsto F \in r$
Range restriction	$E \mapsto F \in r \rhd T$	$E \mapsto F \in r \wedge F \in T$
Domain subtraction	$E \mapsto F \in S \lhd\!\!\!\!- r$	$\neg E \in S \wedge E \mapsto F \in r$
Range subtraction	$E \mapsto F \in r \rhd\!\!\!\!- T$	$E \mapsto F \in r \wedge \neg F \in T$
Relational image	$F \in r[U]$	$\exists x \cdot x \in U \wedge x \mapsto F \in r$

Table 2.4: Binary Relation Operators

Name	Syntax	Definitions
All partial functions	$f \in S \nrightarrow T$	$f \in S \leftrightarrow T \wedge (f^{-1}; f) \subseteq id$
All total functions	$f \in S \rightarrow T$	$f \in S \nrightarrow T \wedge S = dom(f)$
All partial injections	$f \in S \rightarrowtail T$	$f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S$
All total injections	$f \in S \rightarrowtail T$	$f \in S \rightarrow T \wedge f^{-1} \in T \nrightarrow S$
All partial surjections	$f \in S \twoheadrightarrow T$	$f \in S \nrightarrow T \wedge T = ran(f)$
All total surjections	$f \in S \twoheadrightarrow T$	$f \in S \rightarrow T \wedge T = ran(f)$
All bijections	$f \in S \rightarrowtail\!\!\!\!\twoheadrightarrow T$	$f \in S \rightarrowtail T \wedge f \in S \twoheadrightarrow T$

Table 2.5: Function Operators

2.2 Event-B Notation

2.2.1 Components and Relations

As we noted in the introduction to this thesis, a typical Event-B model has two kinds of components: *machines* and *contexts*. A context describes the constant part of a model, namely *carrier sets and, constants, together with axioms* and *theorems stating their properties*. A machine defines the dynamic part of a model, namely *variables, invariants, theorems, variants* and *events*.

Terms such as *refines, extends, sees* are used to describe the relations between components of an Event-B model. The meaning of the relations are defined as follows:

- A machine *refines* another machine means that the new machine is a more concrete version of the old one;
- A context *extends* another context means that the new context includes all of the content of the previous one;
- A machine *sees* a context means that the machine can use all the sets and constants in the context;
- A machine can refine only one machine or no machine at all;
- A context can extend more than one (including one) or no context at all;
- A machine can see more than one (including one) or no context at all;
- The extension of contexts is transitive (if C1 extends C2, then C1 extends all the contexts extended by C2);

- The "sees" relation is also transitive (if M1 sees C1, then M1 sees all the contexts extended by C1).

Figure 2.1 gives an example to illustrate the relations between machines and contexts. The relations between the components of the model satisfy the rules presented

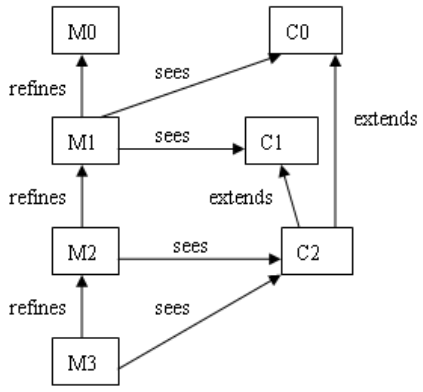

Figure 2.1: Contexts and Machines Relation Example

above. From figure 2.1, we see M0 is an abstract machine (refines no machine) and does not see any context. C2 extends both C1 and C0. Note that the "sees" relation between M3 and C2 is implicated, because M2 sees C2; M3 refines M2 and there is no extension for C2.

2.2.2 Context and Machine Notation

Context

A context is the static part of the model. A typical notation for a context is illustrated as follows. The terms in uppercase are predefined in the Event-B tool Rodin Platform.

CONTEXT
<context name>

EXTENDS
    <context list>
SETS
    <set list>
CONSTANTS
    <constants list>
AXIOMS
    <label>: <predicate>
    ...
THEOREMS
    <label>: <predicate>
    ...
END

All the elements such as the carrier set, the constants, the axioms and the theorems, are implicitly included in the extending contexts. The axioms in a context are some predicates which play the role of the hypotheses in HYP (see section2.1.2). The axioms can be directly used as hypotheses in all proof obligations (POs) without being proved (i.e.,they are assumptions). The *labels* are the identifiers of the predicates. In contrast, the theorems need to be proved before being used as hypotheses in POs. The theorems are some predicates which are derived from the axioms or some already proved predicates. We generally derive some theorems to make the discharging of POs easier, using them as auxiliary assumptions.

Machine

We present the general structure of an Event-B machine. A machine has a format similar to a context and the machine is the main mechanism for describing the system behaviours.

MACHINE
    <machine name>
REFINES
    <machine name>

SEES
    <context list>
VARIABLES
    <variable list>
INVARIANTS
    <label>: <predicate>
    ...
VARIANT
    <variant>
EVENTS
    <event list>
END

The invariants are some predicates which describe the invariant properties of the state transitions (event triggering) in the machine. A special kind of invariant, which is called a *gluing invariant*, connects the current machine and the refined machine. The intension of introducing gluing invariants is to maintain the refinement consistency. The theorems are predicates that have to be proved about the machine. The variant only appears when a machine includes a *convergent event* (see section 2.2.2). The next section indicates the notation of *Event* in detail.

Event

An event describes a transition between the states of the machine.

Event $\widehat{=}$
    <event name>
Status {ordinary, convergent, anticipated}
refines
    <event list>
any
    <abstract parameter list>
where
    <label>: <predicate>
    ...
with
    <label>: <witness>
    ...

then
    <label>: <action>
    ...
end

The status of an event can be ordinary, convergent (the event decreases the variant) or anticipated (the event does not increase the variant). The any indicates the *abstract parameters* in an event. The abstract parameters are those variables that will have to be refined. In practical examples, people use abstract parameters as the input variables. The "where" clause indicates the guards of the event, which are the trigger conditions for some actions. The witnesses of an event are contained in the "with" clause. A witness is always used to indicate those abstract parameters which disappear when an abstract event is refined to a concrete one. The actions of an event are contained in the "then" clause.

2.3 Proof Obligation Rules

Proof obligations are some sequents which are automatically generated by the Rodin Platform tool using existing proof obligation rules. Before we present various proof obligation rules, we define the general notation of an event from [Abr10]:

Event $\widehat{=}$
any
    x
where
    $G(s, c, v, x)$
then
    $v :\mid BA(s, c, v, x, v')$
end

where s stands for the carrier sets, c the constants, v the variables, x the abstract parameters, and v' the substituted variables. G stands for the guards of the event, while

$v :| BA(s, c, v, x, v')$ indicates actions of the event. The notation $BA(s, c, v, x, v')$, called the before after predicate in Event-B, describes the state change after the actions occur. The axioms and theorems are denoted by $A(s, c)$, the invariants by $I(s, c, v)$, and the gluing invariants by $J(s, c, v, w)$ (w stands for the concrete variables[2]).

2.3.1 Invariant Preservation Rule

The invariant preservation rule (INV) says each event shall satisfy the invariants in the machine. The generated POs derived using the INV rule are always denoted by "event name/ invariant name/ INV". The INV rule is written as the following sequent:

$$A(s, c), I(s, c, v), G(s, c, v, x), BA(s, c, v, x, v') \vdash inv(s, c, v')$$

where the $inv(s, c, v')$ is the modified invariant after substitution for variable v.

2.3.2 Feasibility Rule

Event-B supports nondeterministic actions which are written in the form of $v :| BA(s, c, v, x, v')$. The nondeterministic assignment says variable v is assigned a value which satisfies predicate $v :| BA(s, c, v, x, v')$. The feasibility rule (FIS), which denoted by "event name/ action label/ INV", is needed to make sure that the nondeterministic action assigns the variable v an existing value satisfying the before after

[2]Concrete variables are those variables which are refined through the abstract variables.

predicate BA. The FIS rule is written as the following sequent:

$$A(s,c), I(s,c,v), G(s,c,v,x) \vdash \exists v' \cdot BA(s,c,v,x,v')$$

2.3.3 Guard Strengthening Rule

A refined model should satisfy the requirements of the abstract model, so if the event in the abstract model is enabled, then the corresponding concrete events in the refined model shall do the same; guards enable or disable the actions in events. The guard strengthening rule (GRD) ensures that the guards in the refined event are stronger than those in the abstract event. The GRD rule is named "event name/ guard label/ GRD". Suppose an abstract event and a refined event have the following format:

Event *event0* $\widehat{=}$
any
x
where
$g(s,c,v,x)$
...
then
...
end

Event *event1* $\widehat{=}$
refines
*event*0
any
y
where
$H(y,s,c,v,x)$
with
$x : W(x,s,x,c,w,y)$
then
...
end

Then the GRD rule is written as the following sequent:

$$A(s,c), I(s,c,v), J(s,c,v,w), H(y,s,c,w), W(x,s,c,w,y) \vdash g(s,c,v,x)$$

where $W(x,s,c,w,y)$ is the witness predicate, in which the abstract parameters x and y are different.

2.3.4 Simulation Rule

The simulation rule (SIM) ensures that the actions in the refinement simulate the actions in the corresponding abstract machine. Suppose we have two events:

Event $event0 \mathrel{\widehat{=}}$
any
x
where
...
then
$v : |BA1(s, c, v, x, v')$
end

Event $event1 \mathrel{\widehat{=}}$
refines
$event0$
any
y
where
$H(y, s, c, w)$
with
$x : W1(x, s, c, w, y, w')$
$v' : W2(v', s, c, w, y, w')$
then
$w : |BA2(s, c, w, y, w')$
end

The above events present a general format of events, both of which include abstract parameters. Because the abstract parameters x and y are different, while the variables v and w are also different, we have two witness predicates $W1$ and $W2$. The SIM rule which is written in the form "event name/ action label/ SIM", has the sequent:

$$A(s, c), I(s, c, v), J(s, c, v, w), H(y, s, c, w), W1(x, s, c, w, y, w'),$$
$$W2(v', s, c, w, y, w'), BA2(s, c, w, y, w') \vdash BA1(s, c, v, x, v')$$

The $BA1$ and the $BA2$ stand for the before after predicates of the variables in the abstract event and in the concrete event respectively.

2.3.5 Witness Feasibility Rule

The witness feasibility rule (WFIS) ensure the existence of the witness predicate in a refined event. The rule is written in the format "event name/ abstract parameter name/ WFIS". Suppose we have a refined event:

Event $\widehat{=}$
any
y
where
$H(y, s, c, w)$
with
$x : W(x, s, c, w, y, w')$
then
$BA(s, c, w, y, w')$
end

The PO sequent is presented as follows:

$$A(s, c), J(s, c, v), H(y, s, c, w), BA(s, c, w, y, w') \vdash \exists x \cdot W(x, s, c, w, y, w')$$

2.3.6 Well-definedness Rule

The well-definedness rule (WD) ensure that all the axioms (axm/ WD), theorems (thm/ WD), invariants (inv/ WD), guards (grd/ WD) and actions (act/ WD) are well-defined. There are lots of well-definedness conditions for some mathematical expressions. We list several of them in table 2.6:

Expressions	Well-definedness condition
E/F	$F \neq 0$
$EmodF$	$E \geq 0 \wedge F > 0$
$card(S)$	$finite(S)$
$min(S)$	$S \neq \varnothing \wedge \exists x \cdot (\forall n \cdot n \in S \implies x \leq n)$
$max(S)$	$S \neq \varnothing \wedge \exists x \cdot (\forall n \cdot n \in S \implies x \geq n)$

Table 2.6: Well-definedness Rules

Chapter 3

The Insulin Infusion Pump

This chapter describes the system behaviours and the functionalities of the IIP, based on the requirement document [FDA10] from FDA. There are a variety of insulin infusion pumps. The FDA document describes a generic IIP which contains some essential features of an IIP. To make our motivation clear, we omit some extendable features such as bolus correction and food bolus calculation. (Detailed requirements can be found in [FDA10].) Therefore, in this thesis we present a generic IIP, which includes all the basic, common functionalities for different kinds of IIPs. The main functionalities of the IIP are the programming and delivery of basal, temporary basal, normal bolus and extended bolus insulin. Note that the description presented below may contain some mistakes or inconsistencies, which are exactly those defects we are looking for through specifying the IIP system in Event-B and discharging POs generated from the Event-B tool (Rodin Platform).

3.1 Components

The IIP structure is illustrated by the following simple figure 3.1 and we give a brief explanation for each of the parts.

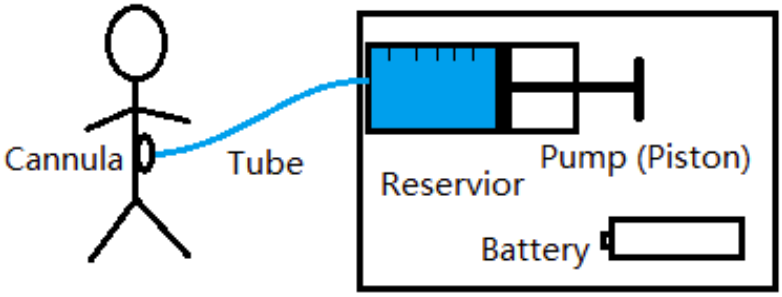

Figure 3.1: IIP

Reservior:

The reservoir is the place where insulin is stored. The amount of insulin in the reservoir can support several days usage, the time of course depending on usage.

Tube:

The insulin runs through the tube from the device to the user.

Cannula:

The cannula connects the patient and device at the end of the tube on the patient side.

Pump:

The piston (may another kind of pump) in the pump pushes the insulin in the reservoir through the tube, consequently delivering the insulin to the patient.

Control panel:

The control panel accepts commands from and sends visible information to the users.

Battery:

Battery is not included in the IIP models.

Sensors:

The battery sensor detects the amount of electricity left in the battery.

The reservoir sensor measures the insulin left in the reservoir.

Flow rate sensor detects the flow rate in the tube.

Alarms:

The battery low alarm, which will be on when the battery is low.

The battery empty alarm, which will be on when the battery is discharged.

The reservoir low alarm, which will be on when the insulin in the reservoir is low.

The reservoir empty alarm, which will be on when the reservoir is empty.

Controller:

The software part of the device, which interacts with other components of the device, sequentially controls the pumping process, notifies the user about the insulin that is delivered in real-time and handles exceptions. This part is the core of the whole device and the controller is what we are going to design and implement (our model also focuses on the controller).

3.2 Commands and Actions

The following commands can be performed by manipulating the buttons (either abstract or concrete) on the control panel.

Cmd-1 Power on (Turn on the IIP.)

Cmd-2 Power off (Turn off the IIP.)

Cmd-3 Prime on (This command is used to avoid air-in-line problems with the tube. The prime on start command is always executed after reservoir refill and before pumping.)

Cmd-4 Prime stop (This command is executed when the priming process is finished; in other words at this time the tube is fully filled by insulin; there is no air left in the tube. This process can be assigned to a command manipulated by the user or an automatic process directed by the controller.)

Cmd-5 Basal profile settings (This is a set of commands programmed by users.)

Cmd-5-1 Basal rate increase (Increase the basal rate.)

Cmd-5-2 Basal rate decrease (Decrease the basal rate.)

Cmd-5-3 Basal rate confirm (Confirm the basal rate.)

Cmd-5-4 Basal rate start time (Select the start time of a selected basal rate.)

Cmd-5-5 Basal rate duration increase (Increase the duration (length of time) of a selected basal rate.)

Cmd-5-6 Basal rate duration decrease (Decrease the duration of a selected basal rate.)

Cmd-5-7 Basal rate duration confirm (Confirm the duration of a selected basal rate.)

Cmd-6 Temporary basal settings (This is a set of commands used to temporarily override a period of basal rate delivery in the basal profile. The sub-commands are almost the same as those in **Cmd-5**.)

Cmd-7 Start (Start the IIP.)

Cmd-8 Pause (Pause the processing performed by IIP.)

Cmd-9 Resume (Resume the IIP from the paused state.)

Cmd-10 Stop (Stop the IIP. This command can be executed during the normal bolus or extended bolus delivery process. The priming process should be executed after **Cmd-10** has been executed.)

Cmd-11 Normal bolus settings (This is a set of commands used to set the bolus.)

Cmd-11-1 Normal bolus dosage increase (Increase the dosage of bolus.)

Cmd-11-2 Normal bolus dosage decrease (Decrease the dosage of bolus.)

Cmd-11-3 Normal bolus dosage confirm (Confirm the dosage of bolus.)

Cmd-11-4 Normal bolus start (Start the confirmed bolus.)

Cmd-12 Extended bolus settings (This is a set of commands used to set the extended bolus.)

Cmd-12-1 Extended bolus rate increase (Increase the rate of extended bolus.)

Cmd-12-2 Extended bolus rate decrease (Decrease the rate of extended bolus.)

Cmd-12-3 Extended bolus rate confirm (Confirm the rate of extended bolus.)

Cmd-12-4 Extended bolus start time (Specify the start time of the extended bolus.)

Cmd-12-5 Extended bolus duration increase (Increase the duration of extended bolus.)

Cmd-12-6 Extended bolus duration decrease (Decrease the duration of extended bolus.)

Cmd-12-7 Extended bolus duration confirm (Confirm the duration of extended bolus.)

Cmd-13 Alarm off (Turn off any kind of alarms.)

Besides the commands programmed by the users through the control panel, there

are some actions performed by the users which are not related to the controller but must be done.

Act-1 Equip (change) the battery.

Act-2 Equip (refill) the reservoir.

Act-3 Connect the IIP to the body (of the user).

Act-4 Disconnect the IIP from the body (of the patient).

3.3 Interaction Behaviour

In this section, we describe the interaction between users and the device from the very beginning to completion for a typical one day usage of the IIP. Commands and actions described in the last subsection are shown in a sequence in this subsection to exhibit their priority. Note that we suppose that the initial state of the power of the device is off, the reservoir and battery are unequipped and the insulin pump is disconnected from the users' body.

1. Equip or change the battery (**Act-1**).

2. Equip or change the reservoir (**Act-2**).

3. Turn on the device (**Cmd-1**) or return to step 1 if the battery is low or empty.

4. Turn off the device (**Cmd-2**) or return to step 2 .Turn on the device will detect the amount of insulin left in the reservoir. If an empty reservoir is detected step 2 should be taken.

5. Prime the IIP (**Cmd-3**). The priming process pushes the air out from reservoir and tube.

6. Stop the priming process (**Cmd-4**).

7. Equip the pump to the patient's body (**Act-3**).

8. Set the basal profile (**Cmd-5**).

8.1 Increase the basal rate (**Cmd-5-1**) or decrease it (**Cmd-5-2**).

8.2 Confirm the basal rate (**Cmd-5-3**).

8.3 Set the start time of the selected basal rate (**Cmd-5-4**).

8.4 Increase the duration of the basal rate (**Cmd-5-5**) or decrease it (**Cmd-5-6**).

8.5 Confirm the stop time of the basal rate (**Cmd-5-7**) and repeat through step 8.1 to step 8.5 until all the basal rates are set.

9. Start to deliver the basal rate (**Cmd-6**).

10. Set the temporary basal rate (**Cmd-6**). The temporary basal rate can override the existing basal rate. The sub commands for this process are almost the same as those in step 8.

11. Pause the insulin pump (**Cmd-8**). This step can happen at any time during the basal delivery.

12. Resume the IIP from a suspension state (**Cmd-8**).

13. Set the normal bolus (**Cmd-11**).

13.1 Increase (**Cmd-11-1**) or decrease (**Cmd-11-2**) the normal bolus dosage .

13.2 Confirm the dosage of normal bolus (**Cmd-11-3**).

13.3 Start the process of normal bolus (**Cmd-11-4**) and the process of normal bolus will be finished automatically. The normal bolus delivery can be stopped (**Cmd-10**).

14. Set the extended bolus (**Cmd-12**).

14.1 Increase (**Cmd-12-1**) or decrease (**Cmd-12-2**) the rate of the extended

bolus.

14.2 Confirm the rate or dosage of the extended bolus (**Cmd-12-3**).

14.3 Set the start time of the extended bolus (**Cmd-12-4**).

14.4 Increase (**Cmd-12-5**) or decrease (**Cmd-12-6**) the duration of the extended bolus.

14.5 Confirm the duration of the extended bolus (**Cmd-12-7**), which means the stop time of the extended bolus is set. In step 13.3, the extended bolus can be stopped (**Cmd-10**).

15. Pause the IIP (**Cmd-8**); makes sure all the pumping activations are stopped.

16. Disconnect the device from the user (**Act-4**).

17. Return to step 5 if the disconnection of the device to the user in not because of a dying battery or empty reservoir. If the disconnection was the result of changing the battery and/or the reservoir, return to step 1 or step 2.

18. Resume the IIP (**Cmd-9**) if the pause or disconnection is not the result of a failure.

19. Turn off the IIP (**Cmd-2**).

Note that step 11 can be executed at any time after the basal rate has been set, because the battery being empty or the reservoir being empty will trigger the pump to stop (this will be mentioned in next section, the system requirements description) and the patient can press the pause button during the basal delivery. Step 19 can be executed at any time. Note, although command 13 Alarm off is not mentioned here, it can be administered by the user when any alarm is on.

3.4 Requirements of IIP

A rational process [DLPPCC86] of software design should begin from the requirements document. Requirements always contain some "black box" specifications of the system on an abstract level.

Because Event-B models can be refined through different abstraction levels, we can either create the model on a very abstract level and consequently prove its adequacy or refine the model to as concrete a level as the code and accordingly prove the correctness of the code. Our project is mainly based on the requirements published by the FDA. The main content of these requirements are on an abstract level, but some details of the system behaviour are still included. An Event-B model can both handle abstraction and the introduction of implementation detail. Although, as we will observe, this draft requirements description contains some mistakes and incompleteness aspects, it is the starting point of the whole project. The requirements present a large number of constraints on the system, either on software or hardware. This thesis is focused on the software and our interest is to detect and correct the faults and incompleteness appearing in the requirements using a formal modelling approach. Because [FDA10] is a confidential document, we only exhibit parts of the requirements from it and create a re-description of it. The rewritten requirements document can be found in Appendix A. One issue we have to note here is that [FDA10] uses unconstrained values to represent some undetermined constant values, but in the next chapter we will show that constraints are essential for these constant values.

Chapter 4

Modelling the IIP

4.1 Refinement Strategies

This chapter presents the modelling process of IIP and is the main contribution in this thesis. We made some extensions to an existing time constraint pattern and apply the new patterns to the IIP model. From the model analysis and all the proof obligations being discharged, we detected several inconsistencies and missing properties from the requirements document.

As we described in Chapter 2, Event-B is a refinement based specification language; the models are refined step by step, both to develop the abstract specification itself, as well as from an abstract level to a concrete level. (We will not actually embark on an implementation, so we will ignore the latter use of refinement.) Most of the time, the abstraction (the initial model) directly affects the difficulties of creating the model. An improper initial model may result in an unreasonable refinement or increase the difficulty of doing refinement. Therefore, a good refinement strategy is important during the modelling process. Unfortunately, there does not exist any

standards related to refinement strategies in Event-B. Engineers always establish the refinement strategy by using their Event-B experience.

Below are some of our own understanding of the refinement strategies.

Program or Algorithm: For a simple program or an algorithm, the initial model shall focus on the purpose of the algorithm. Firstly, we can determine the output of the algorithm and focus the initial model on the final state. In the initial model we can temporarily write the algorithm body as an abstract event. The second stage is to find how the output transits to the final state. The abstract events in the model, which can be refined to several events, represent the body of the algorithm. In this stage, there always exist some events in pairs. For example, we want to find out the minimum value in an array of numbers. There are several algorithms that can be used to solve this problem. The initial model of different algorithm can be a same initial abstraction. We define the array as a total function. The machine of the model includes a final event which indicates the minimum value in the array and an abstract event which describes how the minimum value was found. The refinement here is focusing on the abstract event. We can introduce two variables to indicate the index variation when the algorithm searches for the minimum value in the array. The initial event assigns these two variables to the start index and the end index. The abstract event is refined into two events: one describes the increasing of the start index by comparing corresponding elements for the two indices, the other describes the decreasing of the end index by means of the same comparison. When the two variables are equal to the same value, the minimum value is found in the array.

The strategy of specifying a system is different from specifying a small piece of program.

Interaction Systems: Suppose the purpose of the system is to handle the interaction between serval components. The components can include the users (human beings). The strategy for creating the model for this kind of system is not only focusing on the interaction between agents but also focusing on the abstraction of the components. The refinement stage could be an abstract component decomposition process. The relation between concrete components could be introduced in this stage. We use a bank transaction as an example; our initial model can simply abstract the bank accounts as an agent and the users as an agent. A relation between users and bank accounts should be introduced. To describe the balance changing in the bank account for the account holder and the actions on the user side, the events could be modelling the cash transaction between the two components. During refinement, we could refine the two components into sub components in terms of their features. For the bank account, there are several sub accounts, such as savings account, checking account. The events between the sub accounts can be interpreted as money transfer between sub accounts, but the total amount is stable. On the user side, the users can be refined into several components, which describe the user information, such as id, contact information. Each of the above pieces of information can be a concrete component. When we refine the transaction events between the users and the accounts, for each piece of basic user information, a check must be made through the relation between users and the sub components. Therefore, when modelling the interactive system, one of the refinement strategies is to focus on the abstraction of the components and their relations.

Control Systems: For control systems, the purpose is to monitor and to control some phenomenon. There are two solutions for specifying the control system.

• Solution One:

To create the initial model, we have to find out the control goal of the system and then list the possible input and output variables used for describing the control goal. The next step is to make a comparison between the input and output of the controller and then find some key variables (abstract variables) to describe the phenomenon. Our preference is to start from the outputs, because these outputs are directly related to the control goal. From the outputs we can find the corresponding relevant input variables. In other words, the initial model describes the core feature of the system through the state change producing output variables. In the refinement stage, we can refine the existing events to indicate how these outputs are produced. The input will be introduced, but in an abstract way in terms of *abstract parameters* in Event-B. At the next refinement stage, we shall describe why these inputs should be included and how these inputs are transformed to the output. We can refine the model by adding environmental variables: monitored variables and controlled variables. Some new environment events which describe how the monitored variables are transformed to input variables and how the output is transformed to controlled variables may be introduced into the model at this stage. The connection between the environment and the controller are described by *gluing invariants* (Event-B terminology).

• Solution Two:

The core of solution two is that all the variables are interpreted as monitored variables, controlled variables, input variables and output variables[PM95] (the same classification as those in solution one). Solution two also starts from the control goal. The difference is that solution two emphases the introduction of the output variables

and the controlled variables on the same abstract level. The state change of environment variables [1](controlled variables) and the controller variables (output variables) are specified in separate events. The connection between controlled variables and output variables should be specified on this level as well. The refinement stage could introduce details of how the value of controlled variables and output variables are produced, which is the same as solution one. This stage will also introduce the monitored variables and input variables.

The above two solutions are exemplified some practical examples. Examples such as a traffic lights controller and a location access controller, found in J. R. Abrial's book [Abr10], use solution one, which introduces the controller variables first and then introduces the environment variables in a later refinement. Solution two follows the Four-Variable model of Parnas and Madey [PM95]. The typical examples are a tank water level controller specified in Michael Butler's paper [But09b] and a mechanical press controller in the Event-B book [Abr10].

Based on the above discussion, although both solutions are used in specifying control systems, the resulting models will both include the environment and controller descriptions. We can see the emphases of these two solutions are a little bit different. Solution one emphasizes the control process. In other words, it starts from describing how the controlled variables are produced. Solution two emphasizes the synchronization of the controller and the environment.

In the early phase of modelling the IIP in Event-B we are interested in the flow rate control of the pump, so we decided to use solution one to create the model. Using solution one on the IIP model, we need to make an abstraction of the core input or output variables. The following sections propose the refinement steps which follow

[1]Environment variables and controller variables are the terms used in Event-B.

the idea in solution one. Another interesting topic is illustrated in this chapter. A time concept is not included in Event-B, so we should find some appropriate solutions for dealing with timing issues we meet in this time relevant system in a way that does not change the semantics of Event-B. Fortunately, some time constraint patterns [Reh06][CMR07] specified in Event-B have already been proposed by some Event-B experts. To handle some timing issues occurring in our project, we made some necessary modifications and extensions to the existing patterns. The new patterns can easily handle the timing issues encountered in the IIP model. We hope that the new pattern can also be applied to similar time relevant control systems.

4.2 System Structure Review

From the description in Chapter 3, we have created figure 4.1 to intuitively illustrate the structure of the IIP on an abstract level.

This description is constructed from two parts: a software part and a physical part. The internal functional variables depend on the environment variables [But09a]. The information flow (represented by continuous line arrows), physical effect (represented by fine dashed line arrows) and user actions (represented by dashed line arrows) are tagged by numbers. Note that this structural graphic illustration is a brief description of the system; not all the information is included. Through the refinement process we will go further and excavate more details of the IIP.

The description of the arrows is:

Information flow

1. Information, including visual or audible warning or alarm, interaction guides and feedback.

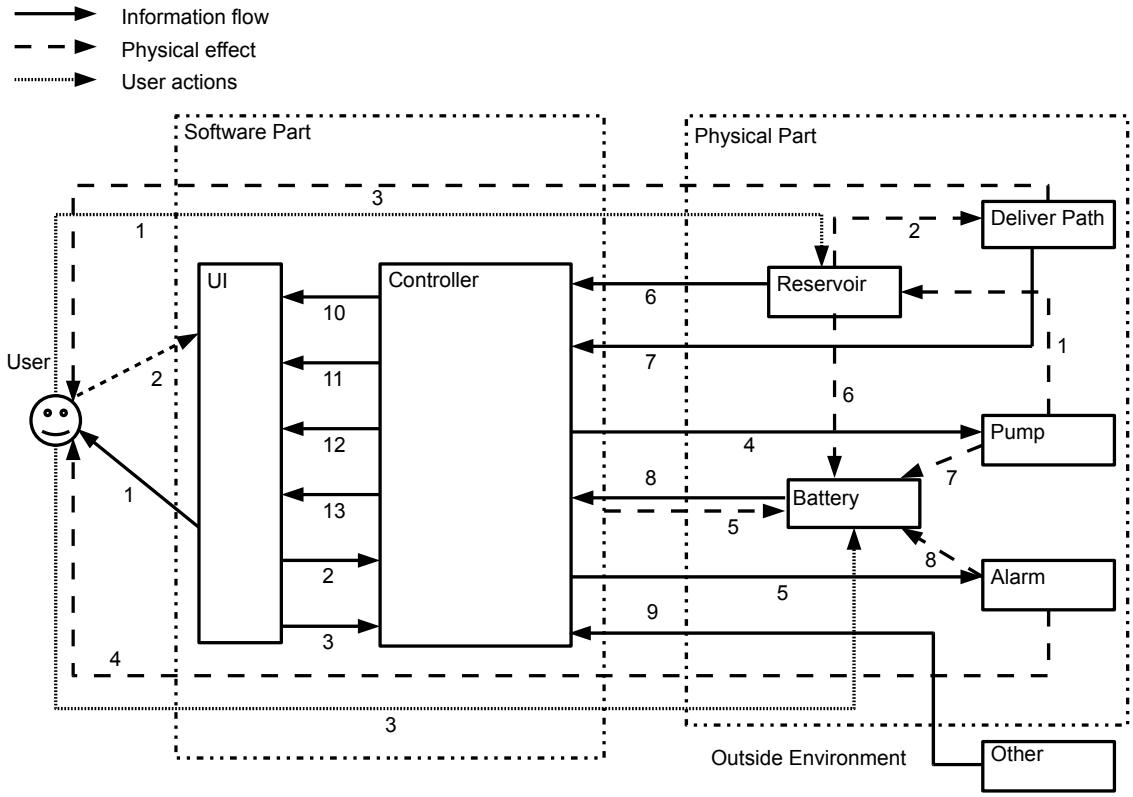

Figure 4.1: Structure of the IIP

2. Programmed variables, which includes the basal and bolus settings, pausing and resuming the pump.

3. Alarm confirmation. The user confirm the system warnings, alerts or alarms.

4. The controller sends commands such as on, off and flow rate to the pump (physical pump).

5. Turn on or turn off message sent from the controller to the alarm.

6. Sensor sensing the amount in the reservoir.

7. Sensor sensing the flow rate in the delivery path.

8. Battery amount.

9. Environment information, which is outside the whole device, such as humidity, temperature, air pressure and so on.

10. Battery level.

11. Reservoir amount. The reservoir volume should be updated after each pump stroke. (requirements 4.2A.4, 4.3A.4)

12. Flow rate or insulin progress. The system shall display the amount of insulin the patient gets. (requirements 3.8A.3)

13. Various alarms, warnings or alerts.

Physical effect

1. Piston in pump pushes the reservoir.

2. Insulin delivered to the delivery path. Here we omit the infusion apparatus, which is not related directly to our system.

3. Insulin delivered through delivery path and arriving at the patient side of the IIP.

4. Tactile alarm to the user.

5. 6. 7. 8. The consuming of the battery by the whole software part, reservoir sensor,

pumping process, alarm respectively.

User actions

1. Change or equip reservoir.

2. Interaction with UI, consequently programming the sending commands to the software controller.

3. Change or equip battery.

As presented before, the intention of this thesis is to create a model focused on the control process of the system. Therefore, for now we only focus on the controller part of the system. From the perspective of the refinement strategy described in the last section, we see this figure 4.1 indicates an intermediate abstraction of the system. There will be a more abstract level above the abstraction level represented in this figure.

4.3 Initial Model

4.3.1 Model Description

The initial model is the highest abstraction of the system. From the requirements, we see the dynamic activities of the system (events) include two phases: the preparation phase and the insulin infusion phase. The preparation phase includes the power on/off, pump priming, input settings (user programmed variables). The insulin infusion phase is the main activation phase of the system. To create the initial model of the IIP, we should understand the control goal of IIP. The control goal is to simulate the behaviour of the pancreas (continuously or intermittently infusing insulin from

an insulin reservoir into the patient). Because the requirements from the FDA do not include the system design documents and the separation between hardware and software is not clarified, we made an assumption about the main output variables from the controller to the physical pump, namely the combined insulin flow rate. In other words, the output of the controller is the amount of insulin to be infused per minimum system time unit[2].

4.3.2 Formalizing the States

The state of any Event-B model has two parts: the static part and the dynamic part. The context defines the static state (carrier sets, constants) and the corresponding properties of the static state, by means of *axioms* or *theorems*. The context of our initial model is denoted by **C0**, which contains an enumerated set called *state* (consisting of *on* and *off*). $\mathbb{N}$ is a default context set in Event-B, so we do not need to write it in our context.

CONTEXT C0
SETS
`state`
CONSTANTS
`on`
`off`
AXIOMS
`axm1` : $state = \{on, off\}$
`axm2` : $on \neq off$

VARIABLES
`rate`
`power`
INVARIANTS
`inv1` : $rate \in \mathbb{N}$
`inv2` : $power \in state$
`inv3` : $power = off \Rightarrow rate = 0$

Because determining whether the system is running depends on the state of the power supply, on this level we introduce two variables (dynamic part of the state),

[2]Because Event-B does not support floating point numbers, to accurately indicate the value of variables we avoid using division in the model. All the flow rates, which have type $\mathbb{N}$, can be interpreted as the insulin amount per minimum system time unit.

rate (combined flow rate) and *power* (the power state of the whole system). The properties of these variables are given by *invariants*. *inv1* and *inv2* indicate the type of *rate* and *power*. The invariants can also be refined in later refinement stages in terms of *gluing invariants*, so on this abstract level we simply say the type of *rate* is a natural number. In the further refinement we can restrict its range if appropriate. *inv3* indicates a property of the abstract model. When the power state is off, the combined flow rate shall be zero. The reason why we use implication in one direction is that there exists a situation when the power is on, but the infusion process is paused, which means the combined flow rate is equal to zero at this moment.

4.3.3 Formalizing the Events

The event part describes the behaviour of the system. There are four events which indicate the state transitions in this abstract model. The *POWER_ON* event and the *POWER_OFF* events just simply describe the power state changes of the system. The *INFU_START* and the *INFU_PROC* indicate the main functional behaviour of the system. Notice that an abstract parameter *ready*, which has type *BOOL* is introduced in the *POWER_ON* event.

Event *POWER_ON* $\widehat{=}$
  when
    grd1 : $power = off$
  then
    act1 : $power := on$
  end
Event *INFU_START* $\widehat{=}$
  any
    ready
  where
    grd1 : $ready \in BOOL$
    grd2 : $ready = TRUE$
    grd3 : $rate = 0$
    grd4 : $power = on$
  then
    act1 : $rate :\in \mathbb{N}_1$
  end
Event *INFU_PROC* $\widehat{=}$
  when
    grd1 : $power = on$
  then
    act1 : $rate :\in \mathbb{N}$
  end
Event *POWER_OFF* $\widehat{=}$
  when
    grd1 : $power = on$
  then
    act1 : $power := off$
    act2 : $rate := 0$
  end

The abstract parameter *ready* will be refined into several events which describe the preparation phase in the next refinement. We use *ready* to abstract the states where the system is getting ready to infuse insulin: *TRUE* means ready; *FALSE* means unready. The *act* (actions) are triggered by the *grd* (guards). The *INFU_START* says when the system is at a ready stage, the *power* is *off* and the *rate* is *0*, the combined flow rate *rate* is assigned as a positive integer value. Notice that because this is an abstraction of the model, the $:\in$ symbol, which stands for unpredictable assignment, is used to handle the nondeterministic value of the combined flow rate. To describe the flow rate changing during the process and for the further refinement, we introduce another event, *INFU_PROC*. The flow rate can be modified to any value in the natural numbers, but the precondition is that the state of power is *on*.

Initialisation
begin
act1 : $rate := 0$
act2 : $power := off$
end

Although the events for describing the pumping process have been presented, the initial state of the model is not defined. The above event provides for the initialization of the model. Because the initialization does not need any trigger condition, there is no guard for this event.

4.3.4 Summary of the Initial Model

This section described the initial model of IIP by following the control goal of the system. The context of the initial model is called **C0** and the initial abstract machine is **IIP0** sees **C0**. Two constant values *on*, *off* and a carrier set *state* are included in

C0. The **IIP0** is constructed using variables: *power, rate* and events: *Initialization, POWER_ON, INFU_START, INFU_PROC, POWER_OFF*.

By running this model in the Rodin Platform, 11 proof obligations (see Appendix B.1.2) are generated and all of them are automatically proved. The full version of the initial model can be found in Appendix B.1.1.

4.4 First Refinement: Refining Phases

4.4.1 Model Description

As we discussed in section 4.3, the behaviour of the system can be abstracted to two phases: the preparation phase and the infusion phase. In this first refinement, we are going to separate these two phases by refining the abstract parameter *ready* in the event *INFU_START* and refining the *INFU_PROC* into several abstract events. Two new events *PRIME* (priming the pump) and *BP_SET* (basal profile setting) are introduced to eliminate *ready*. The *INFU_PROC* event is refined into several sub events which include pausing, resuming and stopping of the infusion process.

4.4.2 Refining the State

The static state (context variables) is extended by an enumerated carrier set *status* and constants *working, paused, stopped* which indicate the status of the infusion process. Two constants *b_2_n, s_2_n*, which represent two total functions intended to convert the non-numeric elements in a set to numbers. From the requirements we see that the pump is enabled for infusing the insulin under the precondition that the pump has been primed and the basal profile has been set. Two variables *prime,*

bp_set are introduced to describe the priming process and basal profile being set in the model. The variable *infu_stat* is an auxiliary variable used to indicate the infusion status.

CONTEXT C1
EXTENDS C0
SETS
`status`
CONSTANTS
`working`
`paused`
`stopped`
`b_2_n`
`s_2_n`
AXIOMS
`axm1` : $status = \{working, paused, stopped\}$
`axm2` : $working \neq paused$
`axm3` : $paused \neq stopped$
`axm4` : $working \neq stopped$
`axm5` : $b\_2\_n \in BOOL \rightarrow \{0, 1\}$
`axm6` : $b\_2\_n(TRUE) = 1$
`axm7` : $b\_2\_n(FALSE) = 0$
`axm8` : $s\_2\_n \in state \rightarrow \{0, 1\}$
`axm9` : $s\_2\_n(on) = 1$
`axm10` : $s\_2\_n(off) = 0$
VARIABLES
`prime`
`bp_set`
`infu_stat`
INVARIANTS
`inv1` : $prime \in BOOL$
`inv2` : $bp\_set \in BOOL$
`inv3` : $infu\_stat \in status$
`inv4` : $infu\_stat = stopped \vee infu\_stat = paused \Rightarrow rate = 0$
`inv5` : $infu\_stat = working \vee infu\_stat = paused \Rightarrow power = on \wedge prime = TRUE \wedge bp\_set = TRUE$

Because this refinement eliminates *ready*, the corresponding event will contain the witness statement:

$$ready : ready = (power = on) \wedge (prime = TRUE) \wedge (bp\_set = TRUE).$$

However, the witness should be equal to a number but not a predicate. A simple solution is to convert non-numeric variables to numeric variables. We defining a total function *b_2_n* from *BOOL* to $\{0, 1\}$ and a total function *s_2_n* from *state* to *0,1*. The corresponding properties are shown in the context *C1*. Then the witness can be expressed as follows:

$$ready : b\_2\_n(ready) = s\_2\_n(power) * b\_2\_n(prime) * b\_2\_n(bp\_set).$$

We use $prime = TRUE$ to indicate the priming process is complete, similarly for *bp_set*. The invariant *inv4* says: when the infusion process has not started or been paused, the combined flow rate shall be equal to zero. *inv 5* says the pump is working or is paused under the circumstance that the preparation stage has been done. In other words, the power should be turned on; the pump should be primed; the basal profile should be set.

4.4.3 Refining the Events

This section refines the delivery process into several sub events and adds a priming process and basal profile setting phase into the model. Some new guards and actions are added to corresponding events[3].

EVENTS
Initialisation
extended
begin
⊕act3 : *prime* := *FALSE*
⊕act4 : *bp_set* := *FALSE*
⊕act5 : *infu_stat* := *stopped*
end
Event *PRIME* ≙
when
grd1 : *power* = *on*
grd2 : *prime* = *FALSE*
then
act1 : *prime* := *TRUE*
end
Event *BP_SET* ≙
when
grd1 : *power* = *on*
grd2 : *bp_set* = *FALSE*
then
act1 : *bp_set* := *TRUE*
end

The above events describe the preparation phase of the system. In *Initialization, prime, bp_set* and *infu_state* are given initial values *FALSE, FALSE and stopped*, respectively. The *PRIME* and *BP_SET* are two straightforward events, which indicate

[3]⊕ indicates the expression is a refined guard or action.

the state change of *prime, bp_set*. We will see in the following refinement that *BP_SET* can be refined to a sequence of events which describe the user input. The following events are the refinement of the infusion phase.

Event *INFU_START* $\widehat{=}$
refines *INFU_START*
when
$\oplus$grd2 : *prime* = *TRUE*
$\oplus$grd3 : *bp_set* = *TRUE*
$\oplus$grd5 : *infu_stat* = *stopped*
with
ready : b_2_n(ready) = s_2_n(power)∗b_2_n(prime)∗ b_2_n(bp_set)
then
$\oplus$act1 : *rate* :$\in \mathbb{N}_1$
$\oplus$act2 : *infu_stat* := *working*
end
Event *INFU_PROC* $\widehat{=}$
refines *INFU_PROC*
when
$\oplus$grd2 : *infu_stat* = *working*
then
$\oplus$act1 : *rate* :$\in \mathbb{N}_1$
end
Event *PAUSE* $\widehat{=}$
refines *INFU_PROC*
when
$\oplus$grd2 : *infu_stat* = *working*
then
$\oplus$act1 : *rate* := *0*
$\oplus$act2 : *infu_stat* := *paused*
end
Event *RESUME* $\widehat{=}$
refines *INFU_PROC*
when
$\oplus$grd2 : *infu_stat* = *paused*
then
$\oplus$act1 : *rate* :$\in \mathbb{N}_1$
$\oplus$act2 : *infu_stat* := *working*
end
Event *INFU_STOP* $\widehat{=}$
refines *INFU_PROC*
when
$\oplus$grd2 : *infu_stat* = *working* $\vee$ *infu_stat* = *paused*
then
$\oplus$act1 : *rate* := *0*
$\oplus$act2 : *infu_stat* := *stopped*
$\oplus$act3 : *prime* := *FALSE*
end
Event *POWER_OFF* $\widehat{=}$
refines *POWER_OFF*
then
$\oplus$act3 : *prime* := *FALSE*
$\oplus$act4 : *bp_set* := *FALSE*
$\oplus$act5 : *infu_stat* := *stopped*
end

INFU_START is refined by adding guards describing that when priming and basal profile setting are done and when the infusion status is *stopped*, then the flow rate shall be assigned as a positive nature number and the infusion status set to *working*. Correspondingly, the *INFU_STOP* just refines *INFU_PROC* by resetting *prime, bp_set* and *infu_stat* values. The reason we do not set *bp_set* back to *FALSE* is that stopping

the infusion process will not delete the existing basal profile set by the user.

As discussed in the last section, the main refinement in this model is to refine the *INFU_PROG* event into four events *INFU_PROC, PAUSE, RESUME, INFU_STOP*. The meanings of *PAUSE* and *RESUME* are easy to understand. The pump pause event only triggers when the current infusion state is *working*, the pump can transit to *paused* state and the combined flow rate *rate* set to *0*. The *RESUME* event has the opposite behaviour to *PAUSE*. The *INFU_PROC* event describes the infusion behaviour except for the case when the pump is paused or stopped. Because the flow rate will be equal to zero only when the pump is paused or stopped, we assign the combined flow rate in *INFU_PROC* as a positive natural number.

4.4.4 Summary of the First Refinement

In this refinement, context **C1** extends **C0**; **IIP1** refines **IIP0** ; **IIP1** sees **C1**. This refinement mainly refines the *INFU_PROG* event to events which describes the pause, resume, stop behaviours of the infusion process; new events for describing the preparation phase are introduced. A small technique for converting non-numeric expression to numeric expressions is used in **C1**. The full version of refinement one can be found in Appendix B.2.1.

By running this refinement in the Rodin Platform, 25 POs (see B.2.2) for **IIP 1** and 4 POs for **C1** are generated and 2 of them need to be discharged manually. All of the other POs are automatically proved by Rodin.

4.5 Second Refinement: Basal Profile Setting

4.5.1 Model Description

From the requirements document, we see that the basal profile can actually be the context for the model if we are only modelling the infusion phase of the system, because the basal profile is just a bunch of data input from the user and their state is not varying during the infusion process. Here we treat the data of temporary basal and basal profile data as independent data. However, in the initial model and the first refinement we have already divided the whole system process into two phases: the preparation phase and the infusion phase. We refine the basal profile setting event into several concrete events.

4.5.2 Refining the State

Before the refinement, let us review the requirements describing the basal profile in [FDA10] requirements document. "The pump shall allow the user to program a basal profile with a set of basal rates, ranging from 0.05 to x (some undetermined maximum value) Units/hour in 0.05 Units/hour increments. For each basal rate in the profile, the user shall define the duration of the particular rate. Durations of all basal rates shall not overlap with each other, and shall together cover 24 hours of a day."

From the requirements we see that the basal profile includes a set of basal flow rate segments. All of these basal segments shall not overlap with each other, but have to cover the whole day. In other words, each minimum time unit during the day has a corresponding basal flow rate. So we can create two carrier set ts, and ba_rs to represent the time set for day time and the set of all possible basal rates. Because

Event-B does not support real numbers, for simplicity we use the natural numbers to describe time approximately. The accurate of the time definitions here depend on the time unit used in the system. For example, if the requirements of system time accuracy is a minute, then the time set ts for one day is 0..1439. If accuracy is seconds, then ts shall be 0..86399. In this refinement, we do not consider the time unit for the system for the moment, so we simply define the day time as a constant set which is a subset of $\mathbb{N}$ (axm1). For the basal rate we define it as a subset of $\mathbb{N}1$ (axm3). From the requirements we see the basal profile shall be in terms of $ts \rightarrow ba\_rs$ (total function from time set to basal rate set). The reason why we are using $\nrightarrow$ (partial function) to describe the basal profile in inv1 is that we only care about the state change when an event triggers. Considering Event-B is an event based specification language, we only need to specify the event when the basal rate is changing. Even if we need the amount of basal insulin that has been delivered, the amount can be calculated from the rate multiplied by the difference between the current time minus the latest basal change time.

CONSTANTS
`ts`
`ba_rs`
AXIOMS
`axm1` : $ts \subseteq \mathbb{N}$
`axm2` : $ts \neq \varnothing$
`axm3` : $ba\_rs \subseteq \mathbb{N}_1$
`axm4` : $ba\_rs \neq \varnothing$

VARIABLES
`bp`
INVARIANTS
`inv1` : $bp \in ts \nrightarrow ba\_rs$
`inv2` : $bp\_set = TRUE \Rightarrow$
$bp \neq \varnothing \wedge 0 \in dom(bp)$

Because we only need the start time of the corresponding basal rate, we shall be very careful when we make the assignment of initial value to the basal rate (the value of basal rate when the infusion process starts), which will be illustrated in the next refinement. To avoid the occurrence of issues such as the start time being before

the minimum start time in the basal profile we made a design assumption that the minimum start time of the basal profile is 0. Therefore the user shall program the basal rate at time 0. The above discussion describes the content of inv2, which says when the basal profile is set, the basal profile is not empty and 0 shall be included in the domain of *bp*.

4.5.3 Refining the Events

This refinement only focuses on the basal profile settings. The basal profile setting is an interaction behaviour between the user and the device. The controller gets the input from the user (environment), validates the input, and makes the corresponding state change in the controller. The basal profile setting process is a database setting process, which includes adding, deleting, modifying, and reading.

Event $BP\_ADD \mathrel{\widehat{=}}$
any
BA_T
BA_R
where
grd1 : $BA\_T \in ts$
grd2 : $BA\_T \notin dom(bp)$
grd3 : $BA\_R \in ba\_rs$
grd4 : $bp\_set = FALSE$
grd5 : $power = on$
then
act1 : $bp := bp \cup \{BA\_T \mapsto BA\_R\}$
end

Event $BP\_DEL \mathrel{\widehat{=}}$
any
BA_T
where
grd1 : $BA\_T \in dom(bp)$
grd2 : $bp\_set = FALSE$
grd3 : $power = on$
then
act1 : $bp := bp \setminus \{BA\_T \mapsto bp(BA\_T)\}$
end

Event $BP\_OVERRIDE \mathrel{\widehat{=}}$
any
BA_T
BA_R
where
grd1 : $BA\_T \in dom(bp)$
grd2 : $bp \neq \varnothing \Rightarrow BA\_R \neq bp(BA\_T)$
grd3 : $BA\_R \in ba\_rs$
grd4 : $bp\_set = FALSE$
grd5 : $power = on$
then
act1 : $bp := (\{BA\_T\} \lhd bp) \cup \{BA\_T \mapsto BA\_R\}$
end

Event $BP\_VIEW \mathrel{\widehat{=}}$
any
BA_T
$result$
where
grd1 : $BA\_T \in dom(bp)$
grd2 : $result = bp(BA\_T)$
grd3 : $power = on$
then
skip
end

In Event-B one usage of *abstract parameters* is to describe the input variables from the environment; another use of abstract parameters is to describe some abstract variables which need to be refined (see *ready* in Section 4.3.3).

The *BA_T, BA_R* stand for the input basal rate start time and basal rate respectively. The *BP_ADD* event adds a new pair which does not exist in the basal profile. The *BP_DEL* event deletes an existing pair from the basal profile. Because for different times, the corresponding basal rate may be different, in *BP_DEL* the start time only is enough to derive the corresponding basal rate from *bp*. The *BP_OVERRIDE* event finds an existing pair in the basal profile and changes the corresponding basal rate to another value. *grd2* says the input *BA_R* substitutes the relevant basal rate of *BA_T* under the condition that *bp* is not empty. The action *act1* in *BP_OVERRIDE* says the *bp* is equal to the domain subtraction ($\lhd$) *BA_T* from *bp* and union the new pair of start time to basal rate. The *BP_VIEW* just simply views the content of the basal profile, so there are no actions in this event. Notice that the basal profile setting has no relation to the priming process, but is directly related to the condition when *power* is *on*. Therefore *power* = *on* exists in all the events related to the basal profile setting in the form of a guard.

Initialisation
extended
begin
$\oplus$`act6` : $bp := \varnothing$
end

Event *BP_COMP* $\widehat{=}$
refines *BP_SET*
when
$\oplus$`grd3` : $bp \neq \varnothing$
$\oplus$`grd4` : $0 \in dom(bp)$

There are two previous events that have some changes during this refinement stage. The *BP_COMP* (basal profile complete) refines the *BP_SET* by adding guards such as *bp* is not an empty set and 0 is included in the domain of basal profile. The

reason for adding these two guards has been discussed in the last section.

4.5.4 Summary of the Second Refinement

This refinement focuses on the basal profile settings by introducing several new events to describe the process of basal profile setting. The context **C2** extends **C1**; the machine **IIP2** refines **IIP1**; **IIP2** sees **C2**. This refinement is mainly about the basal profile function. The new added events illustrate the interaction between users and the device. By running this model on the Rodin Platform, 13 POs (see Appendix B.3.2) are generated and all of them are automatically proved by Rodin. Because we only refined two existing events in *IIP1*, the Appendix B.3.1 only shows the new events and refined events.

4.6 Time Constraint Patterns

As we mentioned at the beginning of this chapter, IIP is a time related control system. To use Event-B for systems where timing is important, it is necessary to find ways of expressing time properties in Event-B specifications. The time constraint pattern for Event-B development and its practical usage can be found in [CMR07] and [Reh06].

4.6.1 An Existing Time Pattern

In this subsection we briefly present the time constraint pattern in [CMR07]. The proposed solution is to create a time activation set *at*, which records all the predictable activation time points of the system. Time elapsing is specified in terms of a special event *TICK_TOCK*. For any time that is smaller than the minimum value in the

time set, the current time will jump to that value in one step without taking on any intervening time value. Each event related to a predictable time point is guarded by some time constraint. Once the specified time constraint is satisfied for some event, the relevant action is triggered and the time point is deleted from the activation time set.

MACHINE Time Pattern
VARIABLES
  `time`
  `at`
INVARIANTS
  `inv1` : $time \in \mathbb{N}$
  `inv2` : $at \subseteq \mathbb{N}$
  `inv3` : $at \neq \varnothing \Rightarrow time \leq min(at)$
EVENTS
Initialisation
  begin
    `act1` : $time := 0$
    `act2` : $at := \varnothing$
  end
Event *POST_TIME* $\widehat{=}$
  any
    tm
  where
    `grd1` : $tm \in \mathbb{N}$
    `grd2` : $tm > time$
  then
    `act1` : $at := at \cup \{tm\}$
  end
Event *PROCESS_TIME* $\widehat{=}$
  when
    `grd1` : $time \in at$
  then
    `act1` : $at := at \setminus \{time\}$
  end
Event *TICK_TOCK* $\widehat{=}$
  any
    tm
  where
    `grd1` : $tm \in \mathbb{N}$
    `grd2` : $tm > time$
    `grd3` : $at \neq \varnothing \Rightarrow tm \leq min(at)$
  then
    `act1` : $time := tm$
  end
END

From the time constraint pattern model we can see this model does not have any context. The variables *time, at* stand for current time and time activation set respectively. The relavant invariants says:

inv1 – the current time should be a natural number.

inv2 – the activation time set should be a subset of the natural numbers.

inv3 – when *at* is not empty we can infer that the current time is smaller than or equal to the minimum value of *at*.

The main events *POST_TIME, PROCESS_TIME, TICK_TOCK* describe the creation of the active time set, the deletion of the current active time from *at*, and time progression. The *POST_TIME* event says for any time *tm*, which is bigger than the current time, it should be added to the *at* set. Because this is a pattern, only one basic guard (grd1) occurs in the *PROCESS_TIME* event. Once the time reaches a time point appearing in *at*, the current time should be deleted from *at*. Another important event *TICK_TOCK* is an updating of the current time variable to keep the current time as the latest time stamp.

4.6.2 Extension of Time Pattern

The time constraint pattern in the last section has already been applied to several practical cases such as the IEEE 1394 protocol [CMR07] and business information systems [BFRR10]. One of the contributions in this thesis is trying to extend this pattern and make it more applicable for time related medical devices such as IIP.

Through the discussion of the time constraint pattern, some disadvantages are discernible. We create a model called *Counter Example* to indicate the problems we meet if we follow the pattern illustrated in Section 4.6.1.

The main disadvantage of the time pattern is that it is unable to handle the situation when an unpredictable event disables the predictable events. A new variable *state* is introduced into the model and the guards of *PROCESS_TIME* are extended by $state = FALSE$. We have two unpredictable new events *NOD_EVT1, NOD_EVT2*, which are triggered through the environment, in the model. This *NOD_EVT1* simply changes the *state* from *FALSE* to *TRUE* and the *NOD_EVT2* has the opposite behaviour. Suppose the *POST_TIME* event assigns the *at* as a set $\{t_1, t_2, t_3...t_n\}$ in an

MACHINE Counter Example
VARIABLES
`time`
`at`
`state`
INVARIANTS
`inv1` : $time \in \mathbb{N}$
`inv2` : $at \subseteq \mathbb{N}$
`inv3` : $at \neq \varnothing \Rightarrow time \leq min(at)$
`inv4` : $state \in BOOL$
EVENTS
Initialisation
begin
`act1` : $time := 0$
`act2` : $at := \varnothing$
`act3` : $state := FALSE$
end
Event *POST_TIME* $\widehat{=}$
any
tm
where
`grd1` : $tm \in \mathbb{N}$
`grd2` : $tm > time$
then
`act1` : $at := at \cup \{time\}$
end
Event *NOD_EVT1* $\widehat{=}$
when
`grd1` : $state = FALSE$
then
`act1` : $state := TRUE$
end
Event *NOD_EVT2* $\widehat{=}$
when
`grd1` : $state = TRUE$
then
`act1` : $state := FALSE$
end
Event *PROCESS_TIME* $\widehat{=}$
when
`grd1` : $time \in at$
`grd2` : $state = FALSE$
then
`act1` : $at := at \setminus \{time\}$
end
Event *TICK_TOCK* $\widehat{=}$
any
tm
where
`grd1` : $tm \in \mathbb{N}$
`grd2` : $tm > time$
`grd3` : $at \neq \varnothing \Rightarrow tm \leq min(at)$
then
`act1` : $time := tm$
end
END

incremental order. If *NOD_EVT1* triggers before some value t_i in at and *NOD_EVT2* triggers after some time $t_j (t_j > t_i)$, then the *PROCESS_TIME* event shall be disabled and the state of at is $\{t_i, t_{i+1} ... t_n\}$. When the *TICK_TOCK* event iterates to t_i, it will stop at the current state and endlessly assign t_i to $time$, which means the $time$ will never reach t_j. So the paradoxical behaviour arises from specifying the model using the time pattern discussed in the last section.

Since we mentioned the infinite iteration of $time$, it is necessary to discuss another disadvantage or even mistake, about the *TICK_TOCK* event. Before the discussion,

one important property of Event-B should be declared. Only one event can be triggered at any point during the execution of the model. Notice that the guard of *TICK_TOCK*, $at \neq \varnothing \Rightarrow tm \leq min(at)$, constrains the abstract parameter tm to iterate to the minimum value of at. But when the tm equals $min(at)$, the *TICK_TOCK* event is still enabled. This situation may result in the *livelock* of the system, which means the system continuously executing *TICK_TOCK* without turning to the other events. The strategy for solving this problem is to disable the *TICK_TOCK* event after each iteration and check wether other events should be triggered. Once some predefined event or unpredictable event is triggered, the *TICK_TOCK* is reset to enabled and executes the next iteration.

To avoid the two disadvantages mentioned above, we made an improvement to the pattern in [CMR07] by adding unpredictable events and by modifying the specification of the tick tock event to a self incrementing clock. We introduce the definitions of predictable and unpredictable events and their definitions here [XM11]. *Predictable events* are those events whose occurrence and timing can be derived from the user defined program. *Unpredictable events* are those events whose occurrence time is unpredictable. In other words, unpredictable events are those events that we do not know whether or when they occur. Examples of such unpredictable events includes user initiated events such as powering off, pausing the IIP, etc. The time increasing in the model is approximately smooth. Smoothly increasing the time is a sufficiently good way to handle and capture the unpredictable events in the model. The variables, *POST_TIME, NOD_EVT1, PROCESS_TIME* are the same as that in *Counter Example*. The core of this pattern is the unpredictable event and the new *TICK_TOCK* event. The modified places are detailed below.

$inv4 : at \neq \varnothing \Rightarrow time \leq min(at) \vee state = TRUE$

We add this new invariant, which says when *at* is not an empty set the *time* should be smaller than the minimum value of *at* or the *state* is *TRUE*.

The modification of the model is focused on the *NOD_EVENT2* and *TICK_TOCK*. First we add *grd*2 into *NOD_EVENT2* and then *act*2. *act*2 says *at* is substituted by its subset and all the elements in the subset are bigger than the current time. For *TOCK_TOCK* we shrink the guards into one guard which is almost the same as *inv*4, except for $time = min(at)$. *act1* in the *TICK_TOCK* event is changed from assigning time an abstract parameter to increasing it by one.

Event *NOD_EVENT2* $\widehat{=}$
  when
    grd1 : $state = TRUE$
    grd2 : $at \neq \varnothing$
  then
    act1 : $state := FALSE$
    act2 : $at : |at' = \{i | i \in at \wedge i > time\}$
  end
Event *TICK_TOCK* $\widehat{=}$
  when
    grd1 : $at \neq \varnothing \Rightarrow time < min(at) \vee state = TRUE$
  then
    act1 : $time := time + 1$
  end

Through the modification of the *Counter Example*, we see *time* in *TICK_TOCK* keeps on incrementing even if the event *PROCESS_TIME* is disabled by the guard $state = FALSE$. Because in *TICK_TOCK* the guard removes the $time = min(at)$ condition, livelock will never happen in this case. When $time = min(at)$, the *TICK_TOCK* is disabled. If $state = FALSE$, *PROCESS_TIME* is triggered; if $state = TRUE$, *PROCESS_TIME* is disabled but the *TICK_TOCK* event is enabled and *time* increments automatically until *NOD_EVENT2* happens. Because the past time will never come back, the later events will only be triggered in the future, all the elements in *at* smaller than or equal to the current *time* shall be deleted from *at*.

4.6.3 A Time Pattern with Unpredictable Events

This new time pattern extended the old one by adding new unpredictable events and by fixing the livelock issue. The new one can effectively handle the unpredictable events, especially under the circumstance when those events appear in pairs. Because a model may contain lots of unpredictable events, we present a general definition of the time pattern.

MACHINE New Time Pattern
VARIABLES
  `time`
  `at`
  `v`
INVARIANTS
  `inv1` : $time \in \mathbb{N}$
  `inv2` : $at \subseteq \mathbb{N}$
  `inv3` : $inv(v_i)$
  `inv4` : $at \neq \varnothing \Rightarrow time \leq min(at) \vee \neg p(v_i)$
EVENTS
Initialisation
  begin
    `act1` : $time := 0$
    `act2` : $at := \varnothing$
    `act3` : $v_i : |v_i' \cdot p(v_i)$
  end
Event *POST_TIME* $\widehat{=}$
  any
    tm
  where
    `grd1` : $tm \in \mathbb{N}$
    `grd2` : $tm > time$
  then
    `act1` : $at := at \cup \{tm\}$
  end
Event *NOD_EVT i* $\widehat{=}$
  when
    `grd1` : $p(v_i)$
  then
    `act1` : $v_i : |v_i' \cdot \neg p(v_i)$
  end
Event *NOD_EVT j* $\widehat{=}$
  when
    `grd1` : $\neg p(v_i)$
    `grd2` : $at \neq \varnothing$
  then
    `act1` : $v_i : |v_i' \cdot p(v_i)$
    `act2` : $at : |at' = \{x | x \in at \wedge x > time\}$
  end
Event *PROCESS_TIME* $\widehat{=}$
  when
    `grd1` : $time \in at$
    `grd2` : $p(v_i)$
  then
    `act1` : $at := at \setminus \{time\}$
  end
Event *TICK_TOCK* $\widehat{=}$
  when
    `grd1` : $at \neq \varnothing \Rightarrow time < min(at) \vee \neg p(v_i)$
  then
    `act1` : $time := time + 1$
  end
END

Symbol Definition

$time$ – current time

at – activation time set

v – a list of variables $v_1, v_2, v_3, v_i...$

$inv(v_i)$ – the corresponding invariants of the element v_i in v

$p(v_i)$ – predicate using variable v_i in v, which is used as a guard of some predictable events (may disable the predictable events)

Pattern Description

This pattern deals with both predictable events and unpredictable events.

POST_TIME are those events which create or change at. A time point which is bigger than the current time shall be added to at. This event creates the activation time set for the predictable events, so the unpredictable time points are not included in at.

PROCESS_TIME describes the predictable events; it simply specifies the state transitions of the model when the current time corresponds to a predictable time point in at.

The *NOD_EVT* indicates those events which are unpredictable. We put $i, j...$ after *NOD_EVT* to show there may exist several unpredictable events in the model. These unpredictable events can appear independently or in pairs. If *NOD_EVT_i* appears independently, in this pattern the *PROCESS_TIME* will never be triggered because of the guard $p(v_i)$. If these events appear in pairs, then *NOD_EVT_j* sets v_i back to the value which satisfies $p(v_i)$. At the same time, the time points in at that are smaller than the current time shall be deleted from at because the relevant time has already elapsed.

The expression $v_i : |v_i' \cdot p(v_i)$ is the notation called before-after predicate assignment in Event-B, meaning v_i is assigned a value that satisfies $p(v_i)$.

The $inv4$ says that if at is not empty, then the current time is smaller than or equal to the minimum value of at or $p(v_i)$ is not true. The reason we add "$\vee \neg p(v_i)$" in $inv4$ is that $TICK\_TOCK$ may be disabled when $time = min(at)$ under the circumstance that the predictable events are disabled by some unpredictable events. Therefore, except when $p(v_i)$ is true, $time$ must be smaller than the minimum of at.

Note that the pattern allows some variations. For example, some of the events can be merged into a single event, such as merging $POST\_TIME$ and NOD_EVT into one event. The application of the new time pattern is exhibited in the refinement of the IIP.

4.6.4 Time Pattern for Classifying the Events

This section discusses another issue often occurring in a time related system – two or even more events triggering simultaneously.

If we use the time pattern discussed in the last subsection to specify this kind of issue, some problems may occur. These problems are a consequence of the state transition mechanism of Event-B – only one enabled event will be randomly picked and executed for each state transition in the model. Suppose we have two events $PROCESS\_TIME\_A$ and $PROCESS\_TIME\_B$. One of the guards in these events is $time = time' \wedge time' \in at$. When this time constraint is satisfied, these two events are enabled and will be enabled for execution. Event-B will randomly pick one of them, say $PROCESS\_TIME\_A$, and execute its actions. But examining the pattern, we see one of the actions is $at := at \setminus \{time\}$, so $time'$ will never be included in at.

The triggering of event *PROCESS_TIME_A* directly results in another event *PROCESS_TIME_B* being disabled. Because the time variable *time* continually increases in *TICK_TOCK*, the *PROCESS_TIME_B*, which we are expecting to be executed, will never happen.

In trying to solve this kind of problem in Event-B, we have come up with two solutions.

Solution I: Combining the Events

If we combine these events, which have the same time constraints, together into one event, the time pattern discussed in section 4.6.2 is an effective way of addressing this issue. Note that the combination of events is dangerous, because it does not mean the old events shall disappear. The old events may still be kept in the model with a slight change to the guard. Therefore, a new issue of deciding wether to keep the old event arises. Moreover, in practice, especially in the development of large scale systems, figuring out all the state transitions with the same time constraints is really a tough job, never mind the problem of combining them together. Even if the developer successfully combines these events into one, checking the correctness of this event is an enormous task. This solution potentially increases the safety risk and the complexity of the system, so we are not recommending it here.

Solution II: Classifying Events

This solution is a modification of the time pattern with unpredictable events. For a large model including a large number of events, we can classify the events into several categories. The criterion for the classification is derived from the variables.

For example, we have *PROCESS_TIME_A* and *PROCESS_TIME_B* in the model. We can find the corresponding events such as *POST_TIME_A* and *POST_TIME_B* in the model. Instead of using *at* as the activation time set, for each category we define an independent time set. The union of these sets defines a global time set which is *at*. Consequently, we create the independent set for the *POST_TIME_A/B* event and delete the time point from each of the corresponding time activation sets in the *PROCESS_TIME_A/B* event, instead of deleting the time point from the global time activation set. Only when all of the events in the model sharing a given time constraint are executed, will the *time* in *TICK_TOCK* increase. Therefore, the most important invariant in the time pattern will also be modified.

$$at\_a \cup at\_b \cup ... \neq \varnothing \Rightarrow time \leq min(at\_a \cup at\_b \cup ...) \vee \neg p(vi)$$

We present the modification of those events which are used to handle this time issue.

Event *POST_TIME_A* $\widehat{=}$
  any
    tm1
  where
    grd1 : $tm1 \in \mathbb{N}$
    grd2 : $tm > time$
  then
    act1 : $at\_a := at\_a \cup \{tm1\}$
  end
Event *POST_TIME_B* $\widehat{=}$
  any
    tm2
  where
    grd1 : $tm2 \in \mathbb{N}$
    grd2 : $tm2 > time$
  then
    act1 : $at\_b := at\_b \cup \{tm2\}$
  end
Event *PROCESS_A* $\widehat{=}$
  when
    grd1 : $time \in at\_a$
  then
    act1 : $at\_a := at\_a \setminus \{time\}$
  end
Event *PROCESS_B* $\widehat{=}$
  when
    grd1 : $time \in at\_b$
  then
    act1 : $at\_b := at\_b \setminus \{time\}$
  end
Event *TICK_TOCK* $\widehat{=}$
  when
    grd1 : $at\_a \cup at\_b \cup ... \neq \varnothing \Rightarrow time < min(at\_a \cup at\_b \cup ...) \vee \neg p(vi)$
  then
    act1 : $time := time + 1$
  end

In *POST_TIME_A*, we change $at := at \cup \{tm\}$ to $at\_a := at\_a \cup \{tm1\}$. Correspondingly, *PROCESS_TIME_A* deletes the current time from at_a (the independent time activation set for A). We make a similar change to *POST_TIME_B* and *PROCESS_TIME_B*. The guard of the *TICK_TOCK* event uses $at\_a \cup at\_b \cup ...$ instead of at. Note that we are not using $p(v_i)$ in *PROCESS_TIME_A* and *PROCESS_TIME_B* but keeping it instead in *TICK_TOCK*, because $p(v_i)$ has no relation with this time issue. Its occurrence or non occurrence has no effect on the pattern. On the other hand, *TICK_TOCK* describes the behaviour of *time*; it will be constrained by all the other $p(v_i)$ which may appear in events that are similar to *PROCESS_TIME*.

4.6.5 Discussions of Zeno Behaviour

Zeno behaviour is often exhibited in hybrid systems which perform continuous and discrete dynamic behaviour [GT08]. Zeno behaviour occurs when an infinite number of discrete transitions happen in a finite amount of time [AAS05].

In principle, our patterns may encounter this Zeno behaviour resulting in the *TICK_TOCK* event being disabled because of infinite occurrences of other events preventing a next tick event. To indicate the nonexistence of Zeno behaviour in our time patterns, we draw the figure 4.2 of an event sequence to explain the Zeno behaviour. Suppose we have events C, D, E, F, A, B and T (TICK_TOCK). The

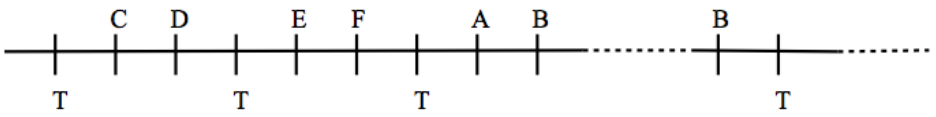

Figure 4.2: Events Trail

sequence of execution of all the events is illustrated by figure 4.2. The Zeno behaviour

says that there are infinite numbers of events between the last two Ts.

The following reasons indicate that the Zeno behaviour does not exist in our time patterns. As we presented in chapter 2, the Event-B models are discretized models. The trigger point of all the events actually follows the sequence specified in the natural numbers. We can express the execution order of the events as the sequence in figure 4.2, but actually the trigger time of all the events except for T are triggered at the same time as the immediately preceding T, because event execution in Event-B is instantaneous, taking no time.. The real trigger times of all the events following the discrete times are illustrated by figure 4.3. Therefore, in our time patterns there does

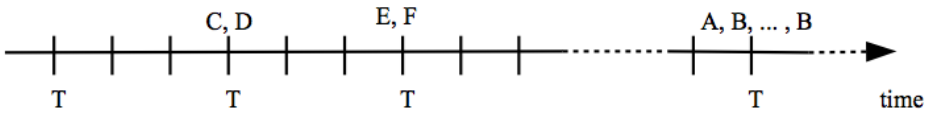

Figure 4.3: Time Line with Events

not exists any events which trigger time between two Ts.

As the time activation set corresponding to an event is finite, each time when an event is triggered the cardinality of the corresponding activation set is decreased by one. Because of the finitness of the sets in our models, Zeno behaviour will not appear in our models.

From a practical aspect, each event in the IIP model takes time, so even if some unpredictable events trigger between two TICK_TOCK events, the TICK_TOCK event still will be reached. Therefore Zeno behaviour does not exist in the IIP models.

4.6.6 Comparison with Other Approaches

Comparision with Event-Action Systems

An event-action system is a software system which trigger actions in the system in response to events triggered by the environment. The triggered actions may trigger some new events, which will trigger other actions. The event-action systems are generally used to model distributed systems. A general-purpose platform called Yeast (Yet another Event-Action Specification Tool), was created by Krishnamurthy and Rosenblum in [KR95] for constructing distributed systems using event-action specifications. The system can handle both temporal and non-temporal events. Time is specified by date and time. Once the date and time have been reached, the date and time will always be past and then any future date and time will be reached. Yeast differentiates events from actions. The events in Yeast describes the events either triggered from the environment or generated from the triggered internal actions. Events can be transient and permanent.

The concept of actions and events in event-action systems is different from Event-B's use of these terms. In Event-B, an event describes several actions. Event-B does not differentiate the events in the environment and the reactions from the system itself. Because timing is not included in the semantics of Event-B, we create patterns to enrich the timing semantics. The idea of specifying time is similar to that in Yeast. Once a time has been reached, the time point will always be in the past.

Comparison with Timed Automata

The classic timed automata [LV95, AD94] use the real numbers to describe the continuous time. There is typically more than one clock in the automaton. A system

transition happens when the passage of time satisfies the time constraint on the transition, which forces the transition between the states, otherwise the system will stay in the previous state. In classic timed automata, the clock can be reset to zero.

The time is described in a discretized way using the natural numbers in our solution. The concept of time in our solution is quite general. The global time set indicates the predictable system transition time points. We only have one clock and the time elapsing is specified as an time tick event. The system transitions happen when the timing constraints (included in the guards) are enabled. We can interpret the constraints on transitions in classic timed automata to the guards in our predictable events. Unpredictable events with unpredictable trigger time are constrained by environmental actions other than the elements in the activation time set. We never reset the time variable to zero.

4.7 Third Refinement: Introducing the Time Pattern

4.7.1 Model Description

This model is the essential refinement for the whole system. The strategy for the refinement is to classify the categories of the infusion process. From the requirements we see the main behaviours of the IIP are described in terms of two kinds of insulin infusion processes, called basal and bolus.

The basal insulin delivery, which supports the basic insulin requirements in the daily life of a user, is of long duration, with small amounts of insulin being infused into the user over the whole day. The basal profile specified in Section 4.4.4 is used

to treat this kind of long duration insulin therapy. The basal insulin delivery can be temporarily overriden by the user. We call this kind of basal temporary basal.

The bolus insulin delivery, which is used to deal with food intake, relative to estimates of exercise levels, has a short infusion period, with relatively large amounts of insulin being processed. The bolus process includes two variants: an instant insulin delivery (normal bolus) and an extended insulin delivery with duration (extended bolus).

From the discussion above, we can classify the insulin infusion process into four categories: basal, temporary basal, bolus, extended bolus. We can apply the time pattern illustrated in Section 4.6.4 on the features of the insulin therapy mentioned above. The requirements expend a lot of words in describing the system behaviour when the insulin infusion pump is paused and resumed during infusion delivery, so the pause and resume of the IIP are essential behaviour. Pause and resume satisfy the definition of unpredictable events we described in Section 4.6.3, so using the time pattern with unpredictable events to handle the pause and resume of IIP is a good choice.

4.7.2 Refining the States

Because this refinement introduces a large amount of constants and variables, we discuss them separately.

Static States

This abstraction level of the model has reached the concrete level of specifying the detailed behaviour of the insulin pump. The *TICK_TOCK* event in the time pattern

we are going to apply on this refinement has a global time activation set, so using the same time unit through the whole model is very important. Intending to specify the system in an accurate way (although using real numbers to describe time is a better choice, but real numbers are not supported by Event-B), we should make our time unit as small as possible. From the above discussion, we made a choice of using seconds as our minimum time unit. Therefore, for all the variables related to time, times are calculated in seconds. For example, the insulin flow rate can be interpreted as the insulin amount delivered per second.

CONSTANTS

`tba_dmax` // maximum duration of temporary basal infusion
`m_f` // maximum flow rate at which the physical pump can function correctly
`exbo_max` // the upper limit of extended bolus the user can be infused with
`bo_max` // the upper limit of bolus the user can be infused with
`exbo_dmax` // the maximum duration of extended bolus infusion

AXIOMS

`axm1` : $ts = 0 \mathinner{..} 86399$
`axm2` : $tba\_dmax \in 1 \mathinner{..} 86399$
`axm3` : $m\_f \in \mathbb{N}_1$
`axm5` : $exbo\_max \in \mathbb{N}_1$
`axm8` : $bo\_max \in \mathbb{N}_1$
`axm9` : $exbo\_dmax \in 1 \mathinner{..} 86399$

The intended meanings of the constants are indicated in the comments beside them. As we discussed above, we choose the second as our time unit. To describe a whole day time point set we use axm1. Although the requirements document does not specify the value of temporary basal duration, we made an assumption here that the duration of a temporary basal infusion is at least 1 second and at most 86399, which means the temporary basal duration will not exceed one day. The same assumption is made about the duration of extended bolus. Although we know the extended bolus

infusion is quite short, for the moment we only can specify the duration of extended bolus does not exceed a day.

Dynamic States

The number of dynamic states of the model is greater than the static ones and their invariants have a direct relationship with the POs of the model. Most of the variables are the quantities mentioned in the requirements. Some of the auxiliary variables, such as basal state (*ba_stat*) and bolus state (*bo_stat*), are conducive to clearer expression of the state transitions in the model. Current time (t) and current day (day) are the variables we add into the model for handling timing issues, which are not mentioned in the requirements. The variables and invariants can be found in:

VARIABLES

```
ba_r // basal rate
ba_stat // basal state
tba_r // temporary basal rate
tba_t // temporary basal stop time
tba_stat // temporary basal state
bo_r // bolus rate
bo_stat // bolus state
bo_req // bolus request amount
bo_am // bolus delivered amount
exbo_a // extended bolus start time
exbo_t // extended bolus stop time
exbo_set // extended bolus setting finish
exbo_req // extended bolus amount requst
exbo_r // extended bolus rate
exbo_stat // extended bolus state
t // current time
day // current day
at_ba // basal activation time set
at_tba // temporary basal activation time set
at_exbo // extended bolus activation time set
```

INVARIANTS

inv1 : $ba\_stat = on \Rightarrow ba\_r \in ba\_rs$
inv2 : $ba\_stat \in state$
inv3 : $tba\_stat = on \Rightarrow tba\_r \in ba\_rs$
inv4 : $tba\_stat \in state$
inv5 : $tba\_stat = on \Rightarrow tba\_t \in \mathbb{N}_1$
inv6 : $bo\_stat = on \Rightarrow bo\_r \in 1 \mathinner{..} m\_f$
inv7 : $bo\_stat \in state$
inv8 : $bo\_stat = on \Rightarrow bo\_req \in 1 \mathinner{..} bo\_max$
inv9 : $bo\_am \in 0 \mathinner{..} bo\_req$
inv10 : $exbo\_a \in \mathbb{N}$
inv11 : $exbo\_t \in \mathbb{N}$
inv12 : $exbo\_set \in BOOL$
inv13 : $exbo\_set = TRUE \Rightarrow exbo\_a < exbo\_t$
inv14 : $exbo\_set = TRUE \Rightarrow exbo\_req \in 1 \mathinner{..} exbo\_max$
inv15 : $exbo\_stat = on \Rightarrow exbo\_r \in \mathbb{N}1$
inv16 : $exbo\_stat \in state$
inv17 : $t \in \mathbb{N}$
inv18 : $day \in \mathbb{N}$
inv19 : $at\_ba \subseteq \mathbb{N}$
inv20 : $at\_tba \subseteq \mathbb{N}_1$
inv21 : $at\_exbo \subseteq \mathbb{N}$
inv22 : $exbo\_set = FALSE \Leftrightarrow at\_exbo = \varnothing$
thm1 : $at\_ba \cup at\_tba \cup at\_exbo \subseteq \mathbb{N}$
inv23 : $(at\_ba \cup at\_tba \cup at\_exbo) \neq \varnothing$
$\Rightarrow$
$t \leq min(at\_ba \cup at\_tba \cup at\_exbo) \vee infu\_stat = paused$
inv24 : $tba\_stat = on \Rightarrow ba\_stat = off \wedge bo\_stat = off$
inv25 : $infu\_stat = paused \Rightarrow ba\_stat = off \wedge tba\_stat = off \wedge exbo\_stat = off \wedge bo\_stat = off$
inv26 : $bp\_set = FALSE \wedge infu\_stat = stopped \Rightarrow at\_ba = \varnothing$
inv27 : $bo\_stat = on \Rightarrow exbo\_stat = off$
inv28 : $ba\_stat = off \Leftrightarrow ba\_r = 0$
inv29 : $tba\_stat = off \Leftrightarrow tba\_r = 0 \wedge at\_tba = \varnothing$
inv30 : $exbo\_stat = off \Leftrightarrow exbo\_r = 0$
inv31 : $bo\_stat = off \Leftrightarrow bo\_r = 0$
inv32 : $rate \leq m\_f$
inv33 : $rate = ba\_r + tba\_r + exbo\_r + bo\_r$

Invariant Description

The invariant inv1 to inv12 and inv14 to inv21 indicate the types of the above variables.

$inv13$ – when the extended bolus setting is complete, the extended bolus start time shall be smaller than the extended bolus stop time.

$inv22$ – the extended bolus setting is not complete is equivalent to the extended bolus activation time set is an empty set.

$thm1$ [4] – the global time activation time set is a subset of the natural numbers.

$inv23$ – the global time activation set is not an empty set implies the current time is smaller than the minimum time point in the global time set or the infusion process is paused.

$inv24$ – the temporary basal is in process implies there is no basal and no normal bolus in process.

$inv25$ – the infusion process is paused implies all the insulin process is stopped.

$inv26$ – basal profile setting has not completed and the infusion state is stopped implies the basal time activation set is an empty set.

$inv27$ – normal bolus is in process implies the extended bolus is not in process.

$inv28$ – basal infusion is not in process is equivalent to current basal flow rate is equal to 0.

$inv29$ – temporary basal rate infusion is not in process is equivalent to the current temporary basal rate is equal to 0 and the relevant time activation set is an empty set.

$inv30$ – extended bolus is not in process is equivalent to the current extended bolus flow rate is equal to 0.

$inv31$ – normal bolus is not in process is equivalent to the current normal bolus flow rate is equal to 0.

$inv32$ – the combined insulin flow rate is smaller than or equal to the maximum flow rate.

$inv33$ – the combined insulin flow rate is equal to the sum of all possible insulin process rates.

Almost all of the above invariants are derived from the requirements documents. Some of them are derived from the requirements in an augmented form. Some requirements are represented in terms of events in the next section. Notice that, because Event-B does not have mechanisms for checking the completeness of system properties, there may exist some missing properties of the system. Furthermore, only

[4] thm stands for those invariants that can be derived from other invariants.

discussions with domain experts can help determine whether there are missing requirements.

4.7.3 Refining the Events

As we mentioned before, our strategy in this refinement is to classify the insulin infusion process into four categories. This subsection will separately describe the events related to these four different insulin infusion processes. Apart from the four categories of insulin infusion, there exist some interesting issues such as infusion start, infusion pause and resume. Because this refinement is a big leap of abstraction to concretion, some inconsistencies and undefinedness occurred during the discharging of POs on this abstract model and we made some corrections to the original requirements document.

Proof Trees in the Rodin Platform

Before refining the events, we should first have presented the reasons behind our intension of discharging the POs generated by our model. Why do we have to prove these POs and how do we prove those POs in the Rodin Platform? The Rodin Platform can generate a bunch of POs, and most of the POs are automatically proved. For the rest of the POs, we need to prove them interactively with the tool. For all the POs, the Rodin Platform will generate a proof tree to describe how the PO is proved or why it is not proved. The following figure 4.4 is a very simple example which is the proof tree of the PO (PAUSE/inv5/INV) in refinement two. Each line in the proof tree indicates a node in terms of a sequent. Arrows with the same indentation mean the nodes come from the same father node. On the right of a node, a comment

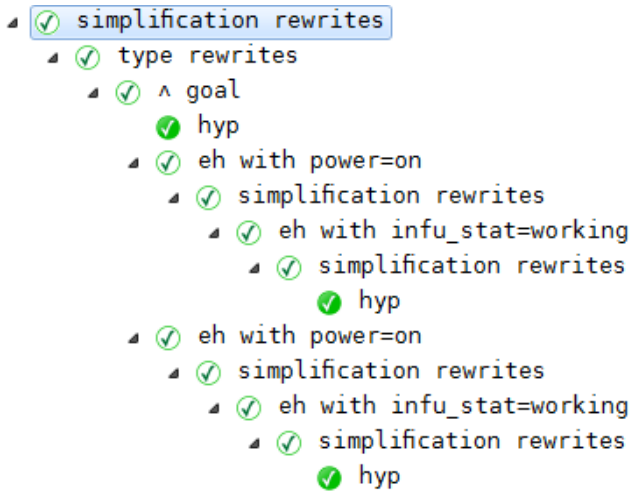

Figure 4.4: Proof Tree in Rodin

explains the discharge justification. Figure 4.5 is the structure of the PO tree in figure 4.4. When all the leaves of the PO trees are proved the whole PO is discharged. So

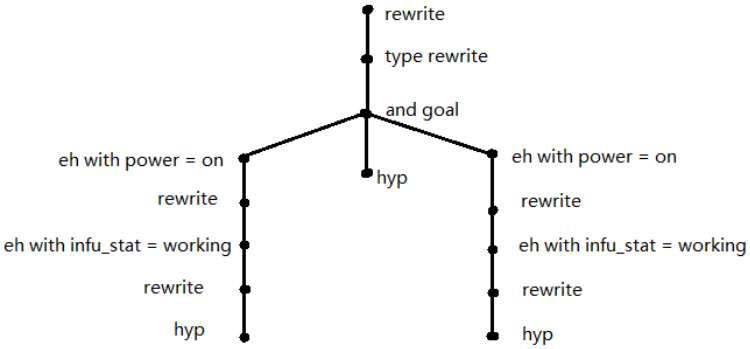

Figure 4.5: Proof Tree Structure

when we are discharging those POs which are not automatically proved by the Rodin Platform, we only need to focus on those branchs which have not been proved. When we meet some leaf sequent that we cannot prove, we can add an assumption to the sequent. If the sequent can be proved by adding the assumption, but the assumption

cannot be proved, this means maybe we forgot to add some guards or some constant property is missing or even an invariant is wrong.

Discussion of Infusion Start

This subsection gives a discussion of the infusion start event. The infusion start is an interesting issue; it is the first enabled event in the second phase in the second refinement. However, when other new events are introduced in this refinement, some issues occur. In later subsections we will add a new event called *EXBO_SET* into the model, because the extended bolus is an infusion process with delay. The user can program the start time and stop time for the extended bolus. The extended bolus setting may occur before the *INFU_START* event. If the extended bolus start time *exbo_a* is later than the start infusion time, then everything goes well. If the extended bolus start time *exbo_a* is equal to the start infusion time, then the combined flow rate becomes basal rate *ba_r* and *exbo_r*. If the extended bolus start time *exbo_a* is before the start infusion time, then a problem occurs. In the requirements document, this situation is not described. The extended bolus cannot infuse (*exbo_stat = off*) if the infusion status is stopped (*infu_stat = stopped*). The requirements did not mention whether the programmed extended bolus should be executed (when the infusion start time is during the programmed duration of extended bolus) or the extended bolus should be cancelled and the user is informed about the delayed infusion start (when the infusion start time is bigger than the extended bolus start time).

From the above discussion, we chose the latter solution. The programmed extended bolus will be cancelled if the infusion start time is later than the programmed extended bolus start time. For the temporary basal and normal bolus, this situation

does not exist. Both of them are instantly delivered, when the controller gets the input from the user. Notice that on this abstract level, we are not considering the action and reaction delay between controller and the real physical pump. All the events are triggered instantly, which is also a feature of Event-B. Therefore, there must be three cases to describe the *INFU_START* event.

Event *INFU_START_NORMAL* $\widehat{=}$
refines *INFU_START*
any
CT
where
$\oplus$grd6 : $CT \in 0 \mathinner{..} 86399$
$\oplus$grd7 : $at\_exbo \neq \varnothing \Rightarrow CT < exbo\_a$
$\oplus$grd8 : $ba\_stat = off \wedge tba\_stat = off \wedge exbo\_stat = off \wedge bo\_stat = off$
$\oplus$grd9 : $at\_ba = \varnothing \wedge at\_tba = \varnothing$
then
$\oplus$act1 : $rate := bp(max(\{i|i \in dom(bp) \wedge i \leq CT\}))$
$\oplus$act3 : $t := CT$
$\oplus$act4 : $at\_ba := \{i|i \in dom(bp) \wedge i > CT\}$
$\oplus$act5 : $ba\_r := bp(max(\{i|i \in dom(bp) \wedge i \leq CT\}))$
$\oplus$act6 : $ba\_stat := on$
end

INFU_START_NORMAL indicates the normal situation, where the start time (*CT*) is smaller than the start time of the extended bolus (*exbo_a*) (*grd6*). We use the *CT* abstract parameter to indicate the current time when the infusion starts. The *grd8* says when the infusion starts, the previous state of the insulin infusion shall be *off*. Corresponding to the time pattern, *INFU_START* (including the two cases not yet discussed), is a typical *POST_TIME* event in the time pattern 4.6.3. *act1* says the flow rate (*rate*) change to the basal rate (*ba_r*), is equal to the corresponding value in *bp* of the biggest time point in the domain of *bp* that is less than or equal to the current time. The basal state is set to *on* and the activation basal time set is overriden by a set with all the elements bigger than current time (*t*).

The second and third case of start infusion is similar to case one; to save space we only present the different parts of case two and case three as compared to case one.

Event *INFU_START_EXBO* $\widehat{=}$
refines *INFU_START*
where
$\oplus$grd7 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge CT = exbo\_a$
$\oplus$grd10 : $exbo\_t - exbo\_a > 0$
$\oplus$grd11 : $exbo\_req/(exbo\_t - exbo\_a) \in \mathbb{N}1$
$\oplus$grd12 : $exbo\_set = TRUE$
then
$\oplus$act1 : $rate := bp(max(\{i|i \in dom(bp) \wedge i \leq CT\})) + exbo\_req/(exbo\_t - exbo\_a)$
$\oplus$act7 : $exbo\_r := exbo\_req/(exbo\_t - exbo\_a)$
$\oplus$act8 : $exbo\_stat := on$
$\oplus$act9 : $at\_exbo := at\_exbo \setminus \{exbo\_a\}$
end

For the case when the start infusion time is equal to *exbo_a*, the combined flow rate becomes the basal rate plus the extended bolus rate (*exbo_r*). *ba_r* holds the same value as that in case one. The extended bolus state (*exbo_stat*) is set to *on*; *exbo_r* is set to a corresponding value; the activation set of the extended bolus set subtracts the extended bolus start time.

Event *INFU_START_LATE_EXBO* $\widehat{=}$
refines *INFU_START*
where
$\oplus$grd7 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge CT > exbo\_a$
then
$\oplus$act7 : $at\_exbo := \varnothing$
$\oplus$act8 : $exbo\_set := FALSE$
end

INFU_START_LATE_EXBO describes the case when the start infusion time is bigger than *exbo_a*. *at_exbo* should be set to the empty set and the auxiliary variable (*exbo_set*) set back to *FALSE*. By running these three events in the Rodin Platform, we see there are several POs which are unproved. We pick up two POs which can not be discharged. Through the automatic proof by the Rodin Platform the proof goal of PO *INFU_START_NORMAL/inv32/INV* stays the same, so there may exist some inconsistency or missing properties of our model. By adding hypotheses and doing

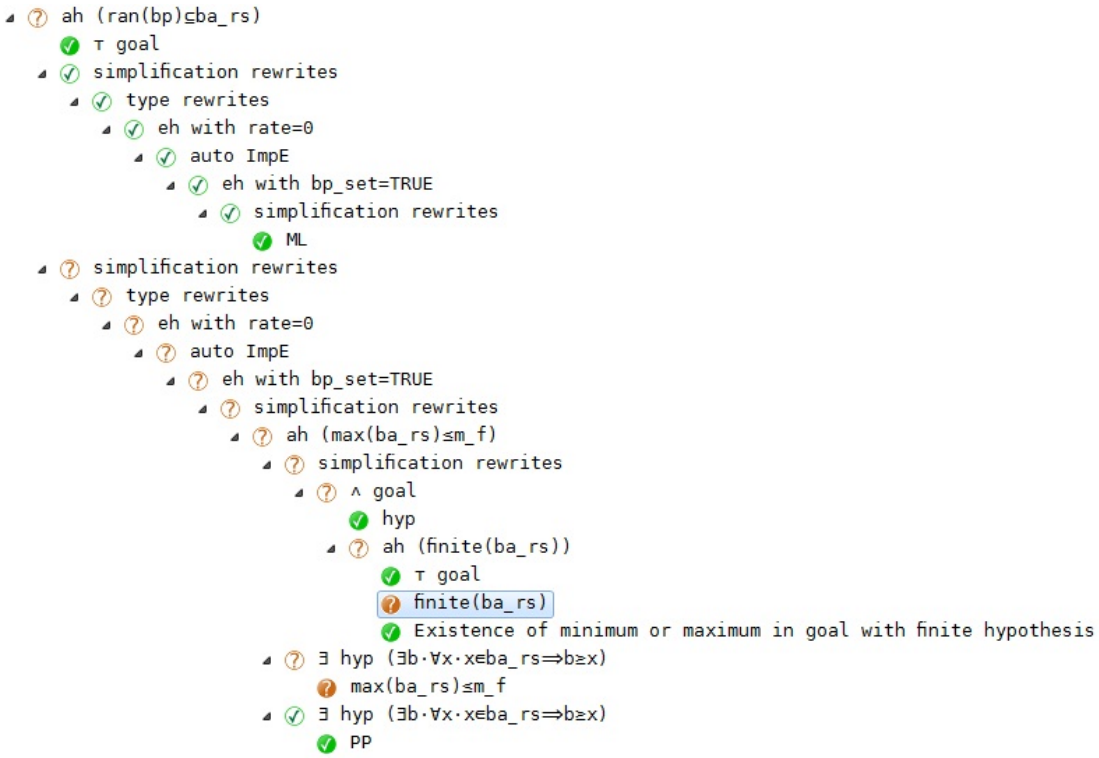

Figure 4.6: Proof Tree of INFU_START_NORMAL/inv32/INV

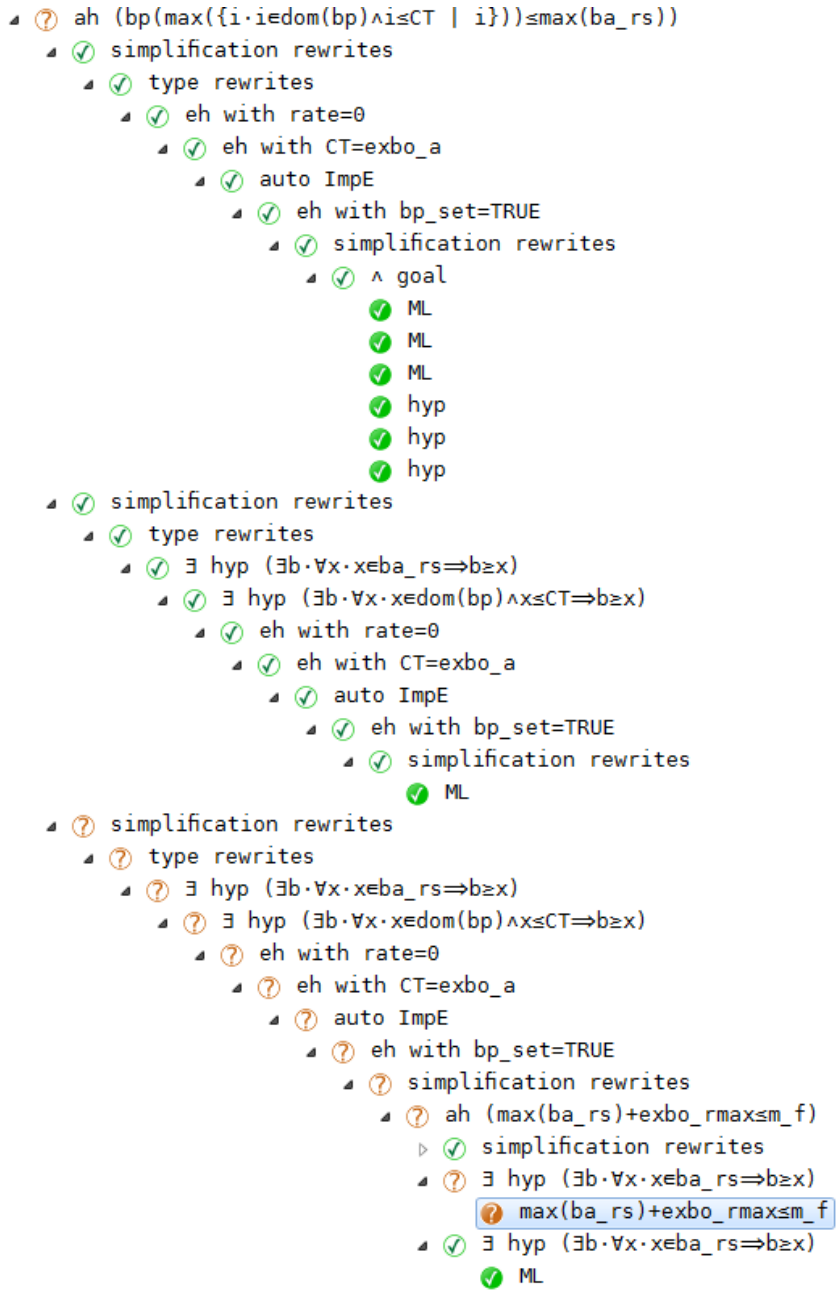

Figure 4.7: Proof Tree of INFU_START_EXBO/inv32/INV

interactive proof with the proof goal, we finally obtained two leaf goals, indicated in figure 4.6. The content in the parentheses next to each node (sequent) in figure 4.6 are the hypotheses made for the sequent. These two goals are both related to the constants in our model, so these two proof goals shall be the missing properties in our model (also in the requirements document). Therefore, we extended the properties (axioms) of constant variables by adding axioms $max(ba\_rs) \leq m\_f$ and $finite(ba\_rs)$ to the context. The new axioms say the maximum basal rate shall be smaller than the maximum flow rate for the pump to function correctly.

The figure 4.7 is another proof tree of a failed proof obligation, namely (*INFU_START_EXBO/inv32/INV*). After interactive proof, we see there exists one unproved goal in the proof tree. This unproved goal is also a property of the constant variables. One issue we should mention here is that the requirements document does not mention the extended bolus flow rate. The *exbo_r* is described in terms of the extended bolus amount over the duration, so there do not exists any constraints on the *exbo_r*. At the first stage of modelling the system, almost all the events related to the extended bolus rate failed to discharge the POs. Therefore, we added the upper limit of the extended bolus (*exbo_rmax*) and from the interactive proof in figure 4.7, a new axiom $max(ba\_rs) + exbo\_rmax \leq m\_f$. The sum of maximum basal insulin flow rate and the maximum extended bolus insulin flow rate shall not exceed the maximum insulin flow rate enabling the pump to function correctly. The grd11 of the event *INFU_START_EXBO* should be revised to $exbo\_req/(exbo\_t - exbo\_a) \in 1 \,..\, exbo\_rmax$.

Discussion of Infusion Pause and Resume

Although the requirements describe in detail the system behaviour of the basal infusion when the system is resumed from the pause state, the requirements do not mention the case where the start time of the extended bolus is set before the pump resume. For the solution of similar issue met in start infusion, we made the same assumption that when the pump is resumed from a paused state, if the start time (*exbo_a*) is earlier than the resume time, then the extended bolus shall be cancelled and the extended bolus set *exbo_set* shall be set to *FALSE*.

The *PAUSE* event is a straightforward event which refines the event with the same name in the abstract model. The actions of *PAUSE* just set the different infusion insulin rates to zero. The reason why *at_tba* is assigned the empty set is that the pause will interrupt any possible infusion process and when the temporary basal is paused, it is impossible to resume it. The basal rate has higher priority than the temporary basal.

Event *PAUSE* $\widehat{=}$
refines *PAUSE*
then
$\oplus$act3 : $ba\_r := 0$
$\oplus$act4 : $ba\_stat := off$
$\oplus$act5 : $tba\_r := 0$
$\oplus$act6 : $tba\_stat := off$
$\oplus$act7 : $at\_tba := \varnothing$
$\oplus$act8 : $exbo\_r := 0$
$\oplus$act9 : $exbo\_stat := off$
$\oplus$act10 : $bo\_r := 0$
$\oplus$act11 : $bo\_stat := off$
end

As we discussed, the resume event may encounter the same situation as the infusion start event. We are going to discuss two cases: one is that the resume time is smaller than or equal to the start time of the extended bolus; another is that the resume time is greater than the start time of the extended bolus.

Event *RESUME_0* $\widehat{=}$

refines *RESUME*
when
$\oplus$grd3 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge t \leq exbo\_a$
then
$\oplus$act1 : $rate := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act3 : $ba\_stat := on$
$\oplus$act4 : $ba\_r := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act5 : $at\_ba := at\_ba \setminus \{i | i \in at\_ba \wedge i \leq t\}$
end
Event *RESUME_1* $\widehat{=}$
refines *RESUME*
when
$\oplus$grd3 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge t > exbo\_a$
then
$\oplus$act1 : $rate := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act3 : $ba\_stat := on$
$\oplus$act4 : $ba\_r := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act5 : $at\_ba := at\_ba \setminus \{i | i \in at\_ba \wedge i \leq t\}$
$\oplus$act6 : $at\_exbo := \varnothing$
$\oplus$act7 : $exbo\_set := FALSE$
end

The behaviours of these two events are almost the same. *act1* changes the value of the combined flow rate to the value of the basal rate and sets the basal state (*ba_state*) to *on*. *ba_r* is transitted to the corresponding value at the current time in the basal profile (*bp*). The time activation set (*at_ba*) is assigned as a new activation time set. The *act6, act7* describe the cancellation of the extended bolus infusion. *RESUME_0, RESUME_1* are typical *NOD_EVTj* event in the time pattern in Section 4.6.3.

Basal Infusion

From the requirements we know the basal infusion process is always running from when the infusion starts, except for the case when the temporary basal is in process or the pump is paused. Here we only need to focus on the basal flow rate fluctuation during the pumping process. For the basal rate changing event we still have two cases to discuss. These cases are not related to basal itself, but related to combined

flow rate. The requirements say that when the bolus is on, the combined flow rate shall increase to the maximum flow rate with which the pump can function correctly. Therefore, we need to consider two situations: basal rate changes when bolus in process; basal rate changes when there is no bolus in process.

Event $BA\_CHANGE\_0 \mathrel{\widehat{=}}$
refines $INFU\_PROC$
when
$\oplus$grd3 : $tmod86400 \in dom(bp) \land (at\_ba \neq \varnothing \Rightarrow t \in at\_ba)$
$\oplus$grd4 : $ba\_stat = on$
$\oplus$grd5 : $ba\_r \in ba\_rs$
$\oplus$grd6 : $rate - ba\_r + bp(tmod86400) \leq m\_f$
$\oplus$grd7 : $bo\_stat = off$
then
$\oplus$act1 : $rate := rate - ba\_r + bp(tmod86400)$
$\oplus$act2 : $at\_ba := at\_ba \setminus \{t\}$
$\oplus$act3 : $ba\_r := bp(tmod86400)$
end
Event $BA\_CHANGE\_1 \mathrel{\widehat{=}}$
refines $INFU\_PROC$
when
$\oplus$grd3 : $tmod86400 \in dom(bp) \land (at\_ba \neq \varnothing \Rightarrow t \in at\_ba)$
$\oplus$grd4 : $ba\_stat = on$
$\oplus$grd5 : $ba\_r \in ba\_rs$
then
$\oplus$act1 : $rate := m\_f$
$\oplus$act2 : $at\_ba := at\_ba \setminus \{t\}$
$\oplus$act3 : $ba\_r := bp(tmod86400)$
end

The above two events illustrate that the state change of ba_r are the same, the only difference is the combined flow rate. Actually for the second event *BA_CHANGE_1*, it is unnecessary to reassign the value of the combined flow rate. But since Event-B has its model refinement consistency rules, we still put $rate := m\_f$ here. Both of the basal rate change events are typical *PROCESS_TIME* events in the time pattern in Section 4.6.3.

Temporary Basal Infusion

The temporary basal infusion process is defined in terms of by two events, which correspond to the variables *tba_r* and *tba_stat* changing events, because the temporary basal infusion shall start immediately when the pump receives the input variables. Here we use *TBA_D, TBA_R* to indicate the duration and flow rate of temporary basal from input. We specified two events *TBA_ON* and *TBA_OFF* to illustrate the state transition related to temporary basal. The *TBA_OFF* is an abstraction of two events: temporary off when time reaches the stop time ($t = tba\_t$); temporary off caused by the user ($t < tba\_t$). The two situations described in *TBA_OFF* have the same system behaviour, so we combine them as one event on this level. They may need to be refined in future refinement stages.

Event $TBA\_ON \mathrel{\widehat{=}}$
refines *INFU_PROC*
  any
    TBA_D
    TBA_R
  where
    $\oplus$grd3 : $TBA\_D \in 1 \mathinner{..} tba\_dmax$
    $\oplus$grd4 : $TBA\_R \in ba\_rs$
    $\oplus$grd5 : $tba\_stat = off$
    $\oplus$grd6 : $bo\_stat = off$
    $\oplus$grd7 : $rate \geq ba\_r$
    $\oplus$grd8 : $at\_tba = \varnothing$
    $\oplus$grd9 : $ba\_stat = on$
  then
    $\oplus$act1 : $rate := rate - ba\_r + TBA\_R$
    $\oplus$act2 : $tba\_r := TBA\_R$
    $\oplus$act3 : $ba\_r := 0$
    $\oplus$act4 : $tba\_t := TBA\_D + t$
    $\oplus$act5 : $tba\_stat := on$
    $\oplus$act6 : $ba\_stat := off$
    $\oplus$act7 : $at\_tba := \{TBA\_D + t\}$
    $\oplus$act8 : $at\_ba := at\_ba \setminus \{i | i \in at\_ba \wedge i < TBA\_D + t\}$
  end
Event $TBA\_OFF \mathrel{\widehat{=}}$
refines *INFU_PROC*

when
$\oplus$grd3 : $t \leq tba\_t$
$\oplus$grd4 : $rate \geq tba\_r$
then
$\oplus$act1 : $rate := rate - tba\_r + bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act2 : $tba\_r := 0$
$\oplus$act3 : $tba\_stat := off$
$\oplus$act4 : $ba\_r := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
$\oplus$act5 : $ba\_stat := on$
$\oplus$act6 : $at\_ba := ran(\lambda i \cdot i \in dom(bp) \wedge i \geq t - day * 86400 | i + day * 86400)$
$\oplus$act7 : $at\_tba := \varnothing$
end

The behaviour of the temporary basal is like a flow rate override. The temporary basal rate overrides a period of basal rate infusion, but with this override there exists an unpredictable situation. Such unpredictable situations may result in some problems. For example, when the temporary basal starts, there exists some start time point of basal between the current time and the programmed time *tba_t*. These start time points of basal will be disabled. If the temporary basal is being infused until the time reaches *tba_t*, to satisfy the timing invariant *inv23* (the current time shall be smaller than or equal to the minimum value in the global time set), we use act8 in *TBA_ON* to delete the time points which are smaller than *tba_t* in *at_ba*. However, the unpredictable stop of temporary basal may result in some basal rate changing event not triggering because of the deletion in act8. Our solution for this problem is to recreate the *at_ba* set. In act6 in *TBA_OFF* we recreate the *at_ba* set when the temporary basal infusion is finished.

Extended Bolus Infusion

The behaviour or the processing of the extended bolus infusion is quite like the system itself. The events related to extended bolus also have two phases: a setting phase and a delivering phase. The setting phase includes the variable setting and programmed

variable modification. Because there are only three input variables, for simplicity we are not considering the input order of these three variables. The modification of the programmed variables event time shall be between the time when the setting has been finished and the programmed time when the extended bolus starts infusion.

Event $EXBO\_SET \mathrel{\widehat{=}}$
any
AM
DUR
ACT
where
$\oplus$grd2 : $AM \in 1 \mathinner{..} exbo\_max$
$\oplus$grd3 : $DUR \in 1 \mathinner{..} exbo\_dmax$
$\oplus$grd4 : $ACT \in t \mathinner{..} t + 86399$
$\oplus$grd5 : $AM/DUR \in 1 \mathinner{..} exbo\_rmax$
$\oplus$grd6 : $exbo\_set = FALSE$
$\oplus$grd7 : $at\_exbo = \varnothing$
then
$\oplus$act1 : $exbo\_req := AM$
$\oplus$act2 : $exbo\_a := ACT$
$\oplus$act3 : $exbo\_t := ACT + DUR$
$\oplus$act4 : $at\_exbo := \{ACT, ACT + DUR\}$
$\oplus$act5 : $exbo\_set := TRUE$
end

Event $EXBO\_SET\_MOD \mathrel{\widehat{=}}$
any
AM
DUR
ACT
where
$\oplus$grd2 : $AM \in 1 \mathinner{..} exbo\_max$
$\oplus$grd3 : $DUR \in 1 \mathinner{..} exbo\_dmax$
$\oplus$grd4 : $ACT \in t \mathinner{..} t + 86399$
$\oplus$grd5 : $AM/DUR \in 1 \mathinner{..} exbo\_rmax$
$\oplus$grd6 : $exbo\_set = TRUE$
$\oplus$grd7 : $t < min(at\_exbo)$
then
$\oplus$act1 : $exbo\_req := AM$
$\oplus$act2 : $exbo\_a := ACT$
$\oplus$act3 : $exbo\_t := ACT + DUR$
$\oplus$act4 : $at\_exbo := \{ACT, ACT + DUR\}$
end

The infusion process of the extended bolus includes extended bolus on and extended bolus off. Because the extended bolus can also be manually stopped, we use one event *EXBO_OFF* to describe the extended bolus stop delivering event.

Event $EXBO\_ON \mathrel{\widehat{=}}$
refines $INFU\_PROC$
when
$\oplus$grd3 : $t = exbo\_a$
$\oplus$grd4 : $exbo\_stat = off$
$\oplus$grd5 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge exbo\_t - exbo\_a > 0$
$\oplus$grd6 : $exbo\_req/(exbo\_t - exbo\_a) \in 1 \mathinner{..} exbo\_rmax$
$\oplus$grd7 : $exbo\_set = TRUE$

$\oplus$grd8 : $bo\_stat = off$
then
$\oplus$act1 : $rate := rate + exbo\_req/(exbo\_t - exbo\_a)$
$\oplus$act2 : $exbo\_r := exbo\_req/(exbo\_t - exbo\_a)$
$\oplus$act3 : $exbo\_stat := on$
$\oplus$act4 : $at\_exbo := at\_exbo \setminus \{exbo\_a\}$
end
Event $EXBO\_OFF \mathrel{\widehat{=}}$
refines $INFU\_PROC$
when
$\oplus$grd3 : $exbo\_a \leq t \wedge t \leq exbo\_t$
$\oplus$grd4 : $exbo\_t - exbo\_a > 0$
$\oplus$grd5 : $exbo\_r \in 1 \mathinner{..} exbo\_rmax$
$\oplus$grd6 : $exbo\_r = exbo\_req/(exbo\_t - exbo\_a)$
$\oplus$grd7 : $exbo\_stat = on$
$\oplus$grd8 : $rate \geq exbo\_r$
then
$\oplus$act1 : $rate := rate - exbo\_r$
$\oplus$act2 : $exbo\_stat := off$
$\oplus$act3 : $exbo\_r := 0$
$\oplus$act4 : $at\_exbo := \varnothing$
$\oplus$act5 : $exbo\_set := FALSE$
end

Because for the extended bolus infusion there does not exist the problem, related to time point overriding, the events only handle the variables related to extended bolus. The extended bolus infusions are typical *PROCESS* events; once time reaches the activation time, the actions trigger and the time point is deleted from the activation time set.

Normal Bolus Infusion

Normal bolus infusion has some different features derived from the above infusion process. The normal bolus is a very short therapy of insulin which is like an injection. On this level we are not expecting to describe the physical model of an injection. Our amount of insulin delivery during time is still a square shape liquid delivery. However, there still exist several interesting issues related to normal bolus.

The stop time decision: Although the requirements say that when the normal bolus is requested, the combined flow rate shall become the maximum flow rate the pump can use and function correctly, how to decide the stop time of the normal bolus is not mentioned. We work out two solutions for this problem. Notice that, for both of these solutions we are not considering the time delay and physical effects such as acceleration of the pump.

Solution one is to add sensors to the physical pump to decide the stop time of bolus infusion. Suppose we have a piston style IIP. We can decide the start time and stop time from sensing the piston position in the reservoir. This solution has one problem. The basal rate changing during this short period is ignored, because by comparing the basal and bolus, we see their magnitudes are hugely different. This short period basal insulin delivery inaccuracy is in the range of tolerance for insulin infusion.

Solution two is to control in real-time the insulin flow rate and accurately calculate the amount of insulin as bolus has been delivered. Once the delivered amount of insulin satisfies the amount of required bolus, control the basal rate to a programmed rate through the basal profile at current time. This solution can accurately deliver the anticipated insulin amount. But the accuracy of the control is based on the accuracy or the calculating speed of the device.

Based on the above discussion, we decided to use the second solution to specify the normal bolus. We create three events (*BO_ON, BO_PROC, BO_OFF*) to describe the behaviour of bolus infusion.

Time handling: The solution we described above needs the delivered amount of insulin to be monitored at each time unit (obviously the smaller the time unit

is, the more accurate is the pump infusion), so we have to add the notion of time increasing to *BO_PROC*. However, our method of specifying the IIP is by following the time patterns described in Section 4.6.4. The *TICK_TOCK* event in the model is the self incrementing clock. If we add a time tick in *BO_PROC*, the model will have two clocks. Trying to solve this problem, we disable the *TICK_TOCK* event by adding guard $bo\_stat = off$. Therefore, when the normal bolus is being delivered, the *BO_PROC* temporarily becomes the clock. Once the normal bolus is stopped or finishes delivering insulin, the *TICK_TOCK* is assigned back to be the global clock.

Event *BO_ON* $\widehat{=}$
refines *INFU_PROC*
  any
    AM
  where
    $\oplus$grd3 : $tba\_stat = off$
    $\oplus$grd4 : $AM \in 1 \mathinner{..} bo\_max$
    $\oplus$grd5 : $bo\_stat = off$
    $\oplus$grd6 : $ba\_r \in ba\_rs$
    $\oplus$grd7 : $exbo\_stat = off$
  then
    $\oplus$act1 : $rate := m\_f$
    $\oplus$act2 : $bo\_r := m\_f - ba\_r$
    $\oplus$act3 : $bo\_stat := on$
    $\oplus$act4 : $bo\_am := 0$
    $\oplus$act5 : $bo\_req := AM$
  end

Event *BO_OFF* $\widehat{=}$
refines *INFU_PROC*
  when
    $\oplus$grd3 : $bo\_am \geq bo\_req$
    $\oplus$grd4 : $bo\_stat = on$
  then
    $\oplus$act1 : $rate := ba\_r$
    $\oplus$act2 : $bo\_stat := off$
    $\oplus$act3 : $bo\_r := 0$
    $\oplus$act4 : $bo\_req := 0$
    $\oplus$act5 : $bo\_am := 0$
  end

Because the normal bolus flow rate depends on the value of other infusion process flow rates, the *BO_PROC* event has lower priority than other events at a specific time point (grd4). act2 is a predicate assignment. Because in Event-B the actions in the same event are executed in parallel, act2 uses predicate assignment to implement the objectives of a sequential assignment.

Event *BO_PROC* $\widehat{=}$
refines *INFU_PROC*
  when
    $\oplus$grd3 : $bo\_stat = on$

⊕grd4 : $t \notin at\_ba \cup at\_tba \cup at\_exbo$
⊕grd5 : $bo\_am < bo\_req$
then
⊕act1 : $rate := m\_f$
⊕act2 : $bo\_r, bo\_am :| bo\_r' = m\_f - ba\_r \wedge bo\_am' = bo\_am + bo\_r'$
⊕act3 : $t := t + 1$
end

From the POs generated from the above events, we got two undischarged POs. One is *BO_PROC/inv6/INV*, whose initial goal is to prove $bo\_stat = on \Rightarrow bo\_r' \in 1 \,..\, m\_f$. By interactive proof with the Event-B tool we get following proof tree:

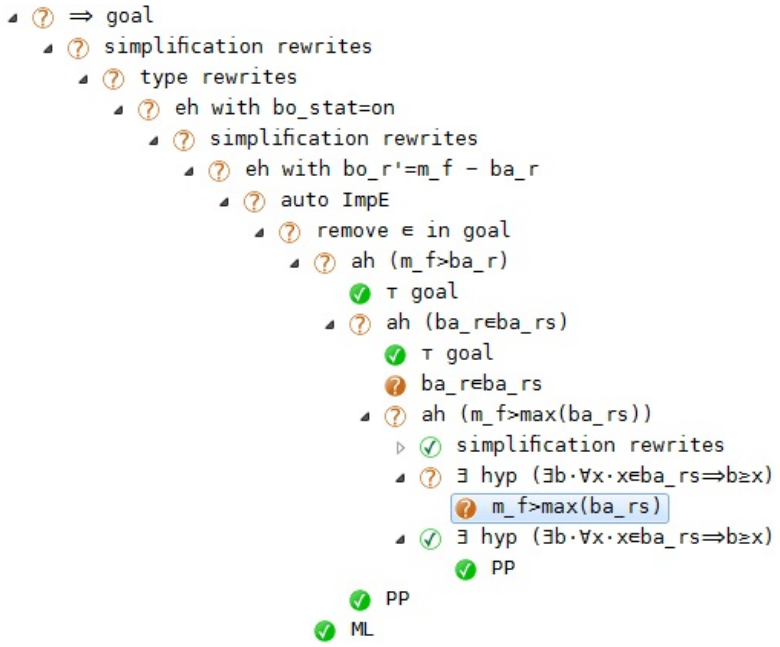

Figure 4.8: Proof Tree of BO_PROC/inv6/INV

The proof tree in figure 4.8 indicates there are two leaf goals that have not been proved. They are the two properties which are missing from the model. For the goal $ba\_r \in ba\_rs$, because ba_r is a dynamic variable, this proof goal is a missing guard from the *BO_PROC* event. For the goal $m\_f > max(ba\_rs)$, we see the axiom $max(ba\_rs) \leq m\_f$ added from the prove obligation in 4.7.3 could be more strict. So

the axiom in **C3** shall be updated.

The second PO is *BO_PROC/inv9/INV*, whose initial proof goal is $bo\_am' \in 0 \,..\, bo\_req$. By automatic proof there are two proof goals generated $0 \leq bo\_am + (m\_f - ba\_r)$ and $bo\_am + (m\_f - ba\_r) \leq bo\_req$.

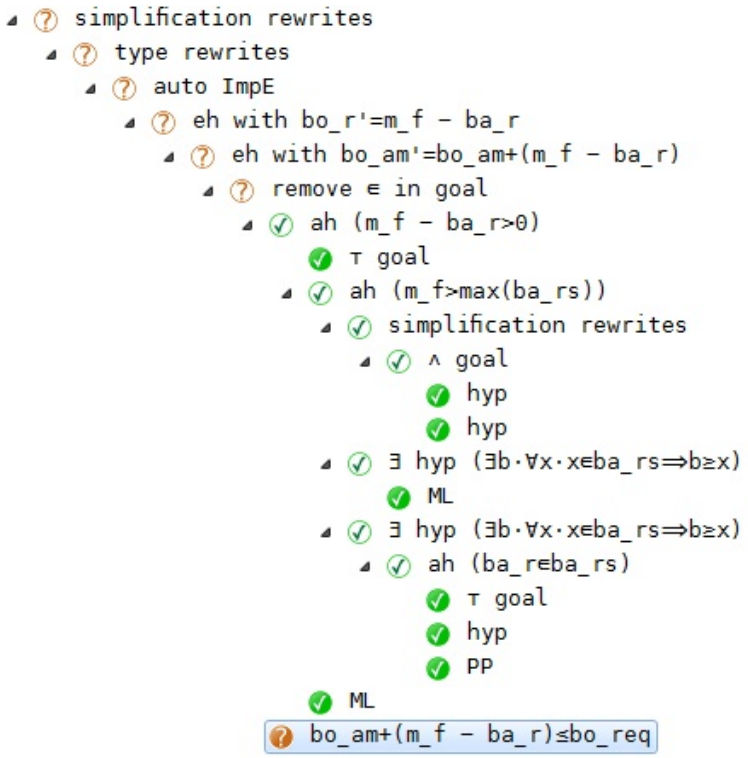

Figure 4.9: Proof Tree of BO_PROC/inv9/INV

By using the restricted axiom derived from 4.8, the first proof goal is proved. The second goal must be the missing guard in *BO_PROC*. By reviewing the *BO_PROC* event, we see grd5 could be replaced by $bo\_am + (m\_f - ba\_r) \leq bo\_req$. The PO can be discharged now. However, by analysing the semantics of the new grd5, we derived a question: what if the requested bolus is smaller than the maximum flow? This is a case which is not mentioned in the requirements. The requirements say once the normal

bolus is in process, the flow rate shall become the maximum flow, which is wrong when the requested bolus amount over the minimum time unit plus the basal rate is smaller than the maximum flow rate ($(bo\_req/timeunit) + ba\_r < m\_f$). Considering this special case we added a special event *BO_SPEC* into the bolus infusion process.

Event $BO\_SPEC \mathrel{\widehat{=}}$
refines *INFU_PROC*
any
AM
where
$\oplus$grd3 : $tba\_stat = off$
$\oplus$grd4 : $AM \in 1 \mathinner{..} bo\_max$
$\oplus$grd5 : $bo\_stat = off$
$\oplus$grd6 : $ba\_r \in ba\_rs$
$\oplus$grd7 : $exbo\_stat = off$
$\oplus$grd8 : $AM + ba\_r < m\_f$
then
$\oplus$act1 : $rate := AM + ba\_r$
$\oplus$act2 : $bo\_r := AM + ba\_r$
$\oplus$act3 : $bo\_stat := on$
$\oplus$act4 : $bo\_am := 0$
$\oplus$act5 : $bo\_req := AM$
end

act1 and act2 are the assumed actions when this special case occurs. Notice that the AM here stands for the amount of insulin delivered during the minimum time unit.

Timing and other Events

The event for describing time increments in this model follows the *TICK_TOCK* event described in section 4.6.4. The only difference is the guards. The reason for adding grd4 has been described in the last subsection. For grd3, we added another event *BA_T_RESET* for counting days and extending *at_ba* to our model.

Event $TICK\_TOCK \mathrel{\widehat{=}}$
when
$\oplus$grd2 : $(at\_ba \cup at\_tba \cup at\_exbo) \neq \varnothing$
$\Rightarrow$
$(t < min(at\_ba \cup at\_tba \cup at\_exbo) \vee infu\_stat = paused)$
$\oplus$grd3 : $t \neq 86400 * day$
$\oplus$grd4 : $bo\_stat = off$
then
$\oplus$act1 : $t := t + 1$
end

The reason why we add this new event into our model is that the domain of basal profiles in our system is one day, but we are using the natural numbers to describe the elapsed time. Therefore the basal profile likes a period of fixed time sliding on the time line towards one direction.

Event $BA\_T\_RESET \widehat{=}$
when
grd1 : $t = 86400 * (day + 1)$
grd2 : $bp\_set = TRUE$
grd3 : $power = on$
then
act1 : $at\_ba := ran(\lambda i \cdot i \in dom(bp) | 86400 * (day + 1) + i)$
act2 : $day := day + 1$
end

Once the current time reaches 86400, the day is incremented by one and the basal activation time set is reset to a new time activation time set with each element 86400 bigger than the corresponding time point in the previous day.

Except for all the events described above, the events used to describe the preparation phase are not refined on this level. The *Initialization, INFU_STOP, POWER_OFF* events assigned all the variables an initial value. To save space we are not describing them here.

4.7.4 Summary of the Third Refinement

This refinement introduced timing issues and the corresponding solutions for them by applying several timing patterns in Event-B. The infusion process was categorized into four kinds. We created four activation time sets for each of the insulin infusion categories. The union of these activation time sets represents the global time set. An event *TICK_TOCK* was used to control time incrementing in the whole system. Because this model is quite near the concrete level, inconsistencies and some missing properties could be determined from this model. By using interactive proof in the Rodin Platform, some missing variables and some properties missing from the

requirements were found. The third refinement is still focused on the controller of the system; the input variables are expressed in terms of abstract parameters. When creating our model, we created a mapping between the requirements documents and our model. By running the third refinement of IIP, 301 POs are generated. 298 POs are generated from the machine, 75 of them are proved interactively, the rest are proved automatically by the Rodin Platform. 3 POs are generated from the Context; all of them are proved automatically. For the full version of refinement three (modified model without fault or inconsistency already found), see Appendix B.4. To save space, we did not record the variables and the invariants in Section 4.7.2.

4.8 Summary of the IIP Model

The IIP is a time related interactive control system. The models presented in this chapter mainly focus on the controller part.

4.8.1 Summary of Refinement Stage

The strategy performed in the refinement process follows the sequence: output variable abstraction → system behaviour abstraction (including the interaction with users) → internal algorithms (including input variable abstraction) → connecting controlled variables → connecting environment variables → implementation refinement.

Notice that, all the above stages may includes several sub refinement stages. Because this project is still under development, the specification presented in this thesis only reaches the internal algorithms stage. Because of the special features of this

system (the user setting information such as pre-setting of basal profile settings), we added a user setting stage before the internal algorithm stage. The summary of the specification is given in the following paragraphs.

Specification Summary

Initial Model – The initialization of the model is an abstraction of the system behaviour by a unpredictable output variable assignment. Some events related to prerequisites of the system operating process (the power state) are also specified in this abstract level.

First Refinement – This refinement step refined the abstraction of system behaviour into two phases. Phase one focuses on the setting of the basal profile and the priming process. The second phase refines the infusion process event to start, pause, resume and stop infusion sub-events.

Second Refinement – The basal profile setting is one of the important processes in phase one. The structure of the basal profile is a simple database. To indicate the data accessing, adding, rewriting etc. behaviour, the second refinement refines the basal profile setting by introducing detailed events.

Third Refinement – The second phase of the system is refined by applying the time patterns to the infusion process. Through the analysis of the requirements, we separate the infusion process into four infusion process categories (basal, temporary basal, normal bolus and extended bolus). Each of the infusion processes has several corresponding variables. We use a special event *TICK_TOCK* to describe time elapsing.

We illustrate the relations between refinement steps in figure 4.10:

The whole IIP project, including learning Event-B, analyzing the requirements document, creating models and discharging POs, took about half a year. Because understanding the Event-B language took a period of time, we cannot accurately estimate the time spent on the project by an expert in Event-B. If a brand new project has similar scale, we estimate that two months is a reasonable time to finish it.

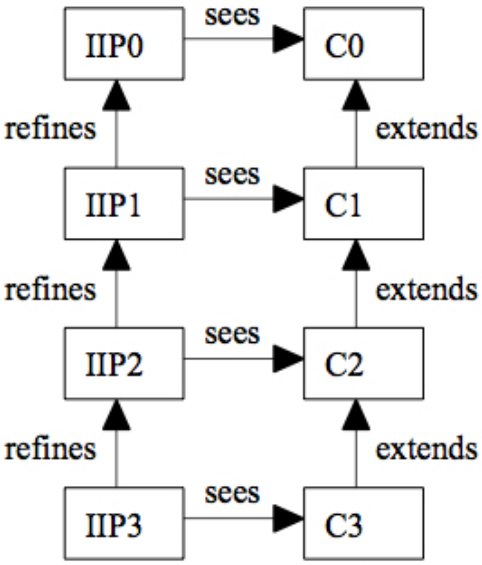

Figure 4.10: Refinement Relations

4.8.2 Summary of Inconsistencies

The main intention in the thesis is to check the correctness and consistency of the requirements documents by creating the model and proving the proof obligations generated from the Event-B model of IIP. We found some inconsistencies and some missing properties of the requirements.

Inconsistencies and Missing Properties

Quantities Added

- The maximum extended bolus flow rate.
- The minimum time unit of the system.
- The maximum duration of the temporary basal infusion.
- The maximum duration of the extended bolus infusion.
- The maximum dosage of the normal bolus.
- The maximum dosage of the extended bolus.

Properties Added

- The sum of maximum basal flow rate and maximum extended bolus rate should be smaller than the maximum flow rate of the pump. (This is the situation when the device is equipped with a powerful pump which can handle the combined flow rate of all infusion processes.)
- The infusions, such as temporary basal, extended bolus and bolus, shall only be executed after the pump has started infusion.

Description Modified

- When the normal bolus infusion starts and the normal bolus amount over the minimum time unit plus the current basal rate is smaller than the maximum flow at which the pump can function correctly, the combined flow shall be equal to the sum of normal bolus amount over the minimum time unit and the current basal rate.

System Behaviour Completeness

There are some situations not considered in the requirements document. For these missing situations, we made some reasonable assumptions to specify the behaviour of the system.

- When the programmed extended bolus start time is later than the infusion start time, the extended bolus needs to be cancelled.
- When the programmed extended bolus start time is between the period of pause and resume, the extended bolus needs to be cancelled.
- When the infusion pump is not powerful enough to handle the combined flow rate for different infusion processes, that is, if the expected combined flow rate exceeds the maximum flow at which the pump can function correctly, there is a question of what the infusion pump should do. (Because we cannot make a reasonable solution to this problem, the model does not indicate any specification for this issue.)
- When the programmed bolus amount is smaller than the amount of insulin the pump can deliver during a minimum time period in the system, the setting of the flow rate is not mentioned.

Chapter 5

Future Work

This chapter presents some future work that should be undertaken on the IIP project and some interesting issues related to Event-B which we are investigating.

5.1 Future Work on the IIP Project

The final goal of the IIP project is to create a faultless practical design process for IIP, so there is a long way to go on the this project to arrive at our destination. For now, we are focusing on the specification stage. To enhance the persuasiveness of the specification, we are using different kinds of specification languages and theorem provers and some hazard analysis techniques on this project, such as tabular expressions, pvs, Event-B, fault trees etc.

The work we are doing now mainly focuses on writing the specification of the IIP. From the description in chapter 4, we see model checking can be used for the validation of the requirements and to enhance the correctness of the system. Because our modelling strategy on IIP using Event-B is following the sequence described in

Chapter 4, there are lots of refinement steps that should be undertaken in the future development of the model. The Event-B specification that has been built for this model only includes some core features of the insulin pump and the the environment variables have not been introduced into the specification, so we are going to add new features and environment variables in the next step of refinement, then connect the environment variables with already existing models. From the perspective of the whole system, the alarms and some exception handling methods should be considered during the future analysis. The model of IIP written in Event-B so far is not that complex. In future refinement steps, by adding new features and environment variables, the complexity of the system will rapidly increase. To handle this complexity, the model decomposition mechanisms should be introduced. A feasible scheme for model decomposition is to decompose the whole system into sub models, namely a controller model, an environment model, an alarm model, a fault events model.

People working on this project are writing in parallel specifications using different specification languages. By comparing the differences between these specification languages we can determine the suitability of the specification languages for specifying different issues. For different components or issues of the system, we could use different specification languages to describe them.

5.2 Future Work on Technical Research

We used Event-B as the specification language on this project. We found some disadvantages that became obstacles during specification of the system. We present some tentative solutions for these disadvantages.

5.2.1 No Support for Real Numbers

Event-B does not support real numbers; all the values related to numbers can only have type $\mathbb{N}$ or $\mathbb{Z}$. This disadvantage results in another problem, which is the integer division problem. All the division in Event-B is an integer division, which decreases the accuracy when we want to express some division calculation.

We can partially solve this problem when the numbers are $\mathbb{Q}$ (rational numbers), which are a subset of $\mathbb{R}$). We can use a surjection function $fraction \in \mathbb{N} \times \mathbb{N}1$ to define the fractions (rational numbers). The arithmetic operators can be represented by a new relation $fraction \times fraction \rightarrowtail\!\!\!\!\!\rightarrow fraction$. For example, the plus operator can be interpreted as $plus \in fraction \times fraction \rightarrowtail\!\!\!\!\!\rightarrow fraction$ together with several relevant events: an event for calculating the sum of the new numerator part, which are the results of multiplying the numerator by the least common multiple (lcm); an event for reduction, which divides both numerator and denominator by the lcm, which is described in the previous event. All the arithmetic operators need to be redefined. However, if we use this solution in a rational number related system, the unreadability and the complexity of the model will definitely increase.

Another solution (actually not a solution, but a special case) of this problem is to keep on using $\mathbb{N}$ or $\mathbb{Z}$ to represent floating point numbers, when the number of decimal places is fixed or the accuracy of the factional part is specified in the requirements. We can amplify the order of magnitude of a floating point number until it becomes an integer. The designers should unify the units when they are amplifying the floating point numbers.

Certainly, the best solution is that the developers of Event-B add support for real numbers in Event-B.

5.2.2 Refinement Consistency Proof Obligation Rules

Event-B is a refinement based specification language. For each refinement stage, there are a set of PO rules used to check the consistency between the refinement and the refined model. A typical refinement rule intending to check the refinement consistency is *SIM* (see proof obligation rules in Chapter 2), which checks that the actions in the refinement simulate the actions in the abstract model. In other words, the events in the refinement can not change the variables in the abstract model.

Although this feature is a meaningful property of Event-B, it increases the difficulty when we specify the system with timing issues using the time patterns (see chapter 4). In refinement three of IIP, we split the infusion process into four different infusion processes. A traditional method of doing the refinement without timing issues being considered is to refine these four infusion processes one by one. The constraint between different infusion processes could be added to the events in terms of guards. The reason we refine these four infusion process in one step is because we have a *TICK_TOCK* event and a global activation time set in our specification. The actions for each infusion process may modify the global time activation time set. If we refine the model step by step, the *SIM* rule will be violated, because the new added event may change the variables of the time activation set. The scale of the IIP is not that big, so refining four infusion processes together in one step is acceptable. If the scale of a system is very large, but the time clock is unique, the complexity will not support human beings in refining the model in only one step.

Therefore, the reason for the problem we meet on the IIP is that it indirectly results from the time pattern we are using.

One solution (may result in faults) to the problem resulting from refinement consistency PO rules is to add new variables to describe the old variables in the abstract model. However, some other issues will come up. The first issue is when we add new variables which have the same meaning in the refinement, the old variables and new variables will both appear in the new model. The readability of the model becomes bad; the readers of the model will get confused when they are reading the model. This is redundant (not the redundancy used in fault tolerant systems) when we design a system. The second issue is that this solution may include some potential safety hazards. The new variables which have the same meaning as those variables in the abstract model may subtly change some variable values (the variables are not expected to be changed) which will not be captured by the POs. That is why we say this solution may result in faults.

Another solution for this problem is: first we use the new variables to indicate the old variables in the old model; secondly, at the final step of refinement we remove all the old variables and copy all the invariants related to the old variables to the final model; in the third step, we substitute all of the old variables by new variables. The model finally obtained by using this solution is theoretically the same as the model obtained from the one step refinement approach. We have not had a chance to create the model by using this solution, so the feasibility and the correctness remain to be proved.

5.2.3 Completeness of the Model

Although Event-B is very powerful for checking the correctness of invariants and the guard completeness of the existing events and the actions, it is weak on checking

the completeness of invariants and events. In other words, if the specifier forgets to write some events or some invariants during the model construction, only using PO discharging, it is hard to find these missing things, such as the special case of bolus delivery in the IIP model (which is not directly detected from the POs).

One solution is to use the animation plug-in tools (AnimB[CM09], ProB) on the Rodin Platform to check the completeness of the model. This solution is similar to software testing, but this way of testing is model based testing. However, during the model animation, how to choose the typical test cases is another issue. Even if all the chosen test cases are running well, we cannot say that the model we created is complete. Another problem is that some restrictions of the animation tool makes the model animation become impossible in some cases. For example, nondeterministic assignment frequently appear when the Event-B model is at a very abstract level. The AnimB tool does not support the nondeterministic assignment to a variable from an infinite set. When this situation occurs, we cannot get the test output that we expect.

Another solution is an idea derived from the IIP project. We are working on this project by using different specification languages. One of the specification languages is tabular expressions (tables) [JP10, JW05]. Tabular expressions have a number of formats, such as function tables, and event tables. One of the significant properties of function tables is completeness and disjointness checking of conditions in the table. If event tables have the same properties (unfortunately, we did not find much related work on event tables), we can write the guards of events in the condition part of tables. Then at least from the perspective of existing guards, we can check the completeness of the events and accordingly create a partially complete model (the completeness of invariant and the completeness of ignored guards cannot be checked). Combining

the completeness checking in tables and model consistency checking in Event-B will increase the reliability of the system.

Chapter 6

Conclusion

This thesis presents a practical case study to demonstrate the system model consistency checking by using Event-B. This thesis is made up of three parts.

The first part is the background knowledge description, which includes the fundamental explanation of Event-B and the IIP project. Event-B plays the role of fault detecter on the specification level during the development of the IIP system. Actually, Event-B can be used throughout the whole development of the software. Comparing Event-B to the V-model [For91], the refinement stages can be interpreted as design stages, such as subsystem design and detailed design in the V-model. The differences between traditional software development and formal methods based development are that the validation or testing in traditional software design is taken at the implementation stage, but formal methods, especially those that have such characteristics as refinement like Event-B, check the validation and verify the consistency of the system during each refinement stage, which evidently reduces the occurrence of errors or bugs in the final implementation.

The second part of the thesis focuses on the usage of Event-B in a practical project

development as an example. The main contributions of the thesis are described in this part. Intending to handle timing issues encountered in the IIP project, we created patterns in Event-B. One pattern is used to handle the triggering times of predictable events and the triggering times of unpredictable events. The main idea of this pattern is to create a global activation time set and a self incrementing clock, in terms of a time event. For each trigger of the time event, the time variable increments by one. A main guard of the time event is that the current time shall be smaller than the minimum value in the global time set. Once the time variable reaches the time specified in the activation time set, the time event is disabled by the main guard and the corresponding predictable event is triggered in the model. Because of the self incrementing mechanism of the time event, the unpredictable events can be easily captured. Because some predictable events may be disabled by the triggering of unpredictable events, to prevent the time event from being disabled when time reaches a time point which is an element in the global activation time set; but the time event is not deleted from the global activation time set; we extended the guard in the time event to make it continuously tick when some unpredictable event is triggered. Another pattern is used to handle the timing issue of events triggering at the same time. This pattern is built on the previous pattern. We handle events triggering at the same time by classifying the events into several categories. For each category we create a corresponding activation time set. The union of these activation time sets substitutes the global activation time set in the previous time pattern; this new global time set prevents some predictable events from being skipped because of the time being inappropriately incremented.

The intention at the specification level is to detect the potential faults and inconsistencies present in the requirements, through creating the model and discharging the POs generated by the Event-B tool. The proof tree structure in the Rodin Platform is a translation from the sequent calculus to the mathematical language of Event-B. From the interactive proving with the Event-B tool, we detected several missing quantities, missing properties, wrong requirements and even missing system behaviour. We presented some discussion and finally made some reasonable changes to deal with the detected problems.

Part three presented future work both on the IIP and on the technical issues we encountered during the usage of Event-B, such as real number support, new variable introduction during refinement, and model completeness checking, Some tentative solutions are proposed to handle the above issues. Each of these problem could be an interesting research topic. We would be interested to see if these problems can be solved by the Event-B community.

Formal methods are the future of software and even system development. There exist a variety of formal specification languages. People who are interested in formal methods are making contributions to this domain. The development of formal methods, not only the theoretical fundamentals, but also tool support, have made are rapid progress over the past few years. In the words of the author of the Event-B book [Abr10] as an ending to this thesis. We believe in the near future we will be able to say "faultless systems – yes we can"!

Appendix A

IIP Requirements

A.1 Infusion Control

1.1 The pump shall suspend all active basal delivery and stop any active bolus during a pump prime or refill.

1.2 The pump shall prohibit any insulin administration during the priming process and resume the suspended basal delivery, either a basal profile or a temporary basal, after the prime is successfully completed.

1.3 The average flow rate in any continuous 60 minute period shall remain accurate within $\pm x\%$ of the programmed rate.

A.2 Basal Programming and Administration

2.1 The pump shall allow the user to program a basal profile with a set of basal rates (at least two), ranging from 0.05 to $x\, Units/hour$ in 0.05 $Units/hour$ increments and the duration for each basal rate.

2.2 Durations of all basal rates shall not overlap with each other, and shall together cover *24* hours of a day.

2.3 Only one basal rate can be activated at any single point of time.

2.4 The pump shall allow the user to temporarily override the current basal delivery with a temporary basal, provided that no normal bolus or other temporary basal is in progress.

2.5 The programmed infusion rate of a temporary basal shall not exceed $x Units/hour$ and the duration of a temporary basal shall not exceed y hours.

2.6 The pump shall allow the user to stop a temporary basal while it is in administration and resume the previously active basal profile after the temporary basal is finished.

2.7 When the user chooses to stop a temporary basal, the pump shall resume the active basal profile prior to the temporary basal.

A.3 Bolus Calculation and Administration

3.1 The pump shall allow the user to define the dosage of a normal bolus in $x Units$ increments.

3.2 The combined flow rate (basal rate + normal bolus rate + extended bolus rate) shall be limited by the maximum flow rate. Normal bolus and extended bolus should not be in progress together.

3.3 The pump shall deliver a valid normal bolus at the highest rate that satisfies *requirement 3.2*.

3.4 The pump shall not allow a normal bolus to start when another normal bolus is in progress.

3.5 The pump shall not allow an extended bolus to start when another extended bolus is in progress.

3.6 The pump shall start a valid extended bolus at the time the user specifies. The extended bolus delivery shall be distributed evenly over its duration.

3.7 The user shall be able to stop an active normal or extended bolus and the bolus shall not be resumed after the suspension.

3.8 When the user stops the bolus the amount of insulin that has been delivered shall be displayed.

3.9 Date/Time change shall result in bolus delivery stopping.

A.4 Drug Reservoir

4.1 The reservoir volume shall be recomputed after priming process.

4.2 The reservoir volume shall be updated after each pump stroke by subtracting the amount of insulin delivered during the stroke.

4.3 The reservoir volume shall be recalculated at the start and end of every basal profile segment, every temporary basal, and every bolus.

A.5 Pump Suspension

5.1 When the user sends a pause command to the pump, the current pump stroke shall be completed prior to suspending the pump.

5.2 When the pump is in suspension mode, insulin deliveries shall be prohibited.

5.3 If the suspension occurs due to a fault condition, the pump shall be stopped

immediately without completing the current pump stroke.

A.6 Alerts and Alarms

6.1 If the insulin remaining in the drug reservoir is less than $x\,Units$ and an infusion is in progress, a *Low Reservoir alert* shall be issued.

6.2 If the insulin remaining in the drug reservoir is 0 $Units$ and an infusion is in progress, an *Empty Reservoir alarm* shall be issued.

6.3 The pump shall monitor the insulin (bolus and basal) delivery in progress. When the actual volume delivered differs from the expected delivery by more than $x\%$, the pump shall signal an alarm and stop the delivery.

6.4 If the currently activated basal profile or the currently ongoing temporary basal has been paused for more than more than x *minutes*, it shall signal an audible alarm for every y *minutes* up to z.

6.5 For each bolus whose dosage exceeds the limit, the pump shall prompt the user to either confirm this bolus or cancel it.

6.6 If the user requests a normal/extended bolus when another normal/extended bolus is in progress, the pump shall issue an alert and deny the request.

6.7 The pump shall continue notifying the user every x *minutes* while an alarm remains unacknowledged and not overridden by alarms with higher priorities.

Appendix B

Model Summary

B.1 Initial Model

B.1.1 Model

CONTEXT C0
SETS
    state
CONSTANTS
    on
    off
AXIOMS
    axm1 : $state = \{on, off\}$
    axm2 : $on \neq off$
END
MACHINE IIP0
SEES C0
VARIABLES
    rate
    power
INVARIANTS
    inv1 : $rate \in \mathbb{N}$
    inv2 : $power \in state$
    inv3 : $power = off \Rightarrow rate = 0$
EVENTS
Initialisation
    begin
        act1 : $rate := 0$
        act2 : $power := off$
    end
Event *POWER_ON* $\widehat{=}$
    when
        grd1 : $power = off$
    then
        act1 : $power := on$
    end
Event *INFU_START* $\widehat{=}$
    any
        ready
    where
        grd1 : $ready \in BOOL$
        grd2 : $ready = TRUE$
        grd3 : $rate = 0$

grd4 : $power = on$
then
act1 : $rate :\in \mathbb{N}_1$
end
Event $INFU\_PROC \mathrel{\widehat{=}}$
when
grd1 : $power = on$
then
act1 : $rate :\in \mathbb{N}$
end

Event $POWER\_OFF \mathrel{\widehat{=}}$
when
grd1 : $power = on$
then
act1 : $power := off$
act2 : $rate := 0$
end
END

B.1.2 Proof Obligations

INITIALISATION/inv1/INV
INITIALISATION/inv3/INV
POWER_ON/inv3/INV
INFU_START/inv1/INV
INFU_START/inv3/INV
INFU_START/act1/FIS
INFU_PROC/inv1/INV
INFU_PROC/inv3/INV
INFU_PROC/act1/FIS
POWER_OFF/inv1/INV
POWER_OFF/inv3/INV

B.2 Refinement One

B.2.1 Model

CONTEXT C1
EXTENDS C0
SETS
status
CONSTANTS
working
paused
stopped
b_2_n
s_2_n

AXIOMS
axm1 : $status = \{working, paused, stopped\}$
axm2 : $working \neq paused$
axm3 : $paused \neq stopped$
axm4 : $working \neq stopped$
axm5 : $b\_2\_n \in BOOL \rightarrow \{0, 1\}$
axm6 : $b\_2\_n(TRUE) = 1$
axm7 : $b\_2\_n(FALSE) = 0$
axm8 : $s\_2\_n \in state \rightarrow \{0, 1\}$
axm9 : $s\_2\_n(on) = 1$

axm10 : $s\_2\_n(off) = 0$
END
MACHINE IIP1
REFINES IIP0
SEES C1
VARIABLES
power
rate
prime
bp_set
infu_stat
INVARIANTS
inv1 : $prime \in BOOL$
inv2 : $bp\_set \in BOOL$
inv3 : $infu\_stat \in status$
inv4 : $infu\_stat = stopped \vee infu\_stat = paused \Rightarrow rate = 0$
inv5 : $infu\_stat = working \vee infu\_stat = paused \Rightarrow power = on \wedge prime = TRUE \wedge bp\_set = TRUE$
EVENTS
Initialisation
extended
begin
act1 : rate := 0
act2 : power := off
act3 : $prime := FALSE$
act4 : $bp\_set := FALSE$
act5 : $infu\_stat := stopped$
end
Event *POWER_ON* $\widehat{=}$
refines *POWER_ON*
when
grd1 : power = off
then
act1 : power := on
end
Event *PRIME* $\widehat{=}$
when
grd1 : $power = on$
grd2 : $prime = FALSE$
then
act1 : $prime := TRUE$
end
Event *BP_SET* $\widehat{=}$
when
grd1 : $power = on$
grd2 : $bp\_set = FALSE$
then
act1 : $bp\_set := TRUE$
end
Event *INFU_START* $\widehat{=}$
refines *INFU_START*
when
grd1 : $power = on$
grd2 : $prime = TRUE$
grd3 : $bp\_set = TRUE$
grd4 : $rate = 0$
grd5 : $infu\_stat = stopped$
with
ready : b_2_n(ready) = s_2_n(power)∗b_2_n(prime)∗b_2_n(bp_set)
then
act1 : $rate :\in \mathbb{N}_1$
act2 : $infu\_stat := working$
end
Event *INFU_PROC* $\widehat{=}$
refines *INFU_PROC*
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
then
act1 : $rate :\in \mathbb{N}_1$
end
Event *PAUSE* $\widehat{=}$
refines *INFU_PROC*
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
then
act1 : $rate := 0$
act2 : $infu\_stat := paused$
end
Event *RESUME* $\widehat{=}$
refines *INFU_PROC*
when
grd1 : $power = on$
grd2 : $infu\_stat = paused$
then
act1 : $rate :\in \mathbb{N}_1$

act2 : *infu_stat* := *working*
end
Event *INFU_STOP* $\widehat{=}$
refines *INFU_PROC*
when
grd1 : *power* = *on*
grd2 : *infu_stat* = *working* $\vee$ *infu_stat* = *paused*
then
act1 : *rate* := *0*
act2 : *infu_stat* := *stopped*
act3 : *prime* := *FALSE*
end
Event *POWER_OFF* $\widehat{=}$
refines *POWER_OFF*
when
grd1 : power = on
then
act1 : power := off
act2 : rate := 0
act3 : *prime* := *FALSE*
act4 : *bp_set* := *FALSE*
act5 : *infu_stat* := *stopped*
end
END

B.2.2 Proof Obligations

axm6/WD
axm7/WD
axm9/WD
axm10/WD
INITIALIZATION/inv4/INV
INITIALIZATION/inv5/INV
POWER_ON/inv5/INV
PRIME/inv5/INV
BP_SET/inv5/INV
INFU_START/ready/WD
INFU_START/ready/WFIS*[1]
INFU_START/inv4/INV
INFU_START/inv5/INV
INFU_START/grd2/GRD*
INFU_PROC/inv4/INV
INFU_PROC/act1/FIS
INFU_PROC/act1/SIM
PAUSE/inv4/INV
PAUSE/inv5/INV
PAUSE/act1/SIM
RESUME/inv4/INV
RESUME/inv5/INV
RESUME/act1/FIS
RESUME/act1/SIM
INFU_STOP/inv4/INV
INFU_STOP/inv5/INV
INFU_STOP/act1/SIM
POWER_OFF/inv4/INV
POWER_OFF/inv4/INV

[1]* indicate those POs which are interactive discharged

B.3 Refinement Two

B.3.1 Model

CONTEXT C2
EXTENDS C1
CONSTANTS
`ts`
`ba_rs`
AXIOMS
`axm1` : $ts \subseteq \mathbb{N}$
`axm2` : $ts \neq \varnothing$
`axm3` : $ba\_rs \subseteq \mathbb{N}_1$
`axm4` : $ba\_rs \neq \varnothing$
END
MACHINE v0.7 IIP2
REFINES v0.7 IIP1
SEES v0.7 C2
VARIABLES
`power`
`rate`
`prime`
`infu_stat`
`bp`
`bp_set`
INVARIANTS
`inv1` : $bp \in ts \nrightarrow ba\_rs$
`inv2` : $bp\_set = TRUE \Rightarrow bp \neq \varnothing \wedge 0 \in dom(bp)$
EVENTS
Initialisation
extended
begin
`act1` : `rate` := 0
`act2` : `power` := `off`
`act3` : `prime` := `FALSE`
`act4` : `bp_set` := `FALSE`
`act5` : `infu_stat` := `stopped`
`act6` : $bp := \varnothing$
end

Event $BP\_ADD \mathrel{\widehat{=}}$
any
BA_T
BA_R
where
`grd1` : $BA\_T \in ts$
`grd2` : $BA\_T \notin dom(bp)$
`grd3` : $BA\_R \in ba\_rs$
`grd4` : $bp\_set = FALSE$
`grd5` : $power = on$
then
`act1` : $bp := bp \cup \{BA\_T \mapsto BA\_R\}$
end
Event $BP\_DEL \mathrel{\widehat{=}}$
any
BA_T
where
`grd1` : $BA\_T \in dom(bp)$
`grd2` : $bp\_set = FALSE$
`grd3` : $power = on$
then
`act1` : $bp := bp \setminus \{BA\_T \mapsto bp(BA\_T)\}$
end
Event $BP\_OVERRIDE \mathrel{\widehat{=}}$
any
BA_T
BA_R
where
`grd1` : $BA\_T \in dom(bp)$
`grd2` : $bp \neq \varnothing \Rightarrow BA\_R \neq bp(BA\_T)$
`grd3` : $BA\_R \in ba\_rs$
`grd4` : $bp\_set = FALSE$
`grd5` : $power = on$
then
`act1` : $bp := (\{BA\_T\} \lhd bp) \cup \{BA\_T \mapsto BA\_R\}$

end
Event *BP_VIEW* $\widehat{=}$
any
BA_T
result
where
grd1 : $BA\_T \in dom(bp)$
grd2 : $result = bp(BA\_T)$
grd3 : $power = on$
then
skip
end

Event *BP_COMP* $\widehat{=}$
refines *BP_SET*
when
grd1 : power = on
grd2 : bp_set = FALSE
grd3 : $bp \neq \varnothing$
grd4 : $0 \in dom(bp)$
then
act1 : bp_set := TRUE
end

B.3.2 Proof Obligations

INITIALISATION/inv1/INV
INITIALISATION/inv2/INV
BP_ADD/inv1/INV
BP_ADD/inv2/INV
BP_DEL/inv1/INV
BP_DEL/inv2/INV
BP_DEL/act1/WD
BP_OVERRIDE/grd2/WD
BP_OVERRIDE/inv1/INV
BP_OVERRIDE/inv2/INV
BP_VIEW/grd2/WD
BP_COMP/inv2/INV
POWER_OFF/inv2/INV

B.4 Refinement Three

B.4.1 Model

CONTEXT v0.7 C3
EXTENDS v0.7 C2
CONSTANTS
tba_dmax
m_f
exbo_rmax
exbo_max
bo_max
exbo_dmax
AXIOMS
axm1 : $ts = 0 \mathinner{..} 86399$
axm2 : $tba\_dmax \in 1 \mathinner{..} 86399$
axm3 : $m\_f \in \mathbb{N}_1$
axm4 : $exbo\_rmax \in \mathbb{N}_1$

axm5 : $exbo\_max \in \mathbb{N}_1$
axm6 : $finite(ba\_rs)$
axm7 : $max(ba\_rs) + exbo\_rmax < m\_f$
axm8 : $bo\_max \in \mathbb{N}_1$
thm : $max(ba\_rs) < m\_f$
axm9 : $exbo\_dmax \in 1 \mathinner{..} 86399$

END
MACHINE IIP3
REFINES IIP2
SEES C3
EVENTS
Initialisation
extended
begin
act1 : rate := 0
act2 : power := off
act3 : prime := FALSE
act4 : bp_set := FALSE
act5 : infu_stat := stopped
act6 : bp := ∅
act7 : $ba\_r := 0$
act8 : $ba\_stat := off$
act9 : $tba\_r := 0$
act10 : $tba\_t := 0$
act11 : $tba\_stat := off$
act12 : $bo\_r := 0$
act13 : $bo\_stat := off$
act14 : $bo\_req := 0$
act15 : $bo\_am := 0$
act16 : $exbo\_a := 0$
act17 : $exbo\_t := 0$
act18 : $exbo\_set := FALSE$
act20 : $exbo\_req := 0$
act21 : $exbo\_r := 0$
act22 : $exbo\_stat := off$
act23 : $t := 0$
act24 : $day := 0$
act25 : $at\_ba := \varnothing$
act26 : $at\_tba := \varnothing$
act27 : $at\_exbo := \varnothing$
end
Event *POWER_ON* $\widehat{=}$
refines *POWER_ON*
when
grd1 : power = off
then
act1 : power := on
end
Event *PRIME* $\widehat{=}$
refines *PRIME*
when
grd1 : power = on
grd2 : prime = FALSE
then
act1 : prime := TRUE
end
Event *BP_ADD* $\widehat{=}$
refines *BP_ADD*
any
BA_T
BA_R
where
grd1 : $BA\_T \in ts$
grd2 : $BA\_T \notin dom(bp)$
grd3 : $BA\_R \in ba\_rs$
grd4 : bp_set = FALSE
grd5 : power = on
then
act1 : $bp := bp \cup \{BA\_T \mapsto BA\_R\}$
end
Event *BP_DEL* $\widehat{=}$
refines *BP_DEL*
any
BA_T
where
grd1 : $BA\_T \in dom(bp)$
grd2 : bp_set = FALSE
grd3 : power = on
then
act1 : $bp := bp \setminus \{BA\_T \mapsto bp(BA\_T)\}$
end
Event *BP_OVERIDE* $\widehat{=}$
refines *BP_OVERIDE*
any
BA_T
BA_R
where
grd1 : $BA\_T \in dom(bp)$

grd2 : bp $\neq \varnothing \Rightarrow$ BA_R $\neq$ bp(BA_T)
grd3 : BA_R $\in$ ba_rs
grd4 : bp_set = FALSE
grd5 : power = on
then
act1 : bp := $(\{BA\_T\} \lhd bp) \cup \{BA\_T \mapsto BA\_R\}$
end
Event *BP_VIEW* $\widehat{=}$
refines *BP_VIEW*
any
BA_T
result
where
grd1 : BA_T $\in$ dom(bp)
grd2 : result = bp(BA_T)
grd3 : power = on
then
skip
end
Event *BP_COMP* $\widehat{=}$
refines *BP_COMP*
when
grd1 : power = on
grd2 : bp_set = FALSE
grd3 : bp $\neq \varnothing$
grd4 : 0 $\in$ dom(bp)
then
act1 : bp_set := TRUE
end
Event *INFU_START_NORMAL* $\widehat{=}$
refines *INFU_START*
any
CT
where
grd1 : *power = on*
grd2 : *prime = TRUE*
grd3 : *bp_set = TRUE*
grd4 : *rate = 0*
grd5 : *infu_stat = stopped*
grd6 : *CT* $\in$ *0 .. 86399*
grd7 : *at_exbo* $\neq \varnothing \Rightarrow$ *CT < exbo_a*
grd8 : *ba_stat = off* $\wedge$ *tba_stat = off* $\wedge$ *exbo_stat = off* $\wedge$ *bo_stat = off*
grd9 : *at_ba* $= \varnothing \wedge$ *at_tba* $= \varnothing$
then
act1 : *rate* := *bp*(*max*($\{i | i \in dom(bp) \wedge i \leq CT\}$))
act2 : *infu_stat* := *working*
act3 : *t* := *CT*
act4 : *at_ba* := $\{i | i \in dom(bp) \wedge i > CT\}$
act5 : *ba_r* := *bp*(*max*($\{i | i \in dom(bp) \wedge i \leq CT\}$))
act6 : *ba_stat* := *on*
end
Event *INFU_START_EXBO* $\widehat{=}$
refines *INFU_START*
any
CT
where
grd1 : *power = on*
grd2 : *prime = TRUE*
grd3 : *bp_set = TRUE*
grd4 : *rate = 0*
grd5 : *infu_stat = stopped*
grd6 : *CT* $\in$ *0 .. 86399*
grd7 : *at_exbo* $= \{exbo\_a, exbo\_t\} \wedge CT = exbo\_a$
grd8 : *ba_stat = off* $\wedge$ *tba_stat = off* $\wedge$ *exbo_stat = off* $\wedge$ *bo_stat = off*
grd9 : *at_ba* $= \varnothing \wedge$ *at_tba* $= \varnothing$
grd10 : *exbo_t* $-$ *exbo_a* $>$ *0*
grd11 : *exbo_req*/(*exbo_t* $-$ *exbo_a*) $\in$ *1 .. exbo_rmax*
grd12 : *exbo_set = TRUE*
then
act1 : *rate* := *bp*(*max*($\{i | i \in dom(bp) \wedge i \leq CT\}$)) + *exbo_req*/(*exbo_t* $-$ *exbo_a*)
act2 : *infu_stat* := *working*
act3 : *t* := *CT*
act4 : *at_ba* := $\{i | i \in dom(bp) \wedge i > CT\}$
act5 : *ba_r* := *bp*(*max*($\{i | i \in dom(bp) \wedge i \leq CT\}$))
act6 : *ba_stat* := *on*
act7 : *exbo_r* := *exbo_req*/(*exbo_t* $-$ *exbo_a*)

act8 : $exbo\_stat := on$
act9 : $at\_exbo := at\_exbo \setminus \{exbo\_a\}$
end
Event $INFU\_START\_LATE\_EXBO \widehat{=}$
refines $INFU\_START$
any
CT
where
grd1 : $power = on$
grd2 : $prime = TRUE$
grd3 : $bp\_set = TRUE$
grd4 : $rate = 0$
grd5 : $infu\_stat = stopped$
grd6 : $CT \in 0 \mathinner{..} 86399$
grd7 : $at\_exbo = \{exbo\_a, exbo\_t\} \land CT > exbo\_a$
grd8 : $ba\_stat = off \land tba\_stat = off \land exbo\_stat = off \land bo\_stat = off$
grd9 : $at\_ba = \varnothing \land at\_tba = \varnothing$
then
act1 : $rate := bp(max(\{i|i \in dom(bp) \land i \leq CT\}))$
act2 : $infu\_stat := working$
act3 : $t := CT$
act4 : $at\_ba := \{i|i \in dom(bp) \land i > CT\}$
act5 : $ba\_r := bp(max(\{i|i \in dom(bp) \land i \leq CT\}))$
act6 : $ba\_stat := on$
act7 : $at\_exbo := \varnothing$
act8 : $exbo\_set := FALSE$
end
Event $PAUSE \widehat{=}$
refines $PAUSE$
when
grd1 : power = on
grd2 : infu_stat = working
then
act1 : rate := 0
act2 : infu_stat := paused
act3 : $ba\_r := 0$
act4 : $ba\_stat := off$
act5 : $tba\_r := 0$
act6 : $tba\_stat := off$
act7 : $at\_tba := \varnothing$
act8 : $exbo\_r := 0$
act9 : $exbo\_stat := off$
act10 : $bo\_r := 0$
act11 : $bo\_stat := off$
end
Event $RESUME\_0 \widehat{=}$
refines $RESUME$
when
grd1 : $power = on$
grd2 : $infu\_stat = paused$
grd4 : $at\_exbo = \{exbo\_a, exbo\_t\} \land t \leq exbo\_a$
then
act1 : $rate := bp(max(\{i|i \in dom(bp) \land i \leq tmod86400\}))$
act2 : $infu\_stat := working$
act3 : $ba\_stat := on$
act4 : $ba\_r := bp(max(\{i|i \in dom(bp) \land i \leq tmod86400\}))$
act5 : $at\_ba := at\_ba \setminus \{i|i \in at\_ba \land i \leq t\}$
end
Event $RESUME\_1 \widehat{=}$
refines $RESUME$
when
grd1 : $power = on$
grd2 : $infu\_stat = paused$
grd4 : $at\_exbo = \{exbo\_a, exbo\_t\} \land t > exbo\_a$
then
act1 : $rate := bp(max(\{i|i \in dom(bp) \land i \leq tmod86400\}))$
act2 : $infu\_stat := working$
act3 : $ba\_stat := on$
act4 : $ba\_r := bp(max(\{i|i \in dom(bp) \land i \leq tmod86400\}))$
act5 : $at\_ba := at\_ba \setminus \{i|i \in at\_ba \land i \leq t\}$
act6 : $at\_exbo := \varnothing$
act7 : $exbo\_set := FALSE$
end

Event $BA\_CHANGE\_0 \widehat{=}$
refines $INFU\_PROC$
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $t - day * 86400 \in dom(bp) \wedge (at\_ba \neq \varnothing \Rightarrow t \in at\_ba)$
grd4 : $ba\_stat = on$
grd5 : $ba\_r \in ba\_rs$
grd6 : $rate - ba\_r + bp(t - day * 86400) \leq m\_f$
grd7 : $bo\_stat = off$
then
act1 : $rate := rate - ba\_r + bp(t - day * 86400)$
act2 : $at\_ba := at\_ba \setminus \{t\}$
act3 : $ba\_r := bp(t - day * 86400)$
end
Event $BA\_CHANGE\_1 \widehat{=}$
refines $INFU\_PROC$
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $t - day * 86400 \in dom(bp) \wedge (at\_ba \neq \varnothing \Rightarrow t \in at\_ba)$
grd4 : $ba\_stat = on$
grd5 : $ba\_r \in ba\_rs$
grd6 : $bo\_stat = on$
then
act1 : $rate := m\_f$
act2 : $at\_ba := at\_ba \setminus \{t\}$
act3 : $ba\_r := bp(t - day * 86400)$
end
Event $TBA\_ON \widehat{=}$
refines $INFU\_PROC$
any
TBA_D
TBA_R
where
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $TBA\_D \in 1 \,..\, tba\_dmax$
grd4 : $TBA\_R \in ba\_rs$
grd5 : $tba\_stat = off$
grd6 : $bo\_stat = off$
grd7 : $rate \geq ba\_r$
grd8 : $at\_tba = \varnothing$
grd9 : $ba\_stat = on$
then
act1 : $rate := rate - ba\_r + TBA\_R$
act2 : $tba\_r := TBA\_R$
act3 : $ba\_r := 0$
act4 : $tba\_t := TBA\_D + t$
act5 : $tba\_stat := on$
act6 : $ba\_stat := off$
act7 : $at\_tba := \{TBA\_D+t\}$
act8 : $at\_ba := at\_ba \setminus \{i | i \in at\_ba \wedge i < TBA\_D + t\}$
end
Event $TBA\_OFF \widehat{=}$
refines $INFU\_PROC$
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $t \leq tba\_t$
grd4 : $rate \geq tba\_r$
then
act1 : $rate := rate - tba\_r + bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
act2 : $tba\_r := 0$
act3 : $tba\_stat := off$
act4 : $ba\_r := bp(max(\{i | i \in dom(bp) \wedge i \leq tmod86400\}))$
act5 : $ba\_stat := on$
act6 : $at\_ba := ran(\lambda i \cdot i \in dom(bp) \wedge i \geq t - day * 86400 | i + day * 86400)$
act7 : $at\_tba := \varnothing$
end
Event $EXBO\_SET \widehat{=}$
any
AM
DUR
ACT
where
grd1 : $infu\_stat = working$
grd2 : $AM \in 1 \,..\, exbo\_max$

grd3 : $DUR \in 1 \mathinner{..} exbo\_dmax$
grd4 : $ACT \in t \mathinner{..} t + 86399$
grd5 : $AM/DUR \in 1 \mathinner{..} exbo\_rmax$
grd6 : $exbo\_set = FALSE$
grd7 : $at\_exbo = \varnothing$

then

act1 : $exbo\_req := AM$
act2 : $exbo\_a := ACT$
act3 : $exbo\_t := ACT + DUR$
act4 : $at\_exbo := \{ACT, ACT + DUR\}$
act5 : $exbo\_set := TRUE$

end

Event *EXBO_SET_MOD* $\widehat{=}$

any

AM
DUR
ACT

where

grd1 : $infu\_stat = working$
grd2 : $AM \in 1 \mathinner{..} exbo\_max$
grd3 : $DUR \in 1 \mathinner{..} exbo\_dmax$
grd4 : $ACT \in t \mathinner{..} t + 86399$
grd5 : $AM/DUR \in 1 \mathinner{..} exbo\_rmax$
grd6 : $exbo\_set = TRUE$
grd7 : $at\_exbo = \varnothing$
grd8 : $t < min(at\_exbo)$

then

act1 : $exbo\_req := AM$
act2 : $exbo\_a := ACT$
act3 : $exbo\_t := ACT + DUR$
act4 : $at\_exbo := \{ACT, ACT + DUR\}$

end

Event *EXBO_ON* $\widehat{=}$

refines *INFU_PROC*

when

grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $t = exbo\_a$
grd4 : $exbo\_stat = off$
grd5 : $at\_exbo = \{exbo\_a, exbo\_t\} \wedge exbo\_t - exbo\_a > 0$
grd6 : $exbo\_req/(exbo\_t - exbo\_a) \in 1 \mathinner{..} exbo\_rmax$
grd7 : $exbo\_set = TRUE$
grd8 : $bo\_stat = off$

then

act1 : $rate := rate + exbo\_req/(exbo\_t - exbo\_a)$
act2 : $exbo\_r := exbo\_req/(exbo\_t - exbo\_a)$
act3 : $exbo\_stat := on$
act4 : $at\_exbo := at\_exbo \setminus \{exbo\_a\}$

end

Event *EXBO_OFF* $\widehat{=}$

refines *INFU_PROC*

when

grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $exbo\_a \leq t \wedge t \leq exbo\_t$
grd4 : $exbo\_t - exbo\_a > 0$
grd5 : $exbo\_r \in 1 \mathinner{..} exbo\_rmax$
grd6 : $exbo\_r = exbo\_req/(exbo\_t - exbo\_a)$
grd7 : $exbo\_stat = on$
grd8 : $rate \geq exbo\_r$

then

act1 : $rate := rate - exbo\_r$
act2 : $exbo\_stat := off$
act3 : $exbo\_r := 0$
act4 : $at\_exbo := \varnothing$
act5 : $exbo\_set := FALSE$

end

Event *BO_ON* $\widehat{=}$

refines *INFU_PROC*

any

AM

where

grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $tba\_stat = off$
grd4 : $AM \in 1 \mathinner{..} bo\_max$
grd5 : $bo\_stat = off$
grd6 : $ba\_r \in ba\_rs$
grd7 : $exbo\_stat = off$
grd8 : $AM + ba\_r \geq m\_f$

then

act1 : $rate := m\_f$
act2 : $bo\_r := m\_f - ba\_r$
act3 : $bo\_stat := on$
act4 : $bo\_am := 0$

act5 : $bo\_req := AM$
end
Event $BO\_SPEC \widehat{=}$
refines $INFU\_PROC$
any
AM
where
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $tba\_stat = off$
grd4 : $AM \in 1 \mathinner{..} bo\_max$
grd5 : $bo\_stat = off$
grd6 : $ba\_r \in ba\_rs$
grd7 : $exbo\_stat = off$
grd8 : $AM + ba\_r < m\_f$
then
act1 : $rate := AM + ba\_r$
act2 : $bo\_r := AM + ba\_r$
act3 : $bo\_stat := on$
act4 : $bo\_am := 0$
act5 : $bo\_req := AM$
end
Event $BO\_PROC \widehat{=}$
refines $INFU\_PROC$
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $bo\_stat = on$
grd4 : $t \notin at\_ba \cup at\_tba \cup at\_exbo$
grd5 : $bo\_am + (m\_f - ba\_r) < bo\_req$
grd6 : $t \neq 86400 * day$
grd7 : $ba\_r \in ba\_rs$
then
act1 : $rate := m\_f$
act2 : $bo\_r, bo\_am : \mid bo\_r' = m\_f - ba\_r \land bo\_am' = bo\_am + bo\_r'$
act3 : $t := t + 1$
end
Event $BO\_OFF \widehat{=}$
refines $INFU\_PROC$
when
grd1 : $power = on$
grd2 : $infu\_stat = working$
grd3 : $bo\_am \geq bo\_req$
grd4 : $bo\_stat = on$
then
act1 : $rate := m\_f - bo\_r$
act2 : $bo\_stat := off$
act3 : $bo\_r := 0$
act4 : $bo\_req := 0$
act5 : $bo\_am := bo\_am$
end
Event $TICK\_TOCK \widehat{=}$
when
grd1 : $infu\_stat = working$
grd2 : $(at\_ba \cup at\_tba \cup at\_exbo) \neq \varnothing \Rightarrow (t < min(at\_ba \cup at\_tba \cup at\_exbo) \lor infu\_stat = paused)$
grd3 : $t \neq 86400 * day$
grd4 : $bo\_stat = off$
then
act1 : $t := t + 1$
end
Event $BA\_T\_RESET \widehat{=}$
when
grd1 : $t = 86400 * (day + 1)$
grd2 : $bp\_set = TRUE$
grd3 : $power = on$
grd4 : $(at\_ba \cup at\_tba \cup at\_exbo) \neq \varnothing \Rightarrow 86400 * (day + 1) \leq min(at\_ba \cup at\_tba \cup at\_exbo) \lor infu\_stat = paused$
then
act1 : $at\_ba := ran(\lambda i \cdot i \in dom(bp) \mid 86400 * (day + 1) + i)$
act2 : $day := day + 1$
end
Event $INFU\_STOP \widehat{=}$
refines $INFU\_STOP$
when
grd1 : $\mathtt{power} = \mathtt{on}$

grd2 : infu_stat = working ∨ infu_stat = paused

then

act1 : rate := 0
act2 : infu_stat := stopped
act3 : prime := FALSE
act4 : *ba_r* := *0*
act5 : *ba_stat* := *off*
act6 : *tba_r* := *0*
act7 : *tba_t* := *0*
act8 : *tba_stat* := *off*
act9 : *bo_r* := *0*
act10 : *bo_stat* := *off*
act11 : *bo_req* := *0*
act12 : *bo_am* := *0*
act13 : *exbo_a* := *0*
act14 : *exbo_t* := *0*
act15 : *exbo_req* := *0*
act16 : *exbo_r* := *0*
act17 : *exbo_stat* := *off*
act18 : *t* := *0*
act19 : *day* := *0*
act20 : *at_ba* := ∅
act21 : *at_tba* := ∅
act22 : *at_exbo* := ∅
act23 : *exbo_set* := *FALSE*

end

Event *POWER_OFF* ≙
refines *POWER_OFF*
END

Bibliography

[AAS05] A.D. Ames, A. Abate, and S. Sastry. Sufficient Conditions for the Existence of Zeno Behavior. *2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on Decision and Control*, pages 696–701, December 2005.

[ABH+10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an Open Toolset for Modelling and Reasing in Event-B. *International Journal on Softwore Tools for Tethnology Transfers*, Volume 12(No. 6):pages 447–466, April 2010.

[Abr96] Jean-Raymond Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, 1996.

[Abr10] Jean-Raymond Abrial. *Modelling in Event-B: system and software engineering.* Cambridge University Press, first edition, 2010.

[AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, Volume 126:pages 183–235, 1994.

[BFRR10] J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, and A. Roth. Patterns for Modelling Time and Consistency in Business Information Systems. *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 105–114, 2010.

[But09a] Michael Butler. Modelling Guidelines for Discrete Control Systems. *Deploy deliverable d15, d6.1 advances in methods public document*, July 7th 2009.

[But09b] Michael Butler. Using Event-B Refinement to Verify a Control Strategy (Unpublished). ECS, University of Southampton, May 2009.

[CM08] Dominique Cansell and Dominique Méry. The event-b modelling method: Concepts and case studies. *Logics of Sepcification Languages*, pages pages 47–152, 2008.

[CM09] Christophe and Mathieu. *Animator of B System Model in the Rodin Platform*. May 2009. Electronically available at `http://www.wiki.event-b.org/index.php/AnimB`.

[CMR07] Dominique Cansell, Dominique Méry, and Joris Rehm. Time Constraint Patterns for Event B Development. *LNCS*, Volume 4355:pages 140–154, 2007.

[Dep08] *Deploy Project*. February 2008. Electronically available at `http://www.deploy-project.eu/`.

[DLPPCC86] David Lorge Parnas and Paul C. Clements. A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering*, Vol. SE-12(No.2), February 1986.

[FDA10] FDA. *Safety Requirements for Generic Insulin Infusion Pump Software*. March 2010. Electronically available at `http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm202511.htm`.

[For91] Mooz H. Forsberg, K. The Relationship of Systems Engineering to the Project Cycle. First Annual Symposium of the National Council on Sesteems Engineering (NCOSE), October 1991.

[GT08] R. Goebel and A.R. Teel. Zeno Behavior in Homogeneous Hybrid Systems. *2008. CDC 2008. 47th IEEE Conference on Decision and Control*, pages 2758 –2763, December 2008.

[JP10] Ying Jin and David Lorge Parnas. Defining the Meaning of Tabular Mathematical expressions. *Science of Computer Programming*, Volume 75:pages 980–1000, November 2010.

[JW05] Ryszard Janicki and Alan Wassyng. Tabular Expressions and Their Relational Semantics. *Fundam. Inf.*, Volume 67(No. 0169-2968):pages 343–370, March 2005.

[KR95] B. Krishnamurthy and D.S. Rosenblum. Yeast: a general purpose event-action system. *Software Engineering, IEEE Transactions on*, Volume 21(No. 10):pages 845–857, October 1995.

[LB08] Michael Leuschel and Michael Butler. Prob : an automated analysis toolset for the B method. *Iinternational Journal on Software Tools for Tethnology Transfer (STTT)*, Volume 10:pages 185–203, 2008.

[LV95] Nancy Lynch and Frits Vandraager. Forward and Backward Simulations - Part II: Timing-Based Systems. *Information and Computation*, Volume 128, 1995.

[MS10] Dominique Méry and Neeraj Kumar Singh. Technical Report on Formal Development of Two-Electrode Cardiac Pacing System. HAL: inria-00465061, February 2010.

[PM95] David Lorge Parnas and Jan Madey. Functional Documents for Computer Systems. *Science of Computer Programming*, Volume 25:pages 41–61, 1995.

[Reh06] Joris Rehm. A Methods to Refine Time Constraints in Event B Framework. In Stephan Merz and Tobias Nipkow, editors, *Automatic Verification of Critical Systems - AVoCS 2006*, pages 173–177, Nancy/France, September 2006.

[rod09] *Rodin Platform and Plug-in Installation.* April 2009. Electronically available at `http://www.event-b.org/install.html`.

[SPHB10] Renato Alexandre Silva, Carine Pascal, Thai Son Hoang, and Michael Butler. Decomposition Tool for Event-B. In Proceedings of the Workshop on Tool Building in Formal Methods - ABZ Conference, Orford, Canada, February 2010.

[XM11] Hao Xu and Tom Maibaum. An Event-B Approach to Timing Issues Applied to the Generic Insulin Infusion Pump. *To appear in LNCS in 2012*, FHIES 2011.

[YZPLJRJ10] Yi Zhang, Paul L. Jones, and Raoul Jetley. A hazard analysis for a generic insulin infusion pump. *Journal of Diabetes Science and Technology*, Volume 4(Issue 2):pages 263–283, March 2010.