Synthesis of Refinement Maps for Real-Time Object Code Verification

Eman M. Al-qtiemat*, Sudarshan K. Srinivasan*, Zeyad A. Al-Odat*, Sana Shuja†
*Electrical and Computer Engineering, North Dakota State University,
Fargo, ND, USA
†Department of Electrical Engineering, COMSATS University,
Islamabad, Pakistan
Emails: *eman.alqtiemat@ndsu.edu, *sudarshan.srinivasan@ndsu.edu, *zeyad.alodat@ndsu.edu,
†SanaShuja@comsats.edu.pk

Abstract—**Refinement-based verification is a formal verification method, it is considered as a very scalable approach for dealing with low-level artifacts such as real-time object code verification. Two main obstacles prevent implementing the refinement-based verification; firstly, it requires formal specification in transition system form while most specifications are of informal or semi-formal form. To solve this issue, we already proposed synthesising procedures to transform both functional and timing requirements from natural language form into formal specifications, our approach was successfully applied on insulin pump safety requirements. Secondly, the verification process requires a construction of refinement map, which is a function maps implementation states (the artifact to be verified) onto specification states. Actually, constructing refinement maps often requires deep understanding and intuitions about the specification and implementation, it is shown very difficult to construct refinement maps manually. To go over this obstacle, the construction of refinement maps should be automated. As a first step toward the automation process, we manually developed refinement maps for various safety properties concerning the software control operation of insulin pumps. In addition, we identified possible generic templates for construction of refinement maps. To complement our previous work, this paper is built on refinement maps and refinement maps templates proposed previously to automate the construction of refinement maps. Synthesising procedures of refinement maps for functional and timing specifications are proposed. In addition, this paper shows more results of formal specifications and their suggested refinement map functions for timing requirements. Our work uses safety requirements of generic infusion pump model as heuristic data.**

Keywords–Formal verification; Synthesising of refinement maps; Formal specifications; Refinement-based verification.

I. INTRODUCTION

Software safety is one of the key challenges facing the design process [1] of safety-critical embedded systems such as medical devices [2]. For example, infusion pump (a medical device that delivers medication such as pain medication, insulin, cancer drugs etc., in controlled doses to patients intravenously) has 54 class 1 recalls related to software issued by the US Food and Drug Administration (FDA) [3]. Class 1 means that the use of the medical device can cause serious adverse health consequences or death.

Despite the fact that testing is the dominant verification technique currently used in commercial design cycles [4], testing can only show the presence of faults, but it never proves their absence [5]. Alternate verification processes should be applied to the software design in conjunction with testing to assure system correctness and reliability. Formal verification can address testing limitations by providing proofs of correctness for software safety. Intel [6], Microsoft [7] and [8], and Airbus [9] have successfully applied formal verification processes.

Refinement-based verification [10] is a formal verification technique that has been demonstrated to be effective for verification of software correctness at the object code level [11]. To apply refinement-based verification, software requirements should be expressed as a formal model. Previously, we have proposed a novel approach to synthesize formal specifications from natural language requirements [12], and in a later work, we have also addressed timing requirements and specifications [13].

Our verification approach is based on the theory of Well-Founded Equivalence Bisimulation (WEB) refinement [10]. In the context of WEB refinement, both the implementation and specification are treated as Transition Systems (TSs). If every behavior of the implementation is matched by a behavior of the specification and vice versa, then the implementation behaves correctly as prescribed by the specification. However, this is not easy to check in practice as the implementation TS and specification TS can look very different. The specification states obtained from the software requirements are marked with atomic propositions (predicates that are true or false in a given state). The implementation states are states of the microcontroller that the object code program modifies. As such, the microcontroller states includes registers, flags, and memory. The various possible values that these components can have during the execution of the object code program gives rise to the many millions of states of the implementation. To overcome this difference, WEB refinement uses the concept of a refinement map, which is a function that provided an implementation state, gives the corresponding specification state. Historically, one of the reasons that refinement-based verification is much less explored than other formal verification paradigms such as model checking is that the construction of refinement maps often requires deep understanding and intuitions about the specification and implementation [14]. However, once a refinement map is constructed, the benefit is that refinement-based verification is a very scalable approach for dealing with low-level artifacts such as real-time object code verification. In our previous paper [1], we have build refinement maps corresponding to formal specifications related to infusion pump safety and we have also proposed three possible generic refinement map templates, which is the first step toward automating the construction of refinement maps. This paper is based on our previous work, we propose synthesising procedures of refinement maps for both functional and timing requirements, the new procedure allow an expert user intervention to assure the correctness of the system. The remainder of this paper is organized as follows. Section II summarizes background information. Section III details related work. Section IV describes the refinement maps and refinement map templates. Section V shows the proposed synthesis of refinement maps for system requirement. Conclusions and direction for future work are noted in Section VI.

II. Background

This section explores the definition of transition systems, the definition of refinement-based verification, and the synthesis of formal specifications as key terms related to our work.

A. Transition Systems

As stated earlier, transition systems (TSs) are used to model both specification and implementation in refinement-based verification. TSs are defined below.

Definition 1: A TS $M = \langle S, R, L\rangle$ is a three tuple in which S denotes the set of states, $R \subseteq SXS$ is the transition relation that provides the transition between states, and L is a labeling function that describes what is visible at each state.

States are marked with Atomic Propositions (APs), which are predicates that are true or false in each state. The labeling function maps states to the APs that are true in every state. An example TS is shown in Figure 1. Here S = {S1, S2, S3, S4}, R = {(S1, S2), (S2, S4), (S4, S3), (S3, S4), (S3, S2), (S1, S3)} and, $L(S2)$ represents the atomic propositions that are true for the S2 state.

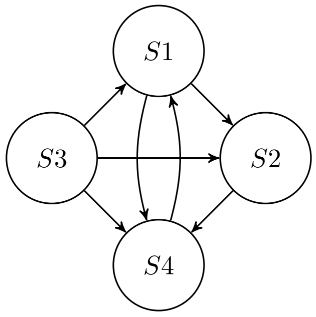


Figure 1. An example of a transition system (TS).

B. Timing Transition Systems

Some applications have requirements with timing conditions on the state's transitions called as timing requirements. Timing requirements explain the system behaviour under some timing constraints. Timing constraints are very important especially if we deal with a critical real time systems. As mentioned in the previous section (Section II-A), transition systems are used to represent the implementation and specification in refinement-based verification, however they do not contain timing requirements. Hence, in the verification of real time systems that contain timing constraints, timing transition systems (TTSs) [11] are used to represent the implementation and specification.

Definition 2: A TTS $M_t = \langle S, R_t, L\rangle$ is a three tuple in which S denotes the set of states and L is a labeling function that describes what is visible at each state. The state transition R_t has the form of $\langle x, y, l_t, u_t\rangle$ where $x, y \in S$ and $l_t, u_t \in N$ represents the lower and upper bounds as the timing condition for the transition.

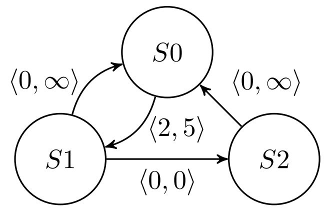


Figure 2. An example of a timing transition system (TTS).

Figure 2 shows an example of a timing transition system that consists of three states { S0, S1, S2 }, for instance; if the system is in state $S0$ it can go to state $S1$ only between 2 and 5 units of time, while going from $S1$ to $S2$ the time is zero meaning that it should happen immediately, going from $S2$ to $S1$ the time is zero to infinity which means that it can happen any point of time, and so on.

C. Refinement-Based Verification

Our verification approach is based on the theory of Well-Founded Equivalence Bisimulation (WEB) refinement. A detailed description of this theory can be found in [10]. Here, we give a very high-level overview of the key concepts. WEB refinement provides a notion of correctness that can be used to check an implementation TS against a specification TS. In the context of WEB refinement, both the implementation and specification are treated as TSs. The implementation behaves correctly as given by the specification, if every behavior of the implementation is matched by a behavior of the specification and vice versa. However, this is not easy to check in practice. Implementation TS and specification TS look quite different. The implementation states are states of the microcontroller that the object code program modifies. As such the microcontroller includes many registers, flags, and memory. The various possible values that these components can have during the execution of the object code program gives rise to the many millions of states of the implementation. To overcome this difference, WEB refinement uses the concept of a refinement map, which is a function that given an implementation states, tells you what is the corresponding specification state. Once a refinement map is constructed, WEB refinement verification proceeds as follows. The idea is to look at each implementation transition. Consider an implementation transition say (w, v), where both w and v are implementation states. To be correct, the transition has two options. The first option is that this implementation transition should match a specification transition, i.e., $r(w) = s$ and $r(v) = u$, where $r()$ is the refinement map, s and u are specification states, and (s, u) is a transition of the specification. This first option is called a non-stuttering implementation transition. The second option is that $r(w) = r(v) = s$, i.e., both w and v match to the same specification state. The second option is called a stuttering implementation transition. There are a few more checks to be performed. The very nice property of WEB refinement is that it is sufficient to reason about single steps of the implementation and specification to check for correctness and find bugs. This property makes WEB refinement very applicable to deal with the complexity of object code.

D. Synthesis of Functional Formal Specifications

Our approach for development and study of refinement maps is based on the formal TS specifications. We developed a previous approach to transform functional requirements into formal specifications [11]. Since this work is closely tied to the prior work, we briefly review it here. Figure 3 summarizes the transformation procedure, the main steps are explained as follows: functional requirement is fed as an input, an English parser called Enju is used to get the parse tree the requirement. The first step of computing the TSs is to apply Atomic Proposition Extraction Rule (APER) extract the APs from the requirements. We developed three Atomic Proposition Extraction Rules (APERs) that work on the parse tree of the requirement to get an initial list of APs. The resulting list is subjected to an expert user check (User Input), where the APs might be appended, eliminated or revised based on the expert users domain knowledge. A high-level procedure for specification transition system synthesis has been proposed to compute the states and transitions using the resulting list of APs under expert user supervision. Finally, the transition system is created using the resulting list of states and transitions. The output of the procedure is a formal specification TS.

E. Synthesis of Timing Formal Specifications

Some system requirements have timing constraints which are called timing requirements. We proposed a previous approach to work on transforming timing requirements into formal specifications [13]. Figure 4 shows the main steps of the synthesising procedure. A brief description of this approach is explained as follows: Timing requirement is fed as an input of the procedure. As in the previous procedure, Enju parser is used to get the parse tree that corresponds to the entered requirement. An Atomic Proposition and Timing Constrains Extraction Rule (APTCER) is applied on the resulted parse tree to get an initial list of APs and Timing Constrains (TCs). APs and TCs are paired together and they are considered the base of a TTS. Then, a set of states are defined based on the resulting list of APs. Transitions are applied between every two states. TCs are assigned to the transitions. This procedure allows input from domain expert as shown in Figure 4. Finally, the TTS is created. The output of this procedure is a formal specification TTS.

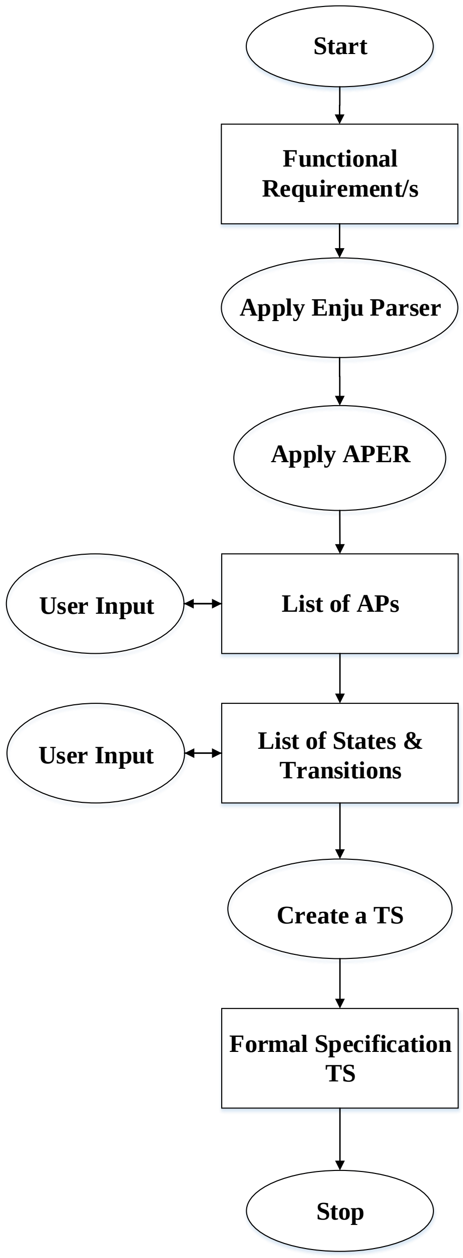


Figure 3. Formal Model synthesis procedure for Functional Requirements.

III. Related Work

This section summarizes a few works on applying refinement processes to get more concrete specifications and refinement-based verification. None of these works are applied to insulin pump formal specifications as our work. To the best of our knowledge, these are the most related state of art in this area of study.

Klein *et al.* [15] introduced a new technique called State Transition Diagrams (STD). It is a graphical specification technique that provides refinement rules, each rule defines an implementation relation on STD specification. The proposed approach was applied to the feature interaction problem. The refinement relation was utilized to add a feature or to define the notion of conflicting features.

Rabiah *et al.* [16] developed a reliable autonomous robot system by addressing A* path planning algorithm reliability issue. A refinement process was used to capture more concrete specifications by transforming High-Level specification into equivalent executable program. Traditional mathematical concepts were used to capture formal descriptions. Then, Z specification language was employed to transform mathematical description to Z schemas to get formal specifications. Z formal refinement theory was used to obtain the implementation specification.

Spichkova [17] proposed a refinement-based verification scheme for interactive real time systems. The proposed work solves the mistakes that rise from the specification problems by integrating the formal specifications with the verification system. The proposed scheme translates the specifications to a higher-order logic, and then uses the theorem prover (Isabelle) to prove the specifications. Using the refinement-based verification, this scheme validates the refinement relations between

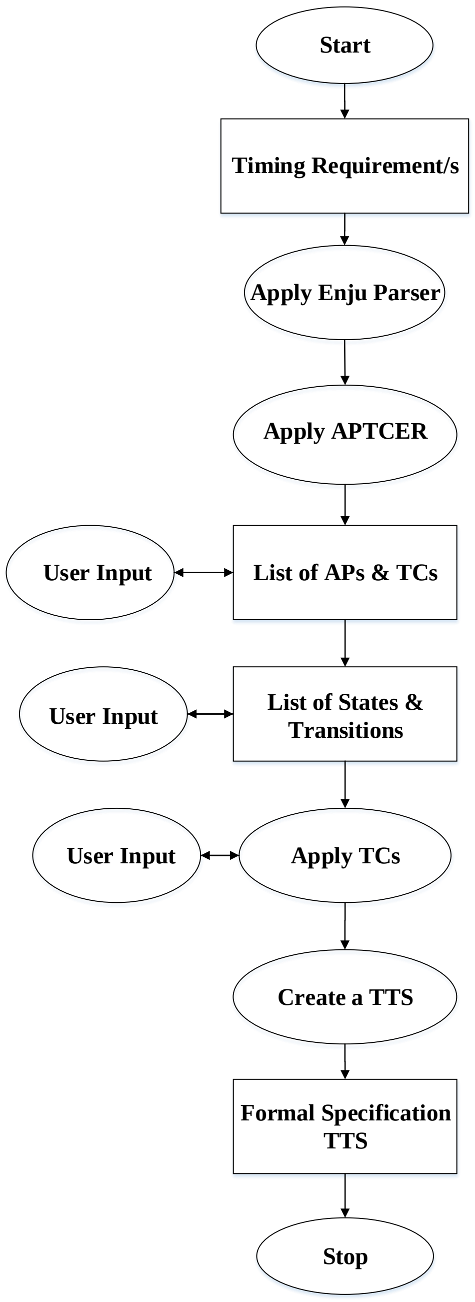


Figure 4. Formal Model synthesis procedure for Timing Requirements.

two different systems. The proposed design was tested and verified using a case study of electronic data transmission gateway.

A new approach that focuses on the refinement verification using state flow charts has been presented by Miyazawa *et al.* [18]. They proposed a refinement strategy that supports the sequential *C* implementations of the state flow charts. The proposed design benefited from the architectural features of model to allow a higher level of automation by retrieving the data relation in a calculation style and rendering the data into an automated system. The proposed design was tested and verified using Matlab Simulink SDK. Through the provided case study, the scheme was able to be scaled to different state charts problems.

Cimatti *et al.* proposed a contract-refinement scheme for embedded systems [19]. The contract-refinement provides interactive composition reasoning, step-wise refinement, and principled reuse refinements for components for the already designed or independently designed components. The proposed design addresses the problem of architectural decomposition of embedded systems based on the principles of temporal logic to generate a set of proof obligations. The testing and verification of the Wheel Braking System (WBS) case study show that the proposed design can detect the problems in the architectural design of the WBS.

Bibighaus [20] employed the Doubly Labeled Transition Systems (DLTS) to reason about possibilities security properties and refinement. This work was compared with three different security frameworks when applied to large class systems. The refinement framework in this work preserves and guarantees the liveness of the model by verifying the timing parameter of the model. The analysis results show that the proposed design preserves the security properties to a series of availability requirements.

A novel approach has been presented [21] to formally specify and analyze the certification process of Partitioning Operating Systems (POSs) by integrating refinement and ontology. An ontology of POSs was developed as an intermediate model between informal descriptions of ARINC 653 and the formal specification in Event-B. A semiautomatic translation has been implemented from the ontology and ARINC 653 into Event-B. Six hidden failures in ARINC were happened and fixed during the formal analysis in the Event-B specification. The existence of these errors has been validated in two open-source POSs: XtratuM and POK. The degree of automatic verification of the Event-B specification reached a higher level because of the ontology. By validation, they have also noticed some errors in open-source POSs. The proposed methodology has shown capability to to formalize and verify systems according to system's informal standards and requirements.

Human factors consider as the most obvious cause of failures especially when a human deals with critical systems such as nuclear and medical systems. A new methodology for developing a Human-Machine Interface (HMI) has been proposed [22], it uses a correct by construction approach. A HMI was developed independently using incremental refinement. Human interactions is dependent on testing, which can not guarantee the absence of failures. Formal method was used to assure the correctness of the human interactions. Even-B modeling language has been used to formalize the internal consistency with respect to the safety properties and events. This generic refinement strategy supports a development of the Model-View-Controller (MVC) architecture.

A specification development method and a generic security model were proposed based on refinement for ARINC Separation Kernels (KSs) [23]. A step-wise refinement framework was presented. Two levels of functional specification are developed by the refinement. Kernel initialization, inter-partition communication, two-level scheduling, and partition and process management were modeled. Isabelle/HOL theorem prover was used to carry out the formal specification and its security proofs. Mechanical check proofs were given to solve convert channels in separation kernels.

Fayolle *et al.* joined Algebraic State-Transition Diagrams (ASTD) with an Event-B specification for better understanding of the system behaviour [24]. They proposed an approach that works on incrementally refine the specification couplings, it takes the new refinement relations and consistency into consideration between data and control system specifications. This work had shown how to use two complementary languages for formal modeling, a railway CBTC-like case study were used. In addition, the principle of complementarity and consistency was explored between ASTD and B-like refinements. Separation between data and behavioural system's aspects were accomplished.

The issues of validating formal models were studied and executed using Event-B method [25]. Firstly, new techniques were created and discussed which allow model execution to be at all abstraction levels. To overcome barriers comes form non-deterministic features, users intervention such as modifying the model or providing ad hoc implementations were needed. Secondly, a new formal definition of the notion of fidelity was given, this definition assures specifying all the observable behaviors of the executable models by the non-deterministic models.

Many other papers discuss and analyze refinement concepts in the context of verifying concurrent objects. For example, Smith *et al.* in [26] provided formal link between trace refinement and linearizability, a comparison between these correctness conditions were explored. The main conclusion of this work is generally that trace refinement reveals linearizability, but linearizabilit does not reveals trace refinement. However, linearizability can reveal trace refinement but under specific conditions. Firstly, trace refinement can prove both safety and liveness properties, while linearizability can only prove safety properties. Secondly, the fact that trace refinement based on the identification of when the implementation operations are noticed to happen. They also studied these differences in the verification context of concurrent objects.

IV. Refinement Maps and Refinement Map Templates

Figures 5-11 show the formal specification TS for 8 insulin pump safety requirements, the figures also show the refinement map we developed corresponding to each specification. In this paper, formal specification TTS corresponding to 4 insulin pump timing requirements are added in Figures 12-15, we develop a refinement map for each specification as shown in the figures. The formal specifications TSs [12] and TTSs [13] were developed as part of our previous work in this area. As can be seen from the figures, each TS or TTS consists of a set of states and transitions between states. Also, each state is marked with the atomic propositions that are true. For TTSs in Figures 12-15, time bounds conditions are added on each transition. Our strategy for constructing the refinement maps is as follows. A specification state can be constructed from an implementation state by determining the APs that are true in the implementation state. If a specification has n APs, then we construct one predicate function for each AP. The predicate functions take the implementation state as input and output a predicate value that indicates if the AP is true in that state or not. Thus, the collection of such predicate functions is the refinement map.

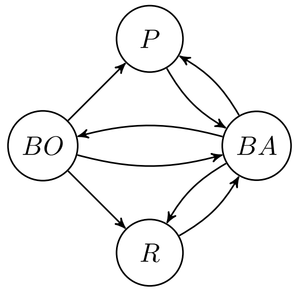
We next discuss the refinement map for the specification in Figure 5. The safety specification from [27] is as follows: ”The pump shall suspend all active basal delivery and stop any active bolus during a pump prime or refill. It shall prohibit any insulin administration during the priming process and resume the suspended basal delivery, either a basal profile or a temporary basal, after the prime or refill is successfully completed.” The APs corresponding to this safety requirement are (1) BO: active bolus delivery; (2) BA: active basal delivery; (3) P: priming process; and (4) R: refill process. The refinement map however has to account for what is happening in the implementation code and relate that to the atomic propositions.

The predicate function for BO uses several variables from the code including NB: Normal Bolus and EB: Extended Bolus as there are more than one type of Bolus dose supported by the system. So the AP BO should be true if there is a NB or an EB. NB is only a flag that indicates that a normal bolus should be in progress. The actual bolus itself will continue to occur as long as a counter that keeps track of the bolus has not reached its maximum value. Therefore, for example for a normal bolus, we use a conjunction of NB and the condition that the NB counter (NB_c) is less than its possible maximum value (NB_m). We use a similar strategy for the extended bolus as well. This refinement map template works for all processes similar to a Bolus dosage delivery, such as basal dosage delivery, priming process, and refill process. Therefore, we term this refinement map template as ”process template.” For the basal dosage (BA AP) a number of basal profiles (BPs) are possible that accounts for BP_1 thru BP_n. TB stands for temporary basal. As can be noted from Figures 6-15, the process template accounts for a large number of predicate functions corresponding to APs.

The second refinement map template is a simple one called the ”projection template,” which is used when the AP in the specification TS corresponds directly to a variable in the code. An example of the projection template can be found in Figure 6, where the User Reminder (UR) AP is mapped directly from a flag variable in the code that corresponds to the user reminder. A variation of this template is a boolean expression of Boolean variables in the code. An example of such an AP is the UIP AP in Figure 10.

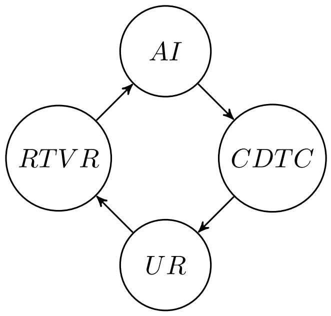
The third refinement map template is called the ”value change template,” which is used when the AP is true only when a value has changed. An example use of this template can be found in Figure 6 for the CDTC AP. CDTC corresponds to the change in drug type and concentration and is true when the drug type or concentration is changed. For the drug type change, DT is the variable that corresponds to the drug type. The question here is how to track that a value has changed. The idea is to use history variables. HDT is a history variable that corresponds to the history of the drug type, i.e., the value of the drug type in the previous cycle. If HDT is not equal to DT in a code state, then we know the drug type has changed. The inequality of HDT and DT is used to construct the predicate function. For all the safety requirements analyzed, these three refinement map templates cover all the APs.

For timed specifications, we next discuss the refinement map for the specification in Figure 13. The safety specification from [27] is as follows: ”An air-in-line alarm shall be triggered within a maximum delay time of x seconds if air bubbles larger than y μL are detected, and all insulin administrations shall be stopped.” The APs corresponding to this safety requirement



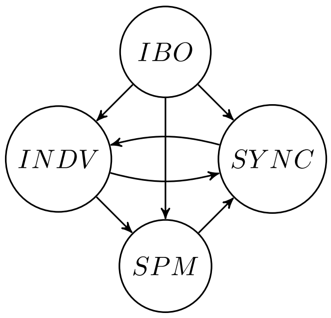
- **BO** = [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]
- **P** = P $\wedge$ $(P_c < P_m)$
- **R** = R $\wedge$ $(R_c < R_m)$
- **BA** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$]

Figure 5. A formal presentation of requirement 1.1.1 from [27] and the suggested refinement maps.



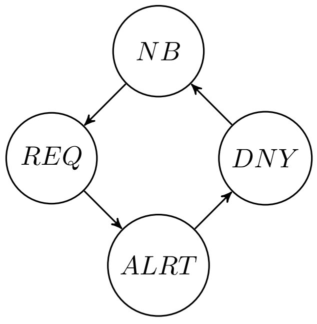
- **AI** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$] $\vee$ [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]
- **CDTC** = (DT $\neq$ HDT) $\wedge$ $(CDTC_c < CDTC_m)$
- **UR** = FLAG
- **RTVR** = (CRV $\neq$ HRV) $\wedge$ $(RTVR_c < RTVR_m)$

Figure 6. A formal presentation of requirement 1.1.3 from [27] and the suggested refinement maps.



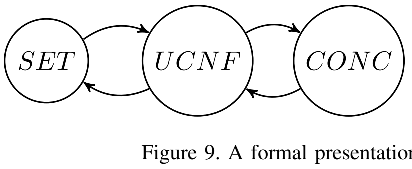
- **IBO** = [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]
- **INDV** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$] $\vee$ [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]
- **SPM** = [P $\wedge$ $(P_c < P_m)$] $\vee$ [R $\wedge$ $(R_c < R_m)$]
- **SYNC** = INCAL $\wedge$ $(INCAL_c < INCAL_m)$

Figure 7. A formal presentation of requirement 1.8.2 and 1.8.5 from [27] and the suggested refinement maps.



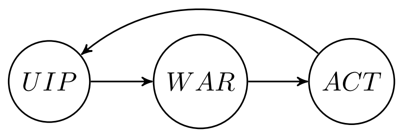
- **NB** = NB $\wedge$ $(NB_c < NB_m)$
- **REQ** = REQ-FLAG
- **ALRT** = ALRT-FLAG
- **DNY** = CALL-FUNCT

Figure 8. A formal presentation of requirement 1.3.5 from [27] and the suggested refinement maps.



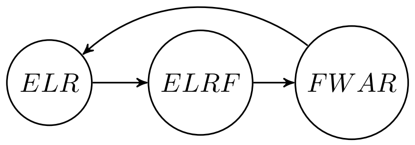
- **SET** = CLRS $\vee$ [CHNS $\wedge$ $(CHNS_c < CHNS_m)$] $\vee$ RESS
- **UCNF** = FLAG
- **CONC** = [SETT $\wedge$ $(SETT_c < SETT_m)$] $\vee$ [CHNC $\wedge$ $(CHNC_c < CHNC_m)$]

Figure 9. A formal presentation of requirement 2.2.2 and 2.2.3 from [27] and the suggested refinement maps.



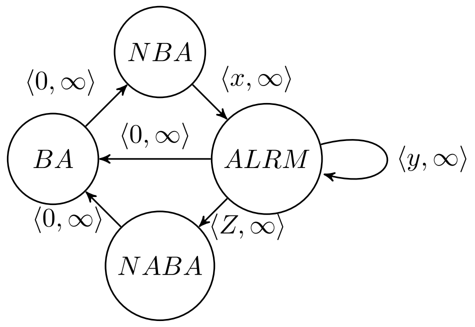
- **UIP** = BG $\vee$ TBG $\vee$ INCR $\vee$ CORF
- **WAR** = FLAG
- **ACT** = CNFI $\vee$ [CHNI $\wedge$ $(CHNI_c < CHNI_m)$]

Figure 10. A formal presentation of requirement 3.2.5 from [27] followed by the suggested refinement maps.



- **ELR** = [EL $\wedge$ $(EL_c < EL_m)$] $\vee$ [LR $\wedge$ $(LR_c < LR_m)$]
- **ELRF** = ELF $\vee$ LRF
- **FWAR** = FLAG

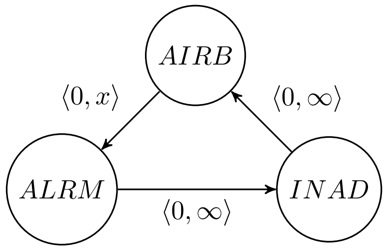
Figure 11. A formal presentation of requirement 3.2.7 from [27] followed by the suggested refinement maps.



NBA
$\langle 0,\infty\rangle$
$\langle x,\infty\rangle$
$\langle 0,\infty\rangle$
BA
$ALRM$
$\langle y,\infty\rangle$
$\langle 0,\infty\rangle$
$\langle Z,\infty\rangle$
$NABA$

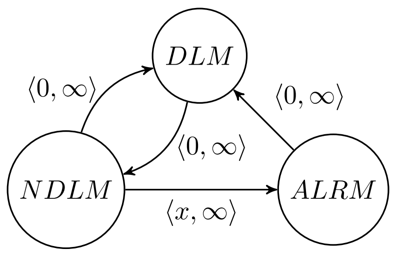
- **BA** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$]
- **NBA** = $\neg$ **BA**
- **ALRM** = ALRM-FLAG
- **NABA** = NBA $\wedge$ NA

Figure 12. A formal presentation of the timing requirement 1.2.8 from [27] and the suggested refinement maps.



- **AIRB** = AB $>$ Y
- **ALRM** = ALRM-FLAG
- **INAD** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$] $\vee$ [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]

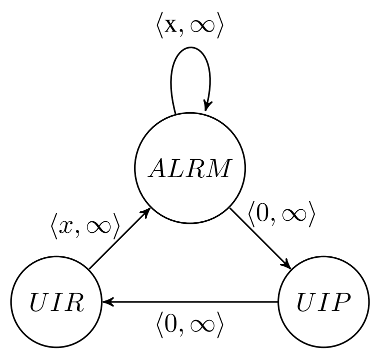
Figure 13. A formal presentation of the timing requirement 1.6.1 from [27] followed by the suggested refinement maps.



DLM
$\langle 0,\infty\rangle$
$\langle 0,\infty\rangle$
$\langle 0,\infty\rangle$
$NDLM$
$ALRM$
$\langle x,\infty\rangle$

- **DLM** = $[BP_1 \wedge (BP_{1c} < BP_{1m})] \vee [BP_2 \wedge (BP_{2c} < BP_{2m})] \vee \ldots \vee [BP_n \wedge (BP_{nc} < BP_{nm})]$ $\vee$ [TB $\wedge$ $(TB_c < TB_m)$] $\vee$ [NB $\wedge$ $(NB_c < NB_m)$] $\vee$ [EB $\wedge$ $(EB_c < EB_m)$]
- **NDLM** = $\neg$ **DLM**
- **ALRM** = ALRM-FLAG

Figure 14. A formal presentation of the timing requirement 1.8.4 from [27] followed by the suggested refinement maps.



- **ALRM** = ALRM-FLAG
- **UIR** = $RTD \lor RDC \lor RRDV \lor \dots$
- **UIP** = $STD \lor SDC \lor RDV \lor \dots$

Figure 15. A formal presentation of the timing requirement 2.2.1 from [27] followed by the suggested refinement maps.

are (1) AIRB: air bubbles; (2) ALRM: air-in-line alarm; (3) INAD: insulin administration. The predicate function for INAD uses several variables from the code including BPs; TB; NB; and EB as explained above. The AP INAD should be true if one of these variables is true and its counter variable is less than the maximum value. This AP is considered as an example use of the process template. For the ALRM AP, it is a simple example of the projection template, it should be true if its corresponding flag is true. The AIRB AP shows another variation of the value change template, which is depends on the changing value of the Air Bubbles (AB) variable. If the AB variable value is greater than Y (Y is a predefined value of the number of bubbles and it is based on the pump model), the AirB AP will be true. Table I gives the expansions for all the abbreviations used in Figures 6-15, so that the corresponding refinement maps can be comprehended by the reader.

V. Synthesis of Refinement Maps for System Requirements

This section explains new automation procedures for constructing refinement maps for both functional and timing system requirements from [27]. The first part of this work uses our previously proposed algorithms for synthesising formal specifications from natural language requirements [12] [13].

Procedure 1 shows the overall flow for computing the refinement map template for each AP in a functional requirement. A set of functional requirements in natural language form is fed as input. Three template lists are the output of the procedure, each list will contain a set of APs based on the heuristic data from the parsed trees belonging to each input requirement.

Three empty template lists are defined; projection template list (line 1), process template list (line 2), and value template list (line 3). A list for functional requirement's APs (AP_f-list) is initialized to null (line 4). Each requirement is input to the Synthesising Procedure for Functional Requirements (SPFR) (line 6) which comes up with formal specifications (explained in II-D). A function called Get_AP-list is used to obtain the resulting AP-list from the SPFR into the AP_f-list (line 7). A function called Get_Sub-tree is applied to each entry (AP_f) in the AP_f-list, this function returns the sub tree that corresponds to the AP_f from Enju parsed tree (line 9). A function called Head stores the head category of the sub tree in variable X (line 10). Check if X is of NX category, right child of X is PP, and left child of X is NX (line 11), then AP is added to the projection template list (line 12). Also if X is of VP category, right child of X is NP, and left child of X is VX (line 11), so AP is added to the projection template list as well. For the AP to be stored in the value change template (line 15), there are two cases; case 1: If X is of NX category, left child of X is VP, and right child is NX-COOD (line 14). Case 2: if X is of VP category, right child of X is CP, and left child of X is VX (line 14). If the sub tree of AP does not meet any of the previous mentioned conditions, then AP will be stored in the process template list (line 17). The procedure allows

Procedure 1 Procedure for synthesizing Refinement Maps for functional requirements

```
Require: Set of Functional Requirements
 1: Projection-Temp-list ← $\emptyset$ ;
 2: Process-Temp-list ← $\emptyset$ ;
 3: Value-Temp-list ← $\emptyset$ ;
 4: $AP_f$-list ← $\emptyset$ ;
 5: for each $Req_f \in$ Functional Requirements do
 6:     Apply_SPFR($Req_f$);
 7:     $AP_f$-list ← Get_AP-list(SPFR);
 8:     for each $AP_f \in AP_f$-list do
 9:         Sub-tree ← Get_Sub-tree ($AP_f$);
10:         $X$ = Head(Sub-tree);
11:         if [($X$ = NX) ∧ (RightChild($X$) = PP) ∧
                (LeftChild($X$) = NX)] ∨ [($X$ = VP) ∧
                (RightChild($X$) = NP) ∧ (LeftChild($X$)
                = VX)] then
12:             Projection-Temp-list ← Projection-Temp-list
                    ∪ $AP_f$;
13:         else
14:             if [($X$ = NX) ∧ (LeftChild($X$) = VP) ∧
                 (RightChild($X$) = NX-COOD) ] ∨ [($X$
                 = VP) ∧ (RightChild($X$) = CP) ∧
                 (LeftChild($X$) = VX)] then
15:                 Value-Temp-list ← Value-Temp-list ∪
                        $AP_f$;
16:             else
17:                 Process-Temp-list ← Process-Temp-list ∪
                        $AP_f$;
18:     Projection-Temp-list ← USR_IN(Projection-Temp-
            list);
19:     Value-Temp-list ← USR_IN(Value-Temp-list);
20:     Process-Temp-list ← USR_IN(Process-Temp-list);
```

TABLE I. LIST OF ABBREVIATIONS

Abbreviation	Meaning
AI	Active Infusion
CDTC	Change Drug Type and Concentration
DT	Data Type
HDT	Historical Data Type
UR	User Reminder
RTVR	Reservoir Time and Volume Recomputed
CRV	Current Reservoir Volume
HRV	Historical Reservoir Volume
IBO	Incomplete Bolus
INDV	Insulin Delivery
SPM	Suspension Mode
SYNC	Synchronization
INCAL	Insulin Calculations
REQ-FLAG	Request Flag
CALL-FUNCT	Call-Function for Calculation
SET	Settings
CLRS	Clear Settings
CHNS	Change Settings
RESS	Reset Settings
UCNF	User Confirmation
SETT	Setting the concentration
CHNC	Changing the Concentration
BG	Blood Glucose
TBG	Targeted Blood Glucose
INCR	Insulin to Carbohydrate ratio
CORF	Correction Factor
ACT	User Action
CNFI	Confirm Input
CHNI	Change Input
ELR	Event or Log Retrieving
EL	Event Logging
LR	Log Retrieving
ELRF	Event Logging or Logging Retrieving Failure
ELF	Event Logging Failure
LRF	Logging Retrieving Failure
ELF	Event Logging Failure
FWAR	Failure Warning
NBA	NO Basal delivery
NABA	No Alarm or Basal delivery
NA	No Alarm
AB	Air Bubbles
DLM	Delivery Mode
NDLM	Non-Delivery Mode
UIR	User Input Requested
RTD	Requested Time and Date
RDC	Requested Drug type and Concentration
RRDV	Requested Reloading Drug reservoir
UIP	User Input Provided
STD	Setting Time and Date
SDC	Setting Drug type and Concentration
RDV	Reloading Drug reservoir

Procedure 2 Procedure for synthesizing Refinement Maps for timing requirements

```
Require: Set of Timing Requirements
1:  Projection-Temp-list ← $\emptyset$ ;
2:  Process-Temp-list ← $\emptyset$ ;
3:  Value-Temp-list ← $\emptyset$ ;
4:  $AP_t$-list ← $\emptyset$ ;
5:  for each $Req_t \in$ Timing Requirements do
6:      Apply_SPTR($Req_t$);
7:      $AP_t$-list ← Get_AP-list (SPTR);
8:      for each $AP_t \in$ AP$_t$-list do
9:          Sub-tree ← Get_Sub-tree ($AP_t$);
10:         $X1$ = Head(Sub-tree);
11:         if [($X1$ = NX) ∧ (RightChild($X1$) = NX)
                ∧ (LeftChild($X1$) = ADJ)]∨ [($X1$ = NX)
                ∧ (RightChild($X1$) = NX) ∧ (LeftChild($X1$)
                = NP)] then
12:             Projection-Temp-list ← Projection-Temp-list ∪ $AP_t$;
13:         else
14:             if [($X1$ = NX) ∧ (RightChild($X1$) = ADJ) ∧
                    (LeftChild($X1$) = NX)] then
15:                 Value-Temp-list ← Value-Temp-list ∪ $AP_t$;
16:             else
17:                 Process-Temp-list ← Process-Temp-list ∪ $AP_t$;
18:     Projection-Temp-list ← USR_IN(Projection-Temp-list);
19:     Value-Temp-list ← USR_IN(Value-Temp-list);
20:     Process-Temp-list ← USR_IN(Process-Temp-list);
```

expert user input to the final template lists (lines 18-20), the user can modify, delete, add or exchange APs from any list if any AP is classified in the wrong list. Procedure 2 shows the overall flow for computing the refinement map template for each AP in a timing requirement. A set of timing requirements in natural language form is fed as an input. Three templates lists are the output of the procedure as in procedure I, each list will contain a set of APs based on the heuristic data from the parsed trees belonging to each input requirement. Three empty template lists are defined; projection template list (line 1), process template list (line 2), and value template list (line 3). A list for APs (AP_t-list) is initialized to null (line 4). Each requirement is input to the Synthesising Procedure for Timing Requirements (SPTR) (line 6) which comes up with formal specifications (explained in Section II-E). A function called Get_AP-list is used to obtain the resulting AP-list from the SPTR into the AP_t-list (line 7). A function called Get_Sub-tree is applied to each entry (AP_t) in the AP_t-list, this function returns the sub tree that corresponds to the AP_t from Enju parsed tree (line 9). A function called Head stores the head category of the sub tree in variable $X1$ (line 10). Check if $X1$ is of NX category, right child of $X1$ is NX, and left child of $X1$ is ADJ (line 11), then AP is added to the projection template list (line 12). Also if $X1$ is of NX category, right child of $X1$ is NX, and left child of $X1$ is NP (line 11), so AP is added to the projection template list. For the AP to be stored in the value change template (line 15), there is only one case;

If $X1$ is of NX category, right child of $X1$ is of ADJ category and left child of $X1$ is also NX (line 14). If the sub tree of AP does not meet any of the previous mentioned conditions, then it will be stored in the process template list (line 17). As in procedure I, this procedure allows expert user input to the final template lists (lines 18-20), the user can modify, delete, add or exchange APs from any list if any AP is classified in the wrong list.

VI. Conclusion and Future Work

In this paper, we have developed a process for synthesizing refinement maps. Heuristics have been developed based on the output of the Enju parser to select a refinement map template for each atomic proposition. The key ideas of our approach are the following. Firstly, the system requirement is fed as an input. Secondly, the previously proposed synthesising procedure of formal specifications is applied on the input requirement. Finally, the heuristic data from the requirement's parsed tree is used to select the suitable refinement map template. The refinement map template is either the process template, the projection template or changing value template. For future, our work can be applied on any critical device that has safety requirements, and more generic refinement map templates can be identified. In addition, the level of automation can be increased by improving the synthesis procedure.

Acknowledgment

This publication was funded by a grant from the United States Government and the generous support of the American people through the United States Department of State and the United States Agency for International Development (USAID) under the Pakistan - U.S. Science & Technology Cooperation Program. The contents do not necessarily reflect the views of the United States Government.

References

[1] E. M. Al-Qtiemat, S. K. Srinivasan, Z. A. Al-Odat, and S. Shuja, "Refinement maps for insulin pump control software safety verification," in The Eleventh International Conference on Advances in System Testing and Validation Lifecycle VALID 2019. IARIA.

[2] B. Fei, W. S. Ng, S. Chauhan, and C. K. Kwoh, "The safety issues of medical robotics," Reliability Engineering & System Safety, vol. 73, no. 2, 2001, pp. 183–192.

[3] FDA, "List of Device Recalls, U.S. Food and Drug Administration (FDA)," 2018, last accessed: 2019-10-11. [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfRES/res.cfm

[4] S. Quadri and S. U. Farooq, "Software testing-goals, principles, and limitations," International Journal of Computer Applications, vol. 6, no. 9, 2010, pp. 7–10.

[5] E. Miller and W. E. Howden, Tutorial, software testing & validation techniques. IEEE Computer Society Press, 1981.

[6] R. Kaivola et al., "Replacing testing with formal verification in intel coretm i7 processor execution engine validation," in Computer Aided Verification, 21st International Conference, CAV, Grenoble, France, June 26 - July 2, 2009. Proceedings, pp. 414–429. [Online]. Available: https://doi.org/10.1007/978-3-642-02658-4_32

[7] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "SLAM and static driver verifier: Technology transfer of formal methods inside microsoft," in Integrated Formal Methods, 4th International Conference, IFM, Canterbury, UK, April 4-7, 2004, Proceedings, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-540-24756-2_1

[8] K. Bhargavan et al., "Formal verification of smart contracts: Short paper," in Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. ACM, 2016, pp. 91–96.

[9] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, "Towards an industrial use of fluctuat on safety-critical avionics software," in International Workshop on Formal Methods for Industrial Critical Systems. Springer, 2009, pp. 53–69.

[10] P. Manolios, "Mechanical verification of reactive systems," PhD thesis, University of Texas at Austin, August 2001, last accessed: 2019-10-04. [Online]. Available: http://www.ccs.neu.edu/home/pete/research/phd-dissertation.html

[11] M. A. L. Dubasi, S. K. Srinivasan, and V. Wijayasekara, "Timed refinement for verification of real-time object code programs," in Working Conference on Verified Software: Theories, Tools, and Experiments. Springer, 2014, pp. 252–269.

[12] E. M. Al-qtiemat, S. K. Srinivasan, M. A. L. Dubasi, and S. Shuja, "A methodology for synthesizing formal specification models from requirements for refinement-based object code verification," in The Third International Conference on Cyber-Technologies and Cyber-Systems. IARIA, 2018, pp. 94–101.

[13] E. M. Al-Qtiemat, S. K. Srinivasan, Z. A. Al-Odat, and S. Shuja, "Synthesis of Formal Specifications From Requirements for Refinement-based Real Time Object Code Verification," International Journal on Advances in Internet Technology, vol. 12, Aug 2019, pp. 95–107.

[14] M. Abadi and L. Lamport, "The existence of refinement mappings," Theoretical Computer Science, vol. 82, no. 2, 1991, pp. 253–284.

[15] C. Klein, C. Prehofer, and B. Rumpe, "Feature specification and refinement with state transition diagrams," arXiv preprint arXiv:1409.7232, 2014.

[16] E. Rabiah and B. Belkhouche, "Formal specification, refinement, and implementation of path planning," in 12th International Conference on Innovations in Information Technology (IIT). IEEE, 2016, pp. 1–6.

[17] M. Spichkova, "Refinement-based verification of interactive real-time systems," Electronic Notes in Theoretical Computer Science, vol. 214, 2008, pp. 131–157.

[18] A. Miyazawa and A. Cavalcanti, "Refinement-based verification of sequential implementations of stateflow charts," arXiv preprint arXiv:1106.4094, 2011.

[19] A. Cimatti and S. Tonetta, "Contracts-refinement proof system for component-based embedded systems," Science of computer programming, vol. 97, 2015, pp. 333–348.

[20] D. L. Bibighaus, "Applying doubly labeled transition systems to the refinement paradox," Naval Postgraduate School Monterey CA, Tech. Rep., 2005.

[21] Y. Zhao, D. Sanán, F. Zhang, and Y. Liu, "Formal specification and analysis of partitioning operating systems by integrating ontology and refinement," IEEE Transactions on Industrial Informatics, vol. 12, no. 4, 2016, pp. 1321–1331.

[22] R. Geniet and N. K. Singh, "Refinement based formal development of human-machine interface," in Federation of International Conferences on Software Technologies: Applications and Foundations. Springer, 2018, pp. 240–256.

[23] Y. Zhao, D. Sanán, F. Zhang, and Y. Liu, "Refinement-based specification and security analysis of separation kernels," IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 1, 2017, pp. 127–141.

[24] T. Fayolle, M. Frappier, R. Laleau, and F. Gervais, "Formal refinement of extended state machines," arXiv preprint arXiv:1606.02016, 2016.

[25] A. Mashkoor, F. Yang, and J.-P. Jacquot, "Refinement-based validation of event-b specifications," Software & Systems Modeling, vol. 16, no. 3, 2017, pp. 789–808.

[26] G. Smith and K. Winter, "Relating trace refinement and linearizability," Formal Aspects of Computing, vol. 29, no. 6, 2017, pp. 935–950.

[27] Y. Zhang, R. Jetley, P. L. Jones, and A. Ray, "Generic safety requirements for developing safe insulin pump software," Journal of diabetes science and technology, vol. 5, no. 6, 2011, pp. 1403–1419.