Actions with Failures in Interval Temporal Logic

Arjen Hommersom and Peter Lucas

Institute for Computing and Information Sciences
Radboud University Nijmegen
{arjenh,peterl}@cs.ru.nl

Abstract. Failures are unavoidable in many circumstances. For example, an agent may fail at some point to perform a task in a dynamic environment. Robust systems typically have mechanisms to handle such failures. Temporal logic is a widely used representation language for reasoning about the behaviour of systems, although dealing with failures is not part of the language. In this paper, it is investigated how interval temporal logic can be extended with an operator describing failure. This logic has a close relationship to exception handling mechanisms in programming languages, which provides an elegant mechanism for modelling and handling failures. The approach is motivated from the context of specification of systems that have to operate in highly dynamic environments. A case study of the formal modelling and verification of the treatment of diabetes mellitus type 2 illustrates the practical usefulness of the approach.

1 Introduction

For agents that do not have a complete model of their environment or lack certain control over it, it is unavoidable that failures to perform tasks occur. Many systems require some type of robustness against these failures, e.g., robots need to make sure that their task will be accomplished, aviation systems need to make sure that the plane does not crash, etc. In agent literature, the semantics of failures have been investigated in a logical sense [11] and have been incorporated in agent programming languages [5]. Similarly, in software engineering, the use of exceptions as first-class citizens in programming languages is wide-spread.

Besides the internal aspects of a system, i.e., a program state or *mental* state of an agent, an important aspect of systems is *behaviour*, i.e., how it acts and reacts in a dynamic environment. To reason about this behaviour, mechanisms that go beyond the scope of classical predicate logic are employed. Since the late seventies, several temporal logics have been proposed to deal with specification and verification of hardware and software systems. In artificial intelligence, many of the logics dealing with actions usually contain some temporal component. In systems where failures heavily determine the final behaviour, modelling of this behaviour is more natural when failures are part of the modelling language. Moreover, we will argue that, since temporal logical formulas can be used to describe behaviour, the failure of a behaviour is best described using a sentential

operator, i.e., as a property of a (temporal) logical sentence. This contrasts with other approaches, where failure is seen as a property of primitive events and corresponds to the major contribution of this paper.

In the next section, we will first consider some motivating examples for reasoning about failure and explain why a simple solution is often unsatisfactory. Then, in Section 3, this is related to exception handling mechanisms that are found in programming languages. In Section 4, preliminaries concerning Interval Temporal Logic (ITL) are introduced, which is subsequently extended with failures in Section 5. In Section 6, we apply this to to a medical guideline that deals with the treatment of diabetes mellitus type 2 and study the formalisation and its properties. In Section 7, related work of modelling failure in AI is discussed. Finally, in Section 8, we discuss the results and future work.

2 Motivating Examples

2.1 Robbing a Bank

A well-known logic to model agents is BDI logic as proposed by Rao & Georgeff [11]. It contains modal operators BEL, GOAL, and INTEND which should be interpreted as the believe, goal and intention of the agent. Moreover, amongst other operators, it contains an operator $failed$ to describe that an event has (just) failed and temporal operators such as $\Box$ (always) and $\Diamond$ (eventually). The introduction of a $failed$ operator is motivated by Rao & Georgeff by the fact that failure may force an agent to replan or revise her plans. They give the following example:

> (...) the consequence of a thief successfully robbing a bank is quite different from a thief failing in his attempt to rob the bank, which is again different from the thief not attempting to rob the bank.

For example, it is possible to model an agent that believes that if he fails to rob the bank, then he will go to jail:

$$\mathsf{BEL}\ \Box(failed(\texttt{rob\_bank}) \rightarrow \Diamond \texttt{locked\_up})$$

which allows the agent to revise its plan after the robbery accordingly. Robbing a bank, however, is not an easy task for any intelligent agent (over 50 percent of the bank robbers are arrested in the US[1]). For example, in case of an armed robbery, it involves threatening the people inside the bank at all times, demanding money and securing a getaway. Failure to accomplish any of the subtasks will result in failure to complete the overall task. Note that failure to demand money will in some sense result in failure to rob the bank; however, it is clear that this does not necessarily lead to the consequence of going to jail. There is a need to model the different 'types' of failure associated with a goal that the agent tries to

[1] http://www.fbi.gov/ucr/cius_02/html/web/specialreport/05-SRbankrobbery.html

accomplish. More importantly, the predicate $failed$ describes failure of events; however, 'robbing a bank' is not a primitive event here, but rather a complex temporal description of several events to accomplish the overall task. Clearly, as $failed$ is a predicate on events, it cannot be used in a temporal formula. The only possibility is to define the event `rob_bank` in terms of more primitive events. The downside of this approach is that there is no mechanism to infer that failure of some of the mandatory sub-tasks will result in failure of the overall task. While it might be possible to specify this, as `rob_bank` is a complex temporal description of events, a description of its failure is, most likely, complex as well. In the next subsection, this is illustrated with some temporal patterns that may occur in medical management.

2.2 Medical Management

Suppose we are modelling an agent, typically a physician, who treats a patient. As almost all drugs may result in side-effects, it is of great importance that the agent does not over-medicate the patient. Therefore, if the disease is not directly life-threatening, management of a disease should start with a non-invasive treatment where one expects as little side-effects as possible. It is not always possible to measure beforehand if the effects of the treatment will be desirable, as this could require a test that is considered to be too invasive or because it is not known which physiological variable should be measured. As a result, a failure to treat the patient may occur, which means that subsequent actions are required.

Medical treatments are performed in sequence or in parallel. Sequential actions are typically done in case an earlier treatment fails or when a certain physiological state should be reached before a subsequent state can be effective. In such a case, failure to perform a treatment will result in a failure of the whole protocol, as it will block the successful administering of subsequent treatments. Parallel treatments occur for example when multiple drugs are prescribed at the same time. If the effects of these drugs are combined, then the combination of drugs will fail if one of the individual actions fails. If failures are not handled appropriately, it may lead to medical mismanagement, e.g., in case drugs become ineffective due to failure of other treatment components, continuing to administer these drugs is considered bad medical practice. As a consequence, failure handling plays an important role in maintaining the quality of medical management.

This idea of an implicit mechanism that "propagates" the failures throughout the management of a disease leads to the idea that such failures could be seen as exceptions that need to be handled appropriately. This idea is pursued in the next section.

3 Exception Handling

The idea of handling failures while performing a task is well-known in the context of programming languages by means of exception handling mechanisms. An exception is a failure of an operation that cannot be resolved by the operation itself

[14]. Exception handling mechanisms provides a way for a program to deal with them. Many programming languages (C++, Ada, Java, etc) now incorporate such extensive exception mechanism in order to facilitate robust applications. Typically, such a mechanism consists of two parts. There is a mechanism to *throw* an exception, which sends a signal that an exception has occurred. Second, *catching* an exception transfers control to the exception handler that defines the response that the program takes when the exception occurs. Looking at it slightly differently, one could say that the program determines the plan that is being executed, while the exception handler is able to revise this plan in case an failure occurs.

For the purpose of this paper, it is useful to summarise the semantics of exception handling mechanisms. A formal semantic model of exceptions in Java based on denotational semantics [1] as well as operational semantics [10] exists. The complete mathematical description of these mechanisms is too extensive to be discussed here, as only a small part of the semantics deals with failures. Instead, we give a more general description of the operational semantics of the exception mechanism. A state, here denoted by σ, consists of the heap, values of the local variables, and optionally an exception. Evaluation rules describe how statements change the state, typically in the form $\sigma_0 \xrightarrow{s} \sigma_1$ which denotes that the execution of statement s starting in state σ_0 can terminate in state σ_1. For exception handling, the state is extended with an exception, i.e., we then deal with assertions $\sigma_0 \xrightarrow{s} \sigma_1^e$ which means that the execution of s in σ_0 can terminate in σ_1 throwing an exception denoted by the superscript e[2]. The operational semantics is then also extended with these assertions, e.g., for sequential composition this yields the following two rules depending on whether or not a failure has occurred in the first statement:

$$\frac{\Gamma \vdash \sigma_0 \xrightarrow{s_1} \sigma_1 \quad \Gamma \vdash \sigma_1 \xrightarrow{s_2} \sigma_2}{\Gamma \vdash \sigma_0 \xrightarrow{s_1;s_2} \sigma_2} \qquad \frac{\Gamma \vdash \sigma_0 \xrightarrow{s_1} \sigma_1^e}{\Gamma \vdash \sigma_0 \xrightarrow{s_1;s_2} \sigma_1^e}$$

where Γ defines the context of the rule. Logically speaking, what we see here is that failures are propagated through the semantics of each programming structure. We will show how to incorporate this idea in terms of temporal logic in the next two sections.

4 Interval Temporal Logic

In this section, we define the necessary preliminaries of interval temporal logic (ITL) [8], which acts as the basis for our approach and can be considered a rich framework for specifying many systems. As it is quite a rich system, it can be considered a rather heavy machinery for solving the problems that were discussed in the previous section. However, it shows that the incorporation of failures in the logic can be done for a wide range of logics, such as the more common linear temporal logic (LTL), a sub-logic of ITL.

[2] abstracting from the different types of exceptions

4.1 Syntax

For the purpose of this paper, we consider the propositional part of ITL. The main difference with standard temporal logic is that interval temporal logic deals with intervals rather than time points, which makes it suitable for logic-based modular reasoning involving periods of time. In this logic there are three primary temporal constructs:

- skip: the interval is a unit interval of length 1
- $\varphi;\psi$: chop the interval into two parts, such that φ holds in the first part and ψ holds in the second part
- φ^*: decompose the interval into a (possibly infinite) number of finite intervals in which φ holds.

Given a non-empty set of propositional variables $\mathcal{P}$, the full syntax can then be given in BNF notation as follows:

$$\varphi \longrightarrow p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{skip} \mid \varphi;\varphi \mid \varphi^*$$

with $p \in \mathcal{P}$. Let true be defined as $p \vee \neg p$ and false as $\neg\mathsf{true}$, for some $p \in \mathcal{P}$. Then, the following additional linear temporal operators are defined that are used in the remainder of this paper:

$\circ\varphi$	$\triangleq$	$\mathsf{skip};\varphi$	in the next state φ
$\bullet\varphi$	$\triangleq$	$\neg \circ \neg\varphi$	if there is a next state, then in the next state φ
last	$\triangleq$	$\bullet\,\mathsf{false}$	this is the last state of the interval
finite	$\triangleq$	$\neg(\mathsf{true};\mathsf{false})$	the interval is finite
$\Diamond\varphi$	$\triangleq$	$\mathbf{finite};\varphi$	eventually φ
$\Box\varphi$	$\triangleq$	$\neg\Diamond\neg\varphi$	always φ

Other propositional connectives are defined as usual, i.e., $\varphi \vee \psi \triangleq \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi \triangleq \neg\varphi \vee \psi$, and **if** α **then** φ **else** $\psi \triangleq (\alpha \wedge \varphi) \vee (\neg\alpha \wedge \psi)$.

4.2 Semantics

Models of this logic are (possibly infinite) sequences of states, denoted by σ, i.e., $\sigma = \sigma_0, \sigma_1, \ldots$. We write $|\sigma|$ to denote one less than the length of the sequence (as usual in ITL), which is either ∞ if there are infinite number of states and otherwise some natural number n. If $\sigma = \sigma_0, \ldots, \sigma_n, \ldots, \sigma_m, \ldots$, then $\sigma^{[n,m]}$ denotes the subsequence $\sigma_n, \ldots, \sigma_m$ of σ. Let each σ_i be a function of type $\mathcal{P} \rightarrow \{\bot, \top\}$, that denotes whether an atomic proposition is either true ($\top$) or

false ($\bot$). The formal semantics is then as follows:

$$
\begin{array}{lll}
\sigma \models p & \Leftrightarrow \sigma_0(p) = \top \\
\sigma \models \neg\varphi & \Leftrightarrow \sigma \not\models \varphi \\
\sigma \models \varphi \wedge \psi & \Leftrightarrow \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma \models \mathsf{skip} & \Leftrightarrow |\sigma| = 1 \\
\sigma \models \varphi; \psi & \Leftrightarrow |\sigma| = \infty \wedge \sigma \models \varphi \text{ or} \\
 & \quad \text{there exists } n \leq |\sigma| \text{ with } \sigma^{[0,n]} \models \varphi \text{ and } \sigma^{[n,|\sigma|]} \models \psi \\
\sigma \models \varphi^* & \Leftrightarrow |\sigma| = 0 \\
 & \quad \text{or there exists } 0 = n_0 < n_1 < \ldots < n_m < |\sigma| \\
 & \quad\quad \text{with } \sigma^{[n_i,n_{i+1}]} \models \varphi \text{ for all } 0 \leq i < m \\
 & \quad\quad \text{and } \sigma^{[n_m,|\sigma|]} \models \varphi \\
 & \quad \text{or there exists infinite many } 0 = n_0 < n_1 < \ldots \\
 & \quad\quad \text{with } \sigma^{[n_i,n_{i+1}]} \models \varphi \text{ for all } 0 \leq i
\end{array}
$$

Sequences of, for example medical, actions can now be modelled quite easily, e.g., action a after b is modelled as $a; b$, provided that a and b may overlap. Repetition of this patterns could be modelled as, for example, $(a; \circ b)^*$. If, on the other hand, a must be applied for a longer period of time, this may be described as $\Box a; b$. What will happen when in this latter case $fail(b)$ holds at some point: does this constitute a failure of the complete sequence? Presumably not, as b does not necessarily have to hold by the given semantics. What if failure occurs at the last time that a holds? Then it seems to be the case that the whole sequence has failed. We will formalise this intuition in the next section.

5 Interval Temporal Action Logic with Failure

In this section, we extend the logic of ITL with actions and introduce an operator that denotes failure of the formula. We will refer to this extended logic as ITALF.

5.1 Syntax and Semantics

Let $\mathcal{A}$ be a set of actions, and $\mathcal{P}$ a set of atomic propositions. Models σ we will be working with consists of a (possible infinite) sequence of states $\sigma_0, \ldots$. Each σ_i is defined as $\langle \pi_i, \alpha_i \rangle$, where π_i is a function $\mathcal{P} \to \{\top, \bot\}$ and α_i a function $\mathcal{A} \to \{\mathsf{inactive}, \mathsf{active}, \mathsf{failed}\}$. When discussing a σ', we will write α'_i and π'_i such that $\sigma'_i = \langle \alpha'_i, \pi'_i \rangle$. Let the language be extended with actions and an operator **fail**. All semantics given by the language of ITL remains the same. Entailment of ITL will be denoted as $\models_{\text{ITL}}$ from now on and $\models$ will be understood as entailment for ITALF.

Actions are interpreted as activations, hence, negations of actions are understood as actions that are not active (i.e., inactive or failed). This is formalised as follows:

$$\sigma \models a \Leftrightarrow \alpha_0(a) = \mathsf{active}$$

For the definition of failure, we need to consider models where we abstract from the difference between inactive and activation, but instead only consider the

1. $\mathbf{fail}(\varphi) \rightarrow \mathbf{fail}(\varphi \wedge \psi)$
2. $\mathbf{fail}(\varphi \vee \psi) \rightarrow \mathbf{fail}(\varphi) \vee \mathbf{fail}(\psi)$
3. $\mathbf{fail}(\varphi) \rightarrow \mathbf{fail}(\varphi; \psi)$, where φ is objective
4. $\mathbf{fail}(\varphi) \rightarrow \mathbf{fail}(\varphi^*)$, where φ is objective

Fig. 1. Propagation of failures in ITALF, where objective formulas are formulas that do not contain any temporal operators.

difference between failures and non-failures. In order to accomplish this, we use the following models, that we denote as $\mathsf{failmodel}(\sigma)$:

Definition 1. *For all* σ, $\mathsf{failmodel}(\sigma) = \sigma'$ *if:*

- $|\sigma| = |\sigma'|$
- *for all* i *such that* $0 \leq i \leq |\sigma|$*:*
  - *for all* $p \in \mathcal{P}$*:* $\pi_i(p) = \pi'_i(p)$
  - *for all* $a \in \mathcal{A}$*: if* $\alpha_i(a) = \mathsf{failed}$ *then* $\alpha'_i(a) = \mathsf{failed}$*, otherwise* $\alpha'_i(a) = \mathsf{active}$

So $\mathsf{failmodel}(\sigma)$ describes σ where non-failures (in particular inactive actions) are interpreted as activations. We can then consider failure as a type of negation in the definition of **fail**, as follows:

$$\sigma \models \mathbf{fail}\,\varphi \Leftrightarrow \sigma \not\models \varphi \text{ and } \mathsf{failmodel}(\sigma) \not\models \varphi$$

To understand this definition, consider φ as a formula that implies that certain propositions and actions are true or false at certain moments in time, even though for some formulas, there is a choice to be made on which point in time. For atomic propositions, the definition is clear and is equivalent to the negation as $\mathsf{failmodel}(\sigma)$ does not evaluate propositions differently than σ. For actions, the situation is more complicated. First, if an action a is implied by φ at a certain moment in time, then φ fails if the action fails on that point in time, which is exactly given looking at $\neg a$ on $\mathsf{failmodel}(\sigma)$. This seems sufficient; however, consider the converse, i.e., that φ implies $\neg a$ at a certain point in time. Then, the formula fails if in fact the action is activated at that point, which corresponds to the first part of the definition. Note that by just looking at $\mathsf{failmodel}(\sigma)$, we can only derive that it must be active or inactive, however, a failure not to do an action does not correspond to this idea.

5.2 Logical Characterisation

As already mentioned in the previous section, with respect to atomic propositions p, it follows:

$$\mathbf{fail}\,p \equiv \neg p$$

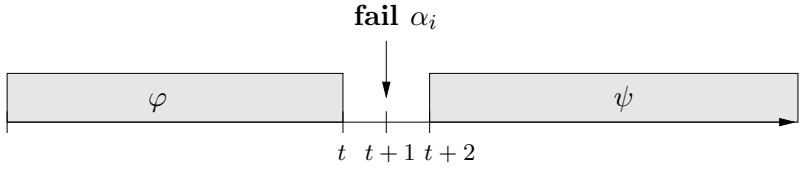

Fig. 2. Sketch of φ **orelse**$_{\{\alpha_i\}}$ ψ in case of failure of α_i.

i.e., failure to accomplish p simply means it is not true. So, the formalisation considers failure as a kind of negation. Typically, in the formalisation of medical management, we are interested in formulas such as:

$$\textbf{fail}\,(p \rightarrow a)$$

i.e., in situation described by p, the action a must be activated. According to the semantics, this is equivalent to $p \wedge \textbf{fail}\,a$, i.e., if the implication fails, then in the situation described by p, the action a indeed fails. As argued in the previous subsection, failure not to do an action a, i.e., $\textbf{fail}\,\neg a$ means that a is in fact done. Conversely $\neg\textbf{fail}\,a$ means that a is either active or inactive. Hence $\textbf{fail}\,\neg\varphi \neq \neg\textbf{fail}\,\varphi$.

In general, the definition of **fail** is such as to propagate to larger formulas. This is summarised in Fig. 1. What is interesting is that the calculus rules of the operational semantics of an imperative programming language described in Section 3 can now be understood in terms of failure inside the logic. For example, for sequential composition, the following calculus rule is sound with respect to the semantics:

$$\frac{\Gamma \vdash \textbf{fail}(\varphi)}{\Gamma \vdash \textbf{fail}(\varphi;\psi)}$$

where φ is objective, which follow directly from item (3) of Fig. 1 and modus ponens.

In order to describe acting on the basis of failure, we define an additional operator:

$$\varphi\ \textbf{orelse}_A\ \psi \triangleq \varphi \wedge \neg\textbf{last};\circ\,(\textbf{failstate}_A \wedge \circ\,\psi)$$

where

$$\textbf{failstate}_A = \bigvee_{a_i \in A} \textbf{fail}\ a_i \wedge \bigwedge_{a_i \in \mathcal{A}} \neg a_i$$

i.e., φ holds forever, or, an action fails at some point after which ψ holds. We assume that φ is true in at least a unit interval, which prevents failures to occur right away. In Fig. 2, a model where a failure occurs and is handled is sketched. The definition of the **failstate** ensures that during this time no action can be active, and thus no additional failures may occur. In some sense, this operator may be read as an exception handling mechanism where failures of 'type' A are caught in the execution of φ, such that ψ is executed when this occurs.

Finally, consider the robbing example of Subsection 2.1. The ITALF logic allows one to reason elegantly about, for example the temporal description $\textbf{fail}(\Box\mathsf{threaten}; \mathsf{flee})$. It is cumbersome to derive an equivalent formula in ITL, which has to describe that in case the fleeing fails or that threatening fails before fleeing, the robbing fails. It is not difficult to see however, that it is *possible* to write down such formula. In fact, this is true for arbitrary formulas of the ITALF language and is discussed next.

5.3 Reduction to ITL

In this subsection, we will show how ITALF can be translated to ITL. We thereby give means to exploit the proof techniques that were developed for ITL. To accomplish a reduction to ITL, additional propositional variables are required. We assume we have an infinite number of propositional variables such that we have a (unique) fresh proposition f_a, standing for failure, for each $a \in \mathcal{A}$.

Definition 2. *Given a formula φ, define $\Phi(\varphi)$ as φ where every occurrence of some action $a \in \mathcal{A}$ has been replaced with $\neg f_a$.*

Definition 3. *The reduction of a formula φ is defined on the structure of φ as follows:*

$$\begin{aligned}
\mathit{reduce}(p) &= p \\
\mathit{reduce}(a) &= a \\
\mathit{reduce}(\neg\varphi) &= \neg\mathit{reduce}(\varphi) \\
\mathit{reduce}(\varphi \wedge \psi) &= \mathit{reduce}(\varphi) \wedge \mathit{reduce}(\psi) \\
\mathit{reduce}(\mathit{skip}) &= \mathit{skip} \\
\mathit{reduce}(\varphi; \psi) &= \mathit{reduce}(\varphi); \mathit{reduce}(\psi) \\
\mathit{reduce}(\varphi^*) &= \mathit{reduce}(\varphi)^* \\
\mathit{reduce}(\textbf{\textit{fail}}\,\varphi) &= \neg\mathit{reduce}(\varphi) \wedge \neg\Phi(\mathit{reduce}(\varphi))
\end{aligned}$$

Below, we refer to $\mathsf{reduct}(\varphi)$ as the ITALF formula that is found by applying the definition exhaustively from left to right. The main result of this subsection which provides the connection between ITL and ITALF follows.

Definition 4. *Given a ITALF formula φ, intended meaning of the failure propositions is defined as follows:*

$$\mathcal{I}(\varphi) = \Box(a_0 \rightarrow \neg f_{a_0} \wedge \cdots \wedge a_n \rightarrow \neg f_{a_n})$$

where $\{a_0, \ldots, a_n\} \subseteq \mathit{actions}(\varphi)$, such that $\mathit{actions}(\varphi)$ is defined as the set of those $a \in \mathcal{A}$ that is a sub-formula of φ.

Note that $\mathcal{I}(\varphi)$ is finitely bounded by actions in a formula, which is important as the total number of actions in the language may be infinite.

Then, we have the following result:

Theorem 1. $\models \varphi$ *iff* $\mathcal{I}(\varphi) \models_{\mathrm{ITL}} \mathit{reduct}(\varphi)$

The proof of this theorem can be found in Appendix A.

5.4 Robbing the Bank Revisited

As an illustration of the theory introduced above, we revisit the example discussed in Section 2.1. Suppose we would model robbing the bank as follows:

$$\texttt{rob\_bank} \triangleq (\Box\texttt{threaten} \wedge \Diamond\texttt{collect\_money} \wedge \textbf{finite}); \texttt{get\_away}$$

i.e., personnel is threatened for a finite amount of time, money is collected, after which it is necessary to get away. Accepting the semantics of failures as presented in this paper, we can now directly talk about failure of the logical sentence above, possibly in combination with '$\Box\neg$**fail** `collect_money`', as this action cannot fail in the sense discussed in Section 2.1, i.e., failure to collect money does not lead to an arrest.

To illustrate what it then means to fail to rob the bank in the language of ITL, one can take the reduction of the sentence (we will omit the definition of failure propositions defined by $\mathcal{I}$ in the formulas below, i.e., we will only consider faithful models, see Appendix A) resulting in:

$$\cdots \wedge \neg((\Box\neg f_{\texttt{threaten}} \wedge \Diamond\neg f_{\texttt{collect\_money}} \wedge \textbf{finite}); f_{\texttt{get\_away}})$$

Now supposing that the money is collected, this can be further simplified using the semantics of the ITL operators, finally yielding models for which holds:

$$\forall n \leq |\sigma| \colon (\exists i \colon (0 \leq i \leq n \text{ and } \sigma_i \models f_{\texttt{threaten}}) \text{ or } \sigma_n \models f_{\texttt{get\_away}})$$

i.e., either in the first part the threatening fails or, otherwise, the robber fails to get away, which is arguably the intended meaning of failing to rob a bank in this example.

This, however, is not easily specified in future-time temporal logic. While we can obviously express this with the negated chop formula above, negated chop formulas are difficult to interpret. Using a more standard linear temporal operator, **until** (see e.g., [15]), the simplified formula can be rephrased as:

$$\neg((\neg f_{\texttt{threaten}})\ \textbf{until}\ (\neg f_{\texttt{get\_away}} \wedge \neg f_{\texttt{threaten}}))$$

which is possibly slightly more understandable. Nevertheless, if the original temporal specification is more complicated than the rather simple example that we provide here, modelling failing behaviour quickly becomes a difficult task.

6 Application to a Medical Guideline

6.1 Introduction

As a more elaborate application of failures, we consider clinical guidelines, which are extensive documents advising clinicians appropriate management of disease. The guideline shown in Fig. 3 is part of the guideline for general practitioners for the treatment of diabetes mellitus type 2 (DM2) [12], which aims at controlling the level of glucose in the blood. The knowledge in this fragment concerns

– Step 1: diet
– Step 2: if Quetelet Index (QI) $\leq$ 24, prescribe a sulfonylurea drug; otherwise, prescribe a biguanide drug
– Step 3: combine a sulfonylurea drug and biguanide (replace one of these by a α-glucosidase inhibitor if side-effects occur)
– Step 4: insulin

Fig. 3. Tiny fragment of a clinical guideline on the management of diabetes mellitus type 2. If one of the steps $k = 1, 2, 3$ is ineffective (fails), the management moves to step $k + 1$.

information about order and time of treatment (e.g., sulfonylurea in step 2), about patients and their environment (e.g., Quetelet index lower than or equal to 27), and finally which drugs are to be administered to the patient (e.g., a sulfonylurea drug).

In order to reason about the quality of guidelines, we require additional medical knowledge, which is based on a methodology for checking quality medical guidelines proposed in [6]. While some of the below formalisation is also discussed in that paper, here, the guideline is formalised in temporal logic, rather than a specialised guideline representation language. In particular, we focus on the issue of failure of treatments.

6.2 Modelling of Medical Knowledge

In order to represent the medical knowledge, a specific language is defined in this section. We restrict ourselves to the knowledge which concerns itself with the primary aim of a guideline, which is to have a certain positive effect on a patient. To establish that this is indeed the case, knowledge concerning the physiology of a patient is required. This is here formalised as a causal model describing effects of the treatment.

We are interested in the prescription of drugs, taking into account their mode of action. Abstracting from the dynamics of their pharmacokinetics, this can be formalised in logic as follows:

$$(d \wedge r) \rightarrow \circ\,(m_1 \wedge \cdots \wedge m_n) \tag{1}$$

where d is the name of a drug, r is a (possibly negative or empty) *requirement* for the drug to take effect, and m_k is a mode of action, such as decrease of release of glucose from the liver, which holds at all future times.

Note that we assume that drugs are applied for an instant, here formalised as 'next'. This is reasonable if we think of the time instants as unspecified periods of time where certain propositions hold. Synergistic effects and interactions amongst drugs can also be formalised along those lines, as required by the guideline under consideration. This can be done either by combining their joint mode of action, by replacing d in the formula above by a conjunction of drugs, or

(1) $\mathit{insulin} \rightarrow$
$\circ\ (\mathit{uptake}(\mathit{liver}, \mathit{glucose}) = \mathit{up} \wedge \mathit{uptake}(\mathit{peripheral\text{-}tissues}, \mathit{glucose}) = \mathit{up})$
(2) $\mathit{uptake}(\mathit{liver}, \mathit{glucose}) = \mathit{up} \rightarrow \mathit{release}(\mathit{liver}, \mathit{glucose}) = \mathit{down}$
(3) $\mathrm{SU} \wedge \neg \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{exhausted}) \rightarrow \circ\ \mathit{secretion}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{up}$
(4) $\mathrm{BG} \rightarrow \circ\ \mathit{release}(\mathit{liver}, \mathit{glucose}) = \mathit{down}$
(5) $\mathit{diet} \wedge \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{normal} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})$
(6) $(\circ\ \mathit{secretion}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{up} \wedge \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{subnormal} \wedge$
$\mathrm{QI} \leq 27 \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})$
(7) $(\circ\ \mathit{release}(\mathit{liver}, \mathit{glucose}) = \mathit{down} \wedge \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{subnormal} \wedge$
$\mathrm{QI} > 27 \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})$
(8) $((\circ\ \mathit{release}(\mathit{liver}, \mathit{glucose}) = \mathit{down} \vee \circ\ \mathit{uptake}(\mathit{peripheral\text{-}tissues}, \mathit{glucose}) = \mathit{up}) \wedge$
$\mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{nearly\text{-}exhausted} \wedge \circ\ \mathit{secretion}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{up} \wedge$
$\mathrm{Condition}(\mathit{hyperglycaemia})) \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})$
(9) $(\circ\ \mathit{uptake}(\mathit{liver}, \mathit{glucose}) = \mathit{up} \wedge \circ\ \mathit{uptake}(\mathit{peripheral\text{-}tissues}, \mathit{glucose}) = \mathit{up} \wedge$
$\mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{exhausted} \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \rightarrow$
$\circ\ (\mathrm{Condition}(\mathit{normoglycaemia}) \vee \mathrm{Condition}(\mathit{hypoglycaemia}))$
(10) $(\mathrm{Condition}(\mathit{normoglycaemia}) \oplus \mathrm{Condition}(\mathit{hypoglycaemia}) \oplus$
$\mathrm{Condition}(\mathit{hyperglycaemia})) \wedge \neg\ (\mathrm{Condition}(\mathit{normoglycaemia}) \wedge$
$\mathrm{Condition}(\mathit{hypoglycaemia}) \wedge \mathrm{Condition}(\mathit{hyperglycaemia}))$

Fig. 4. Background knowledge $\mathcal{B}_{\mathrm{DM2}}$ of diabetes mellitus type 2. An action α holds iff drug x is being administered at that moment in time. The $\oplus$ operator denotes the exclusive OR operator.

by reasoning about modes of actions. As we do not require this feature for the clinical guideline considered in this chapter, we will not go into details.

The modes of action m_k can be combined, together with an *intention* n (achieving normoglycaemia, i.e., normal blood glucose levels, for example), a particular patient *condition* c, and *requirements* r_j for the modes of action to be effective:

$$(\circ\, m_{i_1} \wedge \cdots \wedge \circ\, m_{i_m} \wedge r_1 \wedge \cdots \wedge r_p \wedge c) \rightarrow \circ\, n \tag{2}$$

For example, if the mode describes that there is a stimulus to secrete more insulin and the requirement that sufficient capacity to provide this insulin is fulfilled, then the amount of glucose in the blood will decrease.

The fragment of DM2 is relatively simple, however, diabetes is in fact a complicated disease: various metabolic control mechanisms are deranged and many different organ systems may be affected by the disorder. Pathophysiologically, there are two main phenomena, namely, insufficient secretion of the hormone insulin due to a decreased production of insulin by *B cells* in the Langerhans islets of the *pancreas*, and insulin resistance in liver, muscle and fat tissue. Parts of these mechanisms are described in more detail in [6]. These physiological mechanisms were modelled in temporal logic, which is described in Fig. 4.

6.3 Modelling of the Guideline

In this paper, we mainly focus on the modelling of the guideline fragment of Fig. 3. The possible actions that can be performed is the set $\mathcal{A}$ consisting of $\{diet, SU, BG, insulin\}$. Each treatment A is a subset of $\mathcal{A}$. Treatment changes if a treatment has failed, which can be conveniently be formalised in ITALF. The main structure of the guideline, denoted by $\mathcal{M}$, is then:

$$\begin{array}{l}\Box\ \text{treatment} = \{\text{diet}\} \\ \mathbf{orelse}_{\{\text{diet}\}}\ (\mathbf{if}\ \text{QI} < 27\ \ \mathbf{then}\ (\Box\text{treatment} = \{\text{SU}\}) \\ \qquad\qquad\qquad\qquad\ \ \mathbf{else}\ (\Box\text{treatment} = \{\text{BG}\}) \\ \qquad\qquad \mathbf{orelse}_{\{\text{SU,BG}\}}\ (\Box\text{treatment} = \{\text{SU}, \text{BG}\} \\ \qquad\qquad\qquad\qquad\ \ \mathbf{orelse}_{\{\text{SU,BG}\}}\ \Box\text{treatment} = \{\text{insulin}\}))\end{array}$$

where each term treatment $= A$ is an abbreviation for:

$$\bigwedge (\{\alpha \mid \alpha \in A\} \cup \{\neg\alpha, \neg\mathbf{fail}\ \alpha \mid \alpha \in (\mathcal{A} \setminus A)\})$$

i.e., the actions in A are activated, and all other actions are inactive (i.e., false and have not failed). This formalisation includes the handling of the failures in some sense, however, we also need to define in which cases these failures occur. One can think of this as 'throwing' the exceptions during the management of the disease. Define an abbreviation for this as follows:

$$\mathbf{fails}\,\varphi \triangleq \circ\,\mathbf{fail}\,\varphi$$

The guideline does not specify what amount of time is allowed to pass before it can be concluded that the treatment is not effective. Clearly, if a failure occurs immediately, then patients will all receive insulin treatment. Here, we assume the property of the background knowledge that relevant effects with respect to the condition of the patient are known in the next state. Hence, decisions whether the treatment fails can be taken after one step in the execution. These failure axioms are denoted as $\mathcal{F}$ and formalised as follows:

$$\Box\ (\alpha_i \rightarrow \circ\ ((\alpha_i \wedge \text{Condition}(\textit{hyperglycaemia})) \leftrightarrow \mathbf{fails}\,\alpha_i))$$

for all $\alpha \in \mathcal{A}$.

6.4 Verification

Several tools for ITL have been developed, such as the interpreter Tempura [9] and support for ITL in the theorem prover PVS [3]. For our experiments, we have used the KIV system, an interactive theorem prover, designed for program verification and capable of reasoning about algebraic specifications using classical, dynamic and (interval) temporal logic. The main proof strategy for temporal logic is symbolic execution with induction. Symbolic execution unwinds formulas, e.g.,

$$\Box\ \varphi \Leftrightarrow \varphi \wedge \circ\ \Box\ \varphi$$

and induction is used to proof reason about recurring temporal states. Its theoretical background is described extensively in [2]. Below, we will write sequents $\Gamma \vdash \Delta$ to denote $\mathcal{I}(\Gamma \cup \Delta) \vdash_{\mathrm{KIV}} \mathsf{reduce}(\bigwedge \Gamma \rightarrow \bigvee \Delta)$, where $\vdash_{\mathrm{KIV}}$ denotes the deductibility relation defined by the sound (propositional and temporal) inference rules implemented in KIV.

In the specification of properties presented, we made use of algebraic specification to specify the variables in the background knowledge, though it could be translated to propositional logic if necessary. Furthermore, we made use of some additional variables to represent each treatment (e.g., '*treatmentdiet*' defined as 'treatment = {diet}'), and both failure-states. In practice, this makes the proofs more manageable. The relationship between the actions and these additional variables are defined appropriately in the system, i.e., all the additional propositional variables could be replaced by actions and failure of actions.

Example 1: Diet may be applied indefinitely The first example is the following property. Let $\mathcal{B}_{DM2}$ be the background knowledge, $\mathcal{M}$ be the guideline given in Section 6.3, and $\mathcal{F}$ failure axioms defined in Section 6.3, then:

$$\mathcal{B}_{DM2},\ \mathcal{M}, \mathcal{F},\ \Box\ \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{normal} \\ \vdash \Box \bullet \mathrm{Condition}(\mathit{normoglycaemia})$$

i.e., in case the patient has B cells with sufficient capacity to produce insulin, then diet is sufficient for lowering the level of glucose in the blood. As only the failure of diet is relevant in the proof, $\mathcal{M}$ can be weakened to:

$$(\Box \mathit{treatmentdiet}) \wedge \neg\ \mathbf{last};\ f_{\mathrm{diet}}$$

Symbolic execution, in the context of the background knowledge, leads to the situation where:

$$(\Box \mathit{treatmentdiet};\ f_{\mathrm{diet}}) \wedge \mathrm{Condition}(\mathit{normoglycaemia})$$

Since we have Condition(*normoglycaemia*), it can be derived that *diet* does not fail, thus in the next step it can be derived that the condition is still normoglycaemia, which is exactly the same situation as we had before. By induction, we can then reason that this will *always* be the case. A more detailed proof can be found in Appendix B.

Example 2: Reasoning about the patient in case of failure Guidelines are not applied blindly by physicians, as the physician has to make a decision for an individual patient on the basis of all known information. As a consequence, a physician might be interested in reasons of failure. Suppose we have an arbitrary patient, then we can prove the following:

$$\mathcal{B}_{DM2},\ \mathcal{M}, \mathcal{F}\ \vdash \mathbf{fail}(\Box\ \mathit{diet}) \rightarrow \Diamond \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) \neq \mathit{normal}$$

i.e., if *always* applying diet fails, then apparently the patient has non-normal capacity of its B cells at a certain moment in time. $\mathcal{M}$ is needed here to derive that in case diet stops, a failure has occurred rather than a non-failing termination of diet. Proving this in KIV is similar as the previous example.

Example 3: Level of sugar in the blood will decrease As a third example, we use one of the quality criteria for the diabetes guideline from [6]. This property says that the guideline reaches its intention, namely, the level of sugar in the blood will be lowered for any patient group. This property is formalised as follows:

$$\mathcal{B}_{DM2},\, \mathcal{M},\, \mathcal{F}, \Box\, (\mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin})'') \wedge \\ \Box\ \mathrm{QI} = \mathrm{QI}'' \vdash \Diamond\, \neg\ \mathrm{Condition}(\mathit{hyperglycaemia})$$

where V'' denotes the value of the variable V in the next step. Our proof strategy consisted of splitting the patient group into groups which are cured by the same treatment, e.g., similar to the previous example, when the capacity is normal, then diet is sufficient.

Consider the example where the capacity of insulin in the B cells is nearly-exhausted. KIV derives from the failure axioms that:

$$\Box\, (\alpha_i \rightarrow \circ\, (\alpha_i \leftrightarrow \circ\, (\neg \alpha_i \wedge f_{\alpha_i})))$$

as we may assume that $\Box\, \neg$ Condition(*hyperglycaemia*), because the negation of this formula immediately proves the property. Furthermore, reasoning with the background knowledge, we can derive that proving $\Diamond\ (SU \wedge BG)$ is sufficient to prove this property, because for this patient group a treatment consisting of SU and BG is sufficient to conclude Condition(*normoglycaemia*). It is then easy to see how to complete this proof as the failure axioms specify that all the treatments will fail (after two steps), hence symbolic execution shows that eventually the third step will be activated.

7 Related Work

Failure has received little attention in formal theories of action. Of course, reasoning of actions had always taken into account the notion of failure, as illustrated by the logic of Rao & Georgeff, but it is assumed that failure can be added in a relatively straightforward manner. One notable example of where the notion of failure is part of both the syntax and semantics is the approach of Giunchiglia et al. [4]. Its primitive syntactic structure is:

$$\textbf{iffail}\ \alpha\ \textbf{then}\ \beta\ \textbf{else}\ \gamma$$

And from this, abbreviations are defined such that it allows one to reason conveniently about failures. The semantics is defined in terms of *behaviours* where it said that some behaviours have *failed*, while others are *successful*. Behaviours are defined technically in terms of linear models.

What this language lacks is the notion of time, as behaviours are simply considered a sequence of actions which either fail or do not fail. For medical management, this poses a problem, as failure may occur after a longer period of time. This means that the notion of failure needs a richer structure, so that it is possible to interact between time and failure.

Another important shortcoming for using this language in the context of medical management is that failures are considered properties of a *behaviour*. As said before, in medical management, actions are often performed in parallel, for example, the administering of a combination of drugs. In such cases, some drugs may fail to reach the required effects, while others may be successful. Hence, in the language decisions need to be made on, not only *if* a failure has occurred, but also *what* action has failed. We believe we have clearly demonstrated this in the previous section.

8 Discussion and Conclusions

In this paper, we have introduced semantics of failures in interval temporal logic inspired by the exception mechanism that can be found in many programming languages. The practical usefulness of our approach has been validated using medical guidelines by showing the verification of a fragment of diabetes mellitus type 2 which was formalised elegantly using this logic. However, we think that the results could be used in a much wider context. First, the reasoning about failures can have its applications in agent-based systems. Failures to perform tasks are an important aspect for decision making by agents, so having a reasonably rich language for modelling these failures seems justified. Second, in the context of program refinement, the process of (high-level) specifications to implementations of systems, exceptions are introduced at some point to model failure of components. The results of this paper makes it possible to abstract of concrete programming construct to describe how control of flow should change in case exceptions occur.

The logic that is proposed here can be seen as a three-valued logic, i.e., formulas are true, false, or failed. Some work has been done to link three-valued logics idea to temporal reasoning [7], which is based on Kleen's three-valued calculus that deals with 'unknown' values. This results in different logical properties compared to ITALF, e.g., unknown values propagate over a disjunctions, while failures do not.

Compared to [6], the verification of the investigated properties required significantly less effort. This is mainly due to the fact that in [6] the guideline was formalised in the guideline representation language Asbru [13], which yields overhead in complexity due to a complicated semantics. On the other hand, many of the steps that are required in ITALF were done manually, as it is not obvious to predict the correct next step in the proof. For example, it is important during verification to 'weaken' the irrelevant parts of the guideline, making the symbolic execution more efficient. Moreover, failure propositions on the sequent introduce additional complexity, as the human needs to remember the semantics of these propositions in order to apply the relevant axioms. These facts combined makes it interesting to consider more automatic techniques, such as automated theorem proving or model checking. This will a subject of further research.

A Proof of Theorem 1

A helpful semantic notion is faithfulness, which means that the failure propositions correspond exactly to the failure of the the action it has been introduced for.

Definition 5. *σ is called* faithful *iff for all $a \in \mathcal{A}$ and all i s.t. $0 \leq i \leq |\sigma|$ holds $\alpha_i(a) = \mathit{failed}$ iff $\pi_i(f_a) = \top$.*

In the following two lemmas, it is proven that the reduction is found with respect to those faithful models. In the first lemma, we show that Φ acts as $\mathsf{failmodel}$ on the syntactic level, which is then used to prove equivalence of formulas with its reduction.

Lemma 1. *For all faithful σ and φ:*

$$\mathit{failmodel}(\sigma) \models \varphi \text{ iff } \sigma \models \Phi(\varphi)$$

Proof. By induction on the structure of φ. First suppose $\varphi = a$: ($\Rightarrow$) suppose $\mathsf{failmodel}(\sigma) \models a$ then $\alpha_0(a) \neq \mathsf{failed}$. By faithfulness $\pi_i(f_a) = \bot$, thus $\sigma \models \neg f_a$. All steps can be reversed. The rest of the cases follow almost immediately, taking into account that if the model is faithful, so is every interval within this model, and vice versa.

Lemma 2. *For all faithful models σ it holds that $\sigma \models \varphi \leftrightarrow \mathit{reduce}(\varphi)$.*

Proof. By induction on the structure of φ. In this case, the only interested case is for $\varphi = \mathbf{fail}(\psi)$: ($\Rightarrow$) $\sigma \models \mathbf{fail}(\psi)$ iff $\sigma \not\models \psi$ and $\mathsf{failmodel}(\sigma) \not\models \psi$. By I.H. on the first part, it follows that $\sigma \not\models \mathsf{reduce}(\varphi)$. As σ is faithful, it follows that $\mathsf{failmodel}(\sigma)$ is faithful. Therefore $\mathsf{failmodel}(\sigma) \not\models \mathsf{reduce}(\varphi)$. Using Lemma 1, we get $\sigma \not\models \Phi(\mathsf{reduce}(\varphi))$. Therefore $\sigma \models \neg\mathsf{reduce}(\varphi) \wedge \neg\Phi(\mathsf{reduce}(\varphi))$. By definition, $\sigma \models \mathsf{reduce}(\mathbf{fail}(\varphi))$. All steps are valid in the other direction as well.

These results do not hold for any model, e.g., it is not for all models the case that $f_a \rightarrow \neg a$. A weak form of faithfulness can be encoded as an ITL formula, bounded by the number of actions in some formula. The fact it is bounded by actions in a formula is relevant, because we may have an infinite number of actions in the language, while each formula has a finite length in standard temporal logic.

Using Definition 4, we can then proof the main lemma, which characterises the relation between a formula and its reduction for any model.

Lemma 3. *$\models \varphi$ iff $\models \mathcal{I}(\varphi) \rightarrow \mathit{reduce}(\varphi)$*

Proof. Without loss of generality, this property can be reformulated as

$$\models \neg\varphi \text{ iff } \mathcal{I}(\varphi) \models \mathsf{reduce}(\neg\varphi)$$

as every formula can be stated as a negation and $\mathcal{I}(\neg\varphi) = \mathcal{I}(\varphi)$. Using the definition of reduce, and taking negation on both sides, rewrite this to:

$$\exists_\sigma \sigma \models \varphi \text{ iff } \exists_\sigma \sigma \models \mathcal{I}(\varphi) \wedge \mathsf{reduce}(\varphi)$$

($\Rightarrow$) Suppose there is some σ such that $\sigma \models \varphi$. Construct a σ' such that $\pi'_i(f_a) = \top$ iff $\alpha_i(a) = \mathsf{failed}$, for all $0 \leq i \leq |\sigma|$, actions a, and all fresh variables f_a introduced in the reduction. Let σ' be the same as σ in every other respect. As φ does not contain any variables f_a, it is clear that then $\sigma' \models \varphi$. As σ' is faithful (by construction), it then follows by Lemma 2 that $\sigma' \models \mathsf{reduce}(\varphi)$. Moreover, by construction, it follows that $\sigma' \models \mathcal{I}(\varphi)$.
($\Leftarrow$) Suppose for some σ, $\sigma \models \mathcal{I}(\varphi) \wedge \mathsf{reduce}(\varphi)$. Construct σ' such that for all i such that $0 \leq i \leq |\sigma|$, and all actions a:

- if $\pi_i(f_a) = \top$ then $\alpha'_i(a) = \mathsf{failed}$
- if $\pi_i(f_a) = \bot$ and $\alpha_i(a) = \mathsf{active}$ then $\alpha'_i(a) = \mathsf{active}$
- if $\pi_i(f_a) = \bot$ and $\alpha_i(a) \neq \mathsf{active}$ then $\alpha'_i(a) = \mathsf{inactive}$

In all other respects (length, valuation of atomic propositions), σ and σ' are the same. We then prove for all i and $a \in \mathit{actions}(\varphi)$:

$$\alpha_i(a) = \mathsf{active} \Leftrightarrow \alpha'_i(a) = \mathsf{active}$$

($\Rightarrow$) $\alpha_i(a) = \mathsf{active}$. Then, by the fact $\sigma \models \mathcal{I}(\varphi)$, we know that $\pi_i(f_a) = \bot$. Thus, by definition $\alpha'_i(a) = \mathsf{active}$. ($\Leftarrow$) Suppose $\alpha_i(a) \neq \mathsf{active}$. Then either $\alpha'_i(a) = \mathsf{failed}$ (if $\pi_i(f_a) = \top$) or $\alpha'_i(a) = \mathsf{inactive}$ (if $\pi_i(f_a) = \bot$). In any case, we conclude: $\alpha'_i(a) \neq \mathsf{active}$.
As $\mathsf{reduce}(\varphi)$ does not contain a **fail** operator, it cannot distinguish if an action is inactive or failed. Hence, it follows that $\sigma' \models \mathsf{reduce}(\varphi)$. It is easy to see that σ' is faithful, so by Lemma 2 it follows that $\sigma' \models \varphi$.

Now, Theorem 1 is proved in the following way. By Lemma 3, we know $\models \varphi$ iff $\models \mathcal{I}(\varphi) \rightarrow \mathsf{reduce}(\varphi)$. Observe that the right side does not contain the **fail** operator, hence it cannot distinguish between failures and inactivations. Therefore, $\models \mathcal{I}(\varphi) \rightarrow \mathsf{reduce}(\varphi)$ if all actions are interpreted as propositions. By doing this, $\mathcal{I}(\varphi) \rightarrow \mathsf{reduce}(\varphi)$ is also an ITL formula. Finally, note that the semantics of ITL and ITALF coincide for the language of ITL.

B Proof of Example 1

This appendix provides an outline of the proof performed in KIV. The first steps of the proof consists of simple manipulation of the formulas in order to put them in a comfortable form for presenting the proof. Note that we implicitly use axiom (10) of the background knowledge for making sure that normo-, hyper-, and hypoglycaemia are mutually exclusive. First, recall that the translated failure axiom for diet is:

$$\Box\,(\mathit{diet} \rightarrow \circ\,((\mathit{diet} \wedge \mathit{Condition}(\mathit{hyperglycaemia}) \leftrightarrow \circ\,\textbf{fail}\ \mathit{diet}))$$

Reduction of this to an ITL formula yields:

$$\Box\ (\mathit{diet} \rightarrow \circ\ ((\mathit{diet} \wedge \mathit{Condition}(\mathit{hyperglycaemia}) \leftrightarrow \circ\ (\neg \mathit{diet} \wedge f_{\mathit{diet}})))$$

which, by the use of Γ, can be written as:

$$\Box\ (\mathit{diet} \rightarrow (\circ\ (\mathit{diet} \wedge \mathit{Condition}(\mathit{hyperglycaemia})) \leftrightarrow \circ \circ f_{\mathit{diet}})) \qquad (3)$$

Second, from the background knowledge, we know that:

$$\Box\ (\mathit{diet} \wedge \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{normal} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia}))$$

which, together with the fact that $\Box\ \mathit{capacity}(\mathit{b\text{-}cells}, \mathit{insulin}) = \mathit{normal}$, it can be automatically derived that:

$$\Box\ (\mathit{diet} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})) \qquad (4)$$

Finally, note that the proof obligation can be presented as

$$\bullet\ \Box\ \mathrm{Condition}(\mathit{normoglycaemia}) \qquad (5)$$

By weakening all the uninteresting parts for proving the property, we finally end up with the main proof obligation:

$$\begin{array}{ll}
\Box\ (\mathit{diet} \rightarrow (\circ\ (\mathit{diet} \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \leftrightarrow \circ \circ f_{\mathit{diet}})), & \mathit{Eq}.(3) \\
\Box\ (\mathit{diet} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})), & \mathit{Eq}.(4) \\
(\Box\ \mathit{treatmentdiet} \wedge \neg\ \mathbf{last}); \circ f_{\mathit{diet}}, & \mathcal{M} \\
\Box\ (\mathit{treatmentdiet} \rightarrow \mathit{diet}), & \\
\vdash\ \bullet\ \Box\ \mathrm{Condition}(\mathit{normoglycaemia}) & \mathit{Eq}.(5)
\end{array}$$

Symbolically executing this sequent requires only one possible situation that needs to be proven:

$$\begin{array}{l}
\Box\ (\mathit{diet} \rightarrow (\circ\ (\mathit{diet} \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \leftrightarrow \circ \circ f_{\mathit{diet}})), \\
\Box\ (\mathit{diet} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})), \\
(\Box\ \mathit{treatmentdiet}); \circ f_{\mathit{diet}}, \\
\Box\ (\mathit{treatmentdiet} \rightarrow \mathit{diet}), \\
\mathrm{Condition}(\mathit{normoglycaemia}), \neg \circ f_{\mathit{diet}} \\
\vdash \Box\ \mathrm{Condition}(\mathit{normoglycaemia})
\end{array}$$

This sequent represents the situation where diet has been applied in the first step. From this it was derived that then the condition is normoglycaemia. Using this fact, the failure axiom is used to derive that $\neg \circ f_{\mathit{diet}}$, i.e., diet will not fail in the next step. The rest of the proof consists of the claim that this temporal situation will remain as it is. So we reason by induction that $\Box\ \mathrm{Condition}(\mathit{normoglycaemia})$. Abbreviate the sequent above as $\Gamma \vdash \Delta$: then the sequent is rewritten to:

$$\begin{array}{l}
\Box\ (\mathit{diet} \rightarrow (\circ\ (\mathit{diet} \wedge \mathrm{Condition}(\mathit{hyperglycaemia})) \leftrightarrow \circ \circ f_{\mathit{diet}})), \\
\Box\ (\mathit{diet} \rightarrow \circ\ \mathrm{Condition}(\mathit{normoglycaemia})), \\
(\Box\ \mathit{treatmentdiet}); \circ f_{\mathit{diet}}, \\
\Box\ (\mathit{treatmentdiet} \rightarrow \mathit{diet}), \\
\mathrm{Condition}(\mathit{normoglycaemia}), \neg \circ f_{\mathit{diet}}, \\
t = N, N = N'' + 1\ \mathbf{until}\ \neg\ \mathrm{Condition}(\mathit{normoglycaemia}), \mathbf{IND\text{-}HYP} \vdash
\end{array}$$

where **IND-HYP** $\triangleq t < N \rightarrow (\bigwedge \Gamma \rightarrow \bigvee \Delta)$, N a fresh dynamic variable and t a static variable. The remaining steps consists of symbolically executing this sequent, which ends up in the same sequent with $t = N - 1$. Then, the induction hypothesis can be applied, which finishes the proof.

References

1. J. Alves-Foss and F.S. Lam. Dynamic denotational semantics of Java. In *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*. Springer, 1999.
2. M. Balser. *Verifying Concurrent Systems with Symbolic Execution – Temporal Reasoning is Symbolic Execution with a Little Induction*. PhD thesis, University of Augsburg, Augsburg, Germany, 2005.
3. A. Cau and Moszkowski B. Using PVS for interval temporal logic proofs. part 1: The syntactic and semantic encoding. Technical report, SERCentre, De Montfort University, Leicester, 1996.
4. F. Giunchiglia, L. Spalazzi, and P. Traverso. Planning with failure. In *Artificial Intelligence Planning Systems*, pages 74–79, 1994.
5. K. Hindriks, F. de Boer, W. van der Hoek, and J-J. Ch. Meyer. Failure, monitoring and recovery in the agent language 3APL. In G. De Giacomo, editor, *AAAI 1998 Fall Symposium on Cognitive Robotics*, pages 68–75, 1998.
6. A.J. Hommersom, P.C. Groot, P.J.F. Lucas, M. Balser, and J. Schmitt. Verification of medical guidelines using background knowledge in task networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):832–846, 2007.
7. B. Konikowska. A three-valued linear temporal logic for reasoning about concurrency. Technical report, ICS PAS, Warsaw, 1998.
8. B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.
9. B. Moszkowski. The programming language Tempura. *Journal of Symbolic Computation*, 22(5/6):730–733, 1996.
10. D. von Oheimb and T. Nipkow. Machine-checking the Java specification: Proving type-safety. In *Formal Syntax and Semantics of Java*, volume 1523 of *LNCS*. Springer, 1999.
11. A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of KR'91*, pages 473–484. Morgan Kaufmann, 1991.
12. G.E.H.M. Rutten, S. Verhoeven, R.J. Heine, W.J.C. de Grauw, P.V.M. Cromme, and K. Reenders. NHG-standaard diabetes mellitus type 2 (eerste herziening). *Huisarts Wet*, 42:67–84, 1999.
13. Y. Shahar, S. Miksch, and P. Johnson. The Asgaard project: A task-specific framework for the application and critiquing of time-orientied clinical guidelines. *Artificial Intelligence in Medicine*, 14:29–51, 1998.
14. A.B. Tucker and R.E. Noonan. *Programming Languages – Principles and Paradigms*. McGraw-Hill, second edition, 2007.
15. R. Turner. *Logics for Artificial Intelligence*. Ellis Horwood, Chichester, 1985.