Formal Verification of CNL Health Recommendations*

Fahrurrozi Rahman and Juliana K. F. Bowles

School of Computer Science, North Haugh, St Andrews KY16 9SX, UK
{fr27,jkfb}@st-andrews.ac.uk

Abstract. Clinical texts, such as therapy algorithms, are often described in natural language and may include hidden inconsistencies, gaps and potential deadlocks. In this paper, we propose an approach to identify such problems with formal verification. From each sentence in the therapy algorithm we automatically generate a parse tree and derive case frames. From the case frames we construct a state-based representation (in our case a timed automaton) and use a model checker (here UPPAAL) to verify the model. Throughout the paper we use an example of the algorithm for blood glucose lowering therapy in adults with type 2 diabetes to illustrate our approach.

Keywords: Formal Verification, Controlled Natural Language, Timed automata, Health Recommendations

1 Introduction

Understanding system requirements in software development is important because requirements constitute the foundation for the next phases. There are many ways to specify system requirements, but the most common way is to express them in natural language (NL). Undoubtedly, NL is the easiest way for stakeholders to communicate and understand the requirements of a system. However, the possible ambiguity of NL may lead to various interpretations of requirements, making it also difficult to find (among others) requirement inconsistencies, incompleteness or underspecification. Research on how to find defects in requirements captured in NL is needed as early as possible in the development process. The ability to eliminate defects early can reduce the cost of a later correction or rework, and reduce time spent in implementation and testing phases. Furthermore, our increasing reliance on software systems across a wide range of application domains including critical systems puts further pressure on the development of high quality and dependable software systems. The use of formal verification techniques to automatically identify inconsistencies or incomplete requirements is thus natural but not common at the level of NL requirements.

Recent work by Carvalho [1] shows the use of natural language processing (NLP) and formal methods to automatically generate test cases from written

* This research is partially supported by EPSRC grant EP/M014290/1.

software requirements. By contrast, our present focus is on the use of NLP in a different setting, namely for capturing healthcare recommendations. Clinical texts share the difficulties encountered in software requirements, since these are often ungrammatical, use telegraphic phrases with limited context and often lack complete sentence structure [2]. In addition, the common use of acronyms and abbreviations in clinical texts introduces further ambiguity.

In this paper, we explore the use of formal methods for the validation and verification of clinical texts when used to describe therapy algorithms. Inspired by the approach done by Carvalho et al. [1] for software requirements, we explore more complex cases of controlled natural language (CNL) as used in therapy algorithms. From each sentence in the therapy algorithm we automatically generate a parse tree and derive case frames. From the case frames we automatically construct a state-based representation, in our case a timed automaton [3], and use the model checker UPPAAL [4] to verify the model. We write queries in UPPAAL's logic, a subset of TCTL, on the one side to evaluate the approach, and on the other side to check properties of clinical interest. The choice of timed automata and UPPAAL is motivated by the fact that clinical texts sometimes make reference to timed and periodic events, and we want the added flexibility of probabilistic extensions available in the wide range of tools that are available in the UPPAAL family.

Our approach makes it possible to detect gaps and case omissions, helps to further clarify treatment steps, and in the long term could be useful for patients that want to understand the therapy underlying their disease and the options they may have. Similarly for clinicians and health care providers. Throughout the paper, we use the algorithm for blood glucose lowering therapy in adults with type 2 diabetes from the National Institute for Health and Care Excellence (NICE) to illustrate our approach. NICE[1] provides guidance and advice in the UK to improve health and social care, and publishes clinical pathways for the treatment of chronic conditions including diabetes, hypertension, cancer, and so on. This work is part of a more general aim to integrate formal techniques, such as model checkers and constraint solvers, to detect and resolve inconsistencies in health recommendations and clinical guidelines. This paper adds a NLP dimension to our previous work on detecting inconsistencies in treatments for patients with multimorbidities [5], and the use of theorem provers and constraint solvers to detecting inconsistencies in requirements and scenarios of execution [6, 7].

This paper is structured as follows: we describe existing related work in Section 2, and the problem we are addressing with our proposed framework in Section 3. Section 4 presents the controlled natural language used to capture the sentences in a therapy algorithm, and Section 5 describes how case frames are generated. From the generated case frames we can further construct a timed automaton and use UPPAAL to verify it as described in Section 6. We conclude the paper with some discussion on future work in Section 7.

[1] More details on NICE available at www.nice.org.uk

2 Background and Related Work

When describing related work, we focus on work closer to our own and hence current uses of NLP in healthcare on the one hand, and links between NLP and formal verification on the other hand.

Advances in natural language processing (NLP) have been sought with applications in many fields, and in particular have also found increased interest in healthcare applications in recent years. Within healthcare there are two broad areas where the use of NLP has been explored, namely in processing free text occurrences in electronic health records (EHRs) and pathology reports. For instance, it is the free text in EHRs for mental health patients that contains key information on the evolution of the patient's symptoms and medications, and pathology reports are still to date essentially text-based. The general benefits of NLP in healthcare are clear. Demner-Fushman [8] reports the adaptation of principal NLP strategies to develop clinical decision support (CDS) systems that may help in decision making—e.g. monitoring of clinical events, processing radiology/pathology text-based reports, or processing a mixture of clinical notes — for health care providers as well as the public. Through the use of CDS systems, clinicians can enter patient data to get advice concerning recommendations or assessments from the knowledge base. Incorporating accessible electronic health records (EHRs) in the system makes it possible to automatically generate reminders or alerts when some conditions are met. However, EHRs are often recorded as free narrative text created by clinicians and care providers which adds considerable challenges to introducing automated tools and techniques for creating usable and useful CDS systems.

The information contained in textual form in EHRs and examination reports can in some cases enable new research and links between diseases to be detected. For example, a study by Shah et al. [9] showed how applying text mining to 16 million EHRs led to the understanding that the use of Proton Pump Inhibitors (PPI) may increase the risk of Myocardial Infarction (MI). Although their finding still needs additional investigations, it has demonstrated how data mining can be used to identify drug safety signals by learning on multiple clinical data sources.

The mining system in [9] was built based on previous work [10] which had shown that relationships between adverse drug reactions and (as a consequence of) drug-drug interactions were detectable with high accuracy using a large corpus. Usually patients with multiple long term conditions are subject to multiple treatments and there is a risk of undetected adverse reactions to combinations of prescribed medications for different conditions. As an example, in Scotland, over half of all people with chronic conditions have comorbidities, and a recent survey indicated that medicines are implicated in 5-17% of hospital admissions, of which half are considered preventable [11].

Imler et al. [12] conducted a study to improve the (cost) effectiveness of colorectal cancer (CRC) screening and surveillance. Their aim was to create and test a system at various institutions and identify the necessary components for producing high quality guideline surveillance recommendations through the use of NLP. From 42,569 documents, 750 documents were randomly selected and

split into training and test sets with ratio 1:2. From the 750 documents, five annotators would select 300 random documents each and then each document was annotated by two annotators. The annotating process would identify 19 features consisting of the category of colonoscopy (e.g. adenocarcinoma, advanced adenoma (AA), etc.), the location, and the counts of adenomas. From this more detailed analysis of the individual pathological findings and a variety of textual input, the accuracy of detecting CRC was 99.6%, of AA was 95%, of conventional adenoma was 94.6%, of advanced sessile serrated polyp (SSP) was 99.8% and of nonadvanced SSP was 99.2%.

In software engineering, Carvalho [1] has shown how to combine NLP and formal methods to automatically generate test cases from the written software requirements. The work proposes a framework to process and transform the requirement texts into a formalism called *data flow reactive system* (DFRS), which then is used to generate the test cases. In the DFRS the inputs and outputs of the system are modelled as signals, e.g. input signals from the sensors and output signals from the actuators. The system can also have timers to model time-based behaviour. As the input for the framework, the requirements are given in a controlled natural language (CNL) called SysReq-CNL, a subset of natural language tailored for generating unambiguous requirements. Every sentence is then parsed and structures called *case frames* are constructed from its parse tree following the case grammar theory [13]. In the case grammar theory, every sentence is analysed in terms of the thematic roles by each word or group of words, e.g. agent, patient, instrument, etc. For example, the thematic roles for "**the cat drinks milk**" are {**the cat/Agent**}, {**drinks/Action**}, and {**milk/Patient**}. Inside a case frame, the verb acts as the head of the frame and have several other thematic roles filled by the rest of the sentence elements. In Carvalho's approach, these case frames are transformed into the DFRS formalism and then translated into different target formal methods for verification to generate candidates for test cases. For example, the process algebra CSP [14] was used as the basis for generating test cases in [1]. Once the CSP specifications have been generated, the model checker FDR [15] can check the traces from the refinement property which then serve as test scenarios. In earlier work, DFRS had been transformed into other formalisms such as software cost reduction (SCR) [16], internal model representation (IMR) [17], and coloured Petri nets [18]. Independently of the formal approach used, the intention was always to generate test cases automatically.

Motivated to detect problems at an early stage of software development, Diamantopoulos [19] created a mechanism to automatically map requirements to formal representation through the use of ontologies and semantic role labelling. Comparatively to [1], their approach does not restrict the requirements to be written in a controlled language. From requirement sentences, several ontology concepts are inferred (i.e. project, requirement, actor, action, object, and property) using a model that has been trained from annotated software requirements. These ontology concepts can be used to trace the connection and relationship between them, and guide the translation of specifications to source code.

3 The Problem Domain and Framework

In this paper, we propose a framework to automatically generate logic-based statements from clinical texts—in this case a therapy algorithm— written in a controlled natural language, and enable formal verification of the algorithm. As an example of a therapy algorithm, we take a sample from the algorithm for blood glucose lowering therapy in adults with type 2 diabetes from NICE[2]. Fig. 1 shows a portion of the discussed algorithm. We note that HbA1c refers to *glycated haemoglobin* which can be used to determine the average blood sugar levels of a person over a period of weeks/months, and is a common measure for diabetes. A normal value is below 42 mmol/mol.

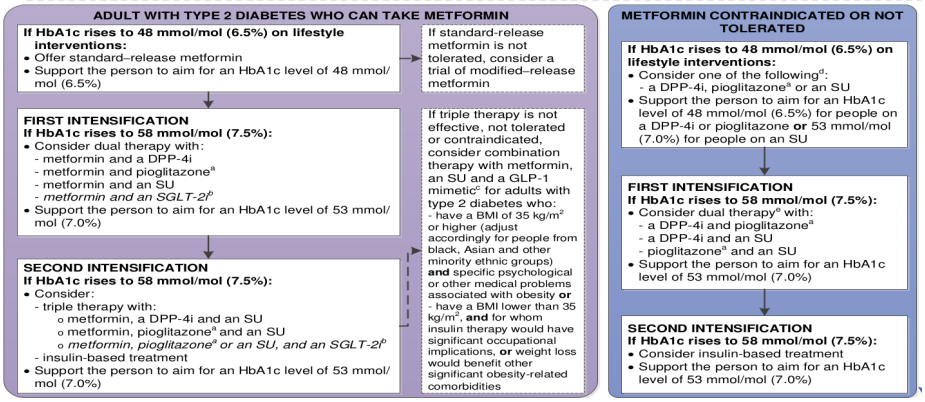

Fig. 1. A snippet of the blood glucose lowering therapy algorithm from NICE

To motivate our work, we want to check whether the therapy algorithm has inconsistencies (for example in the advised medications) and whether some cases (patient allergies or further long term conditions) are not considered and may leave a patient with no treatment options at all (or an option that may cause adverse drug reactions). In the future, we may also want to see if one therapy algorithm has conflicts with another algorithm for treating a different disease. Multimorbidity is increasing in the world, and hence a real concern in healthcare.

Our research is different from what we have discussed before because we do not deal with numerous patient records and clinical texts. Instead we currently only process a specific text related to handling a particular disease. Our approach also differs from [1] because we do not generate output in terms of steps for testing. We identify and verify discrepancies that may inherently be hidden inside therapy algorithms, and identify treatment options for patients with different circumstances.

[2] Full details are available at https://goo.gl/YDDtQY

To process the sentences in the therapy algorithm, we create a controlled natural language to standardise the structure of the sentences. Comparatively to [1], the sentence structure in a therapy algorithm is more complex because the value change is not always clearly visible (for instance *consider dual therapy with metformin and pioglitazone*) or the value changes in percentage (for instance, *aim for an HbA1c level of 53 mmol/mol (7.0%)*). Another difference is that the *agent* and the *patient* of the action are not explicitly stated. Here, we always assume that the agent is a doctor, or more generally a healthcare provider, and the patient is the patient under treatment. Furthermore, the conditions that need to be considered before giving a treatment may be nested and described in narrative form. The rules that we created are discussed in Section 4.

Our proposed framework is shown as a pipeline in Fig. 2 where the area inside the dashed border box indicates the system we have built. In our framework, light arrows show the automated steps whereas dark arrows imply manual processes. For example, we currently rewrite the therapy in CNL format and the logical queries for the model manually.

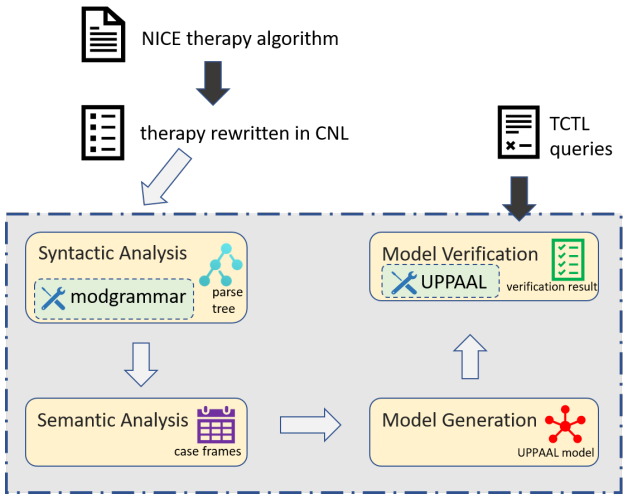

Fig. 2. The framework pipeline.

Before the therapy algorithm is processed, it is rewritten in CNL to conform with the grammar explained in the next section. Once all sentences are in agreement with the grammar, the Syntactic Analysis module will generate the parse trees with the help of an external library (indicated by the toolbox icon). The Semantic Analysis module will then construct the case frames by traversing the parse trees following some pre-defined rules explained in Section 5. The last two modules in the system deal with the UPPAAL model (a timed automaton). From the case frames, the Model Generation module generates the UPPAAL model as an XML file as discussed further in Section 6.1. Finally, the UPPAAL

model is verified against queries specified in the temporal logic TCTL using the UPPAAL model checker (cf. Section 6.2).

4 Controlled Natural Language for Therapy Algorithms

In a therapy algorithm, we consider that every sentence has the following form:

number if conditions, the doctor shall: actions

The number in the beginning of every sentence is needed so we know its order when we generate the model automatically, as steps within a therapy algorithm follow a particular order. For example in Fig 1, a first intensification can only happen after the patient has received a first treatment. Some considerations that we take when we transform all sentences into the standardised form are as follows: for every sentence, we add **the doctor** as the agent of the actions. In addition to explicitly showing who is responsible for a therapy, it is also needed to reduce the ambiguity from the missing information of who is responsible versus who receives the therapy.

Furthermore, if there is a list of choices (conditions or possible therapies), we transform it into

$$\{\textbf{choice}_1, \textbf{choice}_2, \cdots \textbf{choice}_n\}.$$

For example, the conditions in "*if triple therapy is not effective, not tolerated, or contraindicated*" and the advice "*consider therapy with a DPP-4i, pioglitazone, or an SU*" will be written as "*if triple therapy is* {*not effective, not tolerated, contraindicated*}" and "*consider therapy with* {*a DPP-4i, pioglitazone, an SU*}" respectively. This normalisation will ease us when we build the case frame from the parse tree.

Since our grammar is controlled and the lexicons are domain dependent (it is a medical therapy), we have specified in advance the vocabularies and their part-of-speech (POS) tags. Most of our lexical categories follow those by Carvalho [1] with some additional categories to simplify the parsing of the sentences and the case frame generation, such as **LBrace**, **RBrace**, **LPar**, **RPar**, and **Percent**. In addition to the lexical rules, we also follow Carvalho's [1] context free grammar (CFG) rules for our grammar with some modifications.

The start symbol of our grammar is **Advice** which is rewritten as

Number ConditionalClause Comma ActionClause

This means that the advice is in the form of actions guarded by conditions. Since we introduced a new rule to deal with a list of choices, we do not have the conjunctions of disjunctions anymore in our **ConditionalClause** as is visible by the absence of **Or** in our lexical rules. Our final grammar rules can be seen inside the *Grammar rule* box below.

After defining the lexical and grammar rules, the original sentences taken from the therapy algorithm are rewritten manually so they conform to the rules (as indicated by the black arrow in Fig. 2). For example, the original sentence after the first intensification for people who can take metformin in Fig. 1 is

written as "*2 if HbA1c level rises to 58 mmol/mol (7.5%), the doctor shall: consider dual therapy with {metformin and a DPP-4i, metformin and pioglitazone, metformin and an SU, metformin and an SGLT-2i}, support to aim for an HbA1c level of 53 mmol/mol (7.0%)*

To parse the sentences, we use an external library in Python called modgrammar[3] for building parsers using CFG definitions. In modgrammar, the rules are defined as Python classes making it possible to process the parse tree in terms of the underlying tree data structures. As the lexical rule is already predefined, we do not need to build a separate POS tagger to classify the tag for every word in the sentences because we can define the lexical rule in a similar manner as we defined the grammar rules.

Grammar rule

- **Advice** → Number? ConditionalClause Comma ActionClause
- **ConditionalClause** → Conj Condition
- **Condition** → NounPhrase VerbPhraseCondition
- **ActionClause** → NounPhrase VerbPhraseAction
- **VerbPhraseAction** → Shall Colon [VerbAction ToInfClause | VerbComplement | ChoiceAction ConditionalClause? sep=Comma]+
- **ChoiceAction** → LBrace PrepComplement [Comma VerbComplement]+ RBrace
- **VerbPhraseCondition** → VerbCondition Not? VerbComplement
- **ToInfClause** → To VBase VerbComplement
- **VerbComplement** → VariableState | ChoiceComplement | PrepComplement
- **PrepComplement** → [VariableState? Prep] VariableState | ChoiceComplement
- **ChoiceComplement** → LBrace VariableState [Comma VariableState]+ RBrace
- **PrepositionalPhrase** → Prep NounPhrase
- **VariableState** → AdjPhrase | NounPhrase
- **NounPhrase** → [Det? Adj* Noun PrepositionalPhrase*]+ [And NounPhrase+]*
- **VerbAction** → VBase
- **AdjPhrase** → [Adv?] Adj | VPart
- **VerbCondition** → VPre3 | VToBePre3
- **Noun** → Number | NSing | NPlur | Nmass

5 Case Frame Generation

After generating the parse tree, we construct automatically the case frames by traversing the parse tree. We follow eight thematic roles defined by Carvalho [1] with two additional roles: one for handling the nested condition in our sentences (CACT) and another one to mark the sequence of the sentence (NUM). The NUM role is useful in our case because the therapies are given in a sequence

[3] https://bitbucket.org/modgrammar/modgrammar

and we need to keep track of the order of their occurrence. For example, a value of 1 for NUM means the case frame is from the first sentence of the therapy, meanwhile 1.1 means the first alternative treatment for that first therapy. Apart from the NUM role, the thematic roles can be grouped into *action statements* and *conditional clause*. Here is a brief summary of each thematic role.

- Action statement thematic roles
  1. *Action* (ACT): the action to be performed if the conditions are met. It is taken from the **VBase** or the **ToInfClause** inside **VerbPhraseAction**. Here, we only allow *consider*, *review*, *offer*, *support*, and *aim* for this category.
  2. *Agent* (AGT): the actor of the action. It is taken from the **NounPhrase** node found inside **ActionClause**. There is only one agent, i.e. the doctor.
  3. *Patient* (PAT): the entity affected by the action. It is taken from the **VariableState** node found inside **VerbComplement**.
  4. *To Value* (TOV): the value given to the patient. It is taken from the **VariableState** inside **PrepositionalComplement** or **ChoiceComplement**.
  5. *Nested Condition Action* (CACT): the additional condition for the action. For every ACT found, a list of CACT is created whose value could be empty, i.e. an empty list. It is taken from the **ConditionalClause** found inside **VerbPhraseAction**.
- Conditional clause thematic roles
  1. *Condition Action* (CAC): the action for the condition. The values are taken from all **VerbCondition** inside **ConditionalClause**.
  2. *Condition Patient* (CPT): the entity related to the condition. Because in the therapy algorithm there is only one entity related to the condition(s), its value is taken from the **NounPhrase** found in **ConditionalClause**.
  3. *Condition To Value* (CTV): the new value of the condition patient. It is taken from the **VariableState** in **VerbComplement** or **ChoiceComplement**, or from the **Noun** in **PrepComplement**.
  4. *Condition Modifier* (CMD):the modifier for the condition. The value is taken from the first **Noun** in **PrepComplement** if there are more than one.

Fig. 3 shows an example of a case frame for the case when taking the first treatment in Fig. 1. The sentence is rewritten as "*1 if HbA1c level rises to 48 mmol/mol (6.5%) on lifestyle interventions, the doctor shall: offer standard-release metformin, support to aim for an HbA1c level of 48 mmol/mol (6.5%)* ."

6 Model Generation and Verification

In this paper, we generate state-based models automatically from an original description given in CNL, and verify the models with the model checker UPPAAL [4]. We first describe how models are generated.

NUM:	1		
CONDITION #1 - Main Verb (CAC):	rises		
CPT:	HbA1c level	CTV:	48 mmol/mol (6.5%)
CMD:	on lifestyle interventions		
ACTION - Main Verb (ACT):	offer		
AGT:	the doctor	TOV:	-
PAT:	standard-release metformin	CACT:	-
ACTION - Main Verb (ACT):	aim		
AGT:	the doctor	TOV:	48 mmol/mol (6.5%)
PAT:	HbA1c level		

Fig. 3. Example of a case frame

6.1 Model Generation

UPPAAL is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata. Timed automata (TA) [3] add the notion of time to standard automata (based on a finite set of states and labelled transitions between them) through a set of variables called *clocks*. Clocks are special variables which can be inspected or reset but not assigned a value. A time unit represents a second, minute or month, depending on what is a sensible unit for the model (note that there is no time in our therapy algorithm snippet). A constraint can be placed on *locations* (the term used for states in a TA) to denote a *location invariant* (to indicate for instance how long the automaton can remain in the location) and on transitions where it acts as a guard.

In UPPAAL we can create several instances of processes with the same behaviour. The behaviour is captured in a so-called template (a TA), and one or more instances of that template can be declared for runtime verification. In our example of a therapy algorithm for diabetes we need a template for the behaviour captured in the algorithm for treating diabetes, and an additional template to simulate a random change of the value of HbA1c in the blood.

The process of generating the diabetes automaton is done automatically by traversing the case frames for each sentence following the sequence number in the NUM role. This is shown as the Model Generation module in Fig. 2. We identified all variables from CPT and PAT roles and their values from the CTV and TOV respectively. If the value contains a number, the type of the variable is **int**, otherwise it is **bool**. For example, finding HbA1c can have value of 48, 53, and 58 made it an **int** variable. To model how HbA1c can change its value, we created a simple template that always increments the value of HbA1c. We assume $40 \leq HbA1c \leq 60$.

The locations are built sequentially from the first sentence to the last. Whenever the verb **rises** is found in CAC role, its CTV value is used as the upper limit for the current location's invariant and as the lower limit for the transition's guard to the next location. Updates in transitions are done by setting the boolean variable from PAT to **true**.

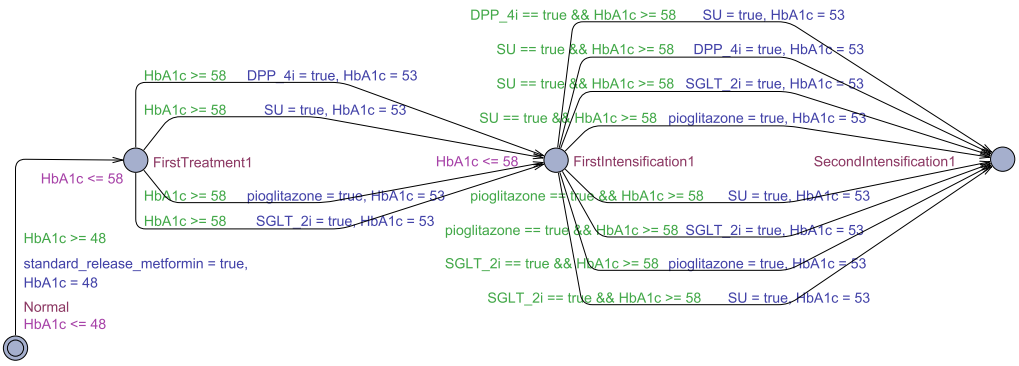

Fig. 4. Generated model for adults with type 2 diabetes that tolerate metformin

Fig. 4 shows the model generated automatically. In this model, we only show the therapy path where standard-release metformin is tolerated, the first intensification and the second intensification. The locations have a location invariant on the value of HbA1c to indicate that if HbA1c rises above a certain value (for instance above 48 in the location Normal) then the location must be left and a treatment given. The different transitions between intensifications show the different options available and are dependant on the first treatment received (the taken medications are captured by a boolean variable with the same name and value true). The first alternative path where modified-release metformin is used will create a mirror of locations and transitions which only differs on the guard for the first transition. The full model is omitted here.

When investigating the generated model, we noticed that there is no transition that takes the system (in this case a potential patient) back to the initial state Normal. As it may be more realistic to assume that other factors (such as changing life style habits and diet) can have an effect and recovery is possible, we want to refine the model to take this into account. This means that the model should cover a situation where the patient's condition turns back to normal after a time under treatment. On the other side, this may not be a very frequent outcome and ideally we would like to quantify the likelihood of such a recovery to happen as opposed to a deterioration of the condition as given in the model of Fig. 4. This can be done with Probabilistic Timed Automata [20] which extend TA with probabilistic transitions. How to quantify such transitions is outside the scope of the present paper but should be informed by analysis of large datasets of records for patients with diabetes.

We modified the model generation process to create some branch points after a treatment is taken. A branch is created whenever we find the verb **aim** denoting the situation when the doctor tries to stabilise the HbA1c level after giving a treatment. At present, we give a value of 20 and 80 as the weight to go back to a normal state and to the next (deteriorated) state respectively. Fig. 5 shows the graphical model with branch points. To keep the model more readable, we

currently only show going back to a normal state, but further possible transition branches include returning to any other previous treatment state with different weights.

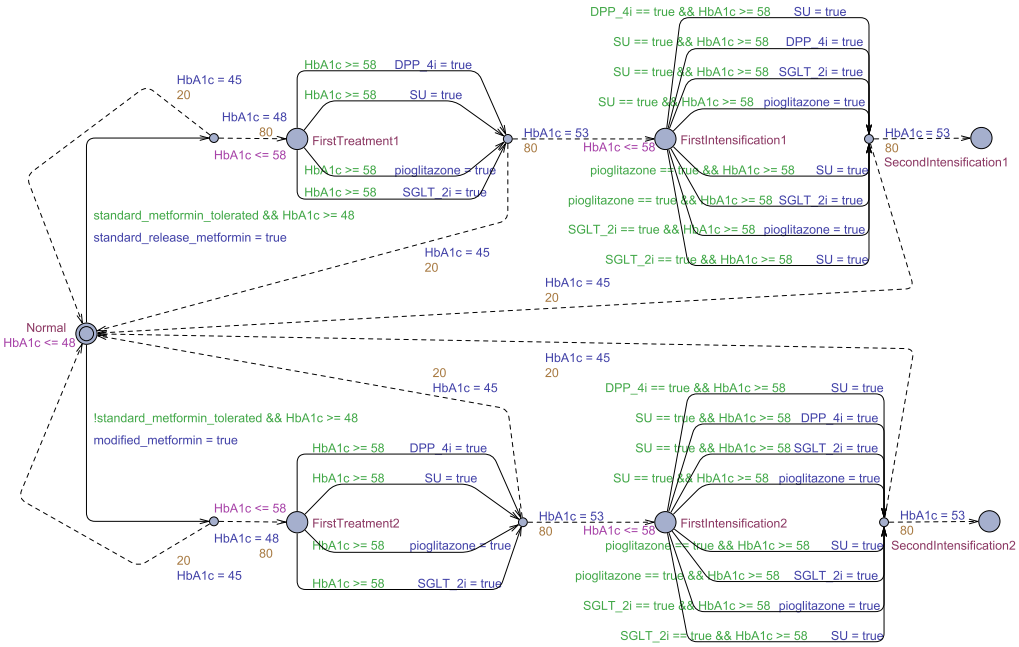

Fig. 5. Model with Branch Points

We note that the actual model generation is done by generating an xml file which can then be visualised in the tool as in Fig. 5 (after minor adjustments to take into account visual placement of the elements). The xml file contains all the information of locations, location invariants, transitions and variables, but can also given directly on the command line to the model checker for verification.

6.2 Model Verification

We have verified our model against some properties as shown in Table 1. The properties selected are used primarily to evaluate the approach. UPPAAL takes properties written in a restricted form of the temporal logic TCTL. The properties were verified against the model in Fig. 5 after an initial analysis of the original model and its refinement to include recovery. The property

```
A[] !diab.SecondIntensification1 && !diab.SecondIntensification2
```

illustrates the situation where a second intensification of the therapy has been reached (the property itself only holds iff second intensification is never reached,

and the fact that it is not satisfied will result in a trace that shows a second intensification of the therapy being reached). Another similar property, `A[] !deadlock`, is not satisfied because the model at present and as generated from the therapy algorithm contains no further steps after second intensification and hence deadlocks at that point. This suggests that further discussions with clinicians are required to understand available options from that point and what is realistic.

Table 1: Verification Result

UPPAAL queries	Verification Result	Remark
`A[] !deadlock`	not satisfied	There is a path that leads to a deadlock.
`E<> !diab.FirstIntensification1 && !diab.FirstIntensification2`	satisfied	There exists a path where first intensification is never reached.
`E<> !diab.SecondIntensification1 && !diab.SecondIntensification2`	satisfied	There exists a path where second intensification is never reached.
`A[] !diab.SecondIntensification1 && !diab.SecondIntensification2`	not satisfied	There is a path that reaches a second intensification.
`diab.FirstIntensification2 --> diab.SecondIntensification2`	not satisfied	There is a path in which a second intensification will never be reached from a first intensification.

From the model in Fig. 5, we can also see that there is a possibility of never reaching a first intensification shown by verifying the formulae:

```
E<> !diab.FirstIntensification1 && !diab.FirstIntensification2
```

The same is also true for second intensification. Again, these situations became possible because we added a scenario where the patient's condition improves after getting some treatment. The last property (which uses temporal implication)

```
diab.FirstIntensification2 --> diab.SecondIntensification2
```

also shows the possibility of never evolving to a second intensification after reaching the first intensification.

7 Conclusion

We have presented an approach to analyse therapy algorithms published by NICE automatically. Therapy algorithms contain instructions which we treat as CNL statements and from which we ultimately build a state-based representation, in our case a timed automaton, that can be analysed by a model checker

such as UPPAAL. A particular problem that we wanted to investigate was how to detect inconsistencies and gaps in the treatment options. In particular, it was noted that the algorithm did not consider a possible recovery in the different treatment stages that a patient with type 2 diabetes may go through nor what happens after a second intensification (which creates a deadlock eventually).

In future work, and in collaboration with a general practitioner (GP), we will analyse EHRs for patients with diabetes in Scotland, in order to detect cases or treatments that are not currently captured in the guidelines, and extend the algorithm accordingly. More broadly, we also want to see if one algorithm may have conflicts with another from a different disease (cf. [5]).

Our long term goal is to build a framework of therapy models which can: (1) be used as a reference to give advice to clinicians and patients based on their current situation and future treatment options, and (2) compare the therapy algorithm with actual practice. For the latter, note that adding real data from EHRs will enable us to obtain models for current practice and allow us to contrast these with NICE guidelines and recommendations. In particular, the choice of timed automata and UPPAAL allows us to add further therapy algorithms as different templates which can then be composed together and verified directly. In addition, our models can be easily extended to incorporate probabilities on branched transitions to reflect real data from EHRs. Another possible direction to guarantee the scalability of our approach, consists of exploring links to constraint solvers such as Z3 (for cases of bounded model checking). We have used constraint solvers considerably in our work, including our recent work in [5, 7].

References

1. Carvalho, Gustavo H. P. d.: NAT2TEST: Generating Test Cases from Natural Language Requirements based on CSP. PhD Thesis (2016)
2. Townsend, H.: Natural Language Processing and Clinical Outcomes: The Promise and Progress of NLP for Improved Care. Journal of AHIMA 84, no.2 (March 2013): 44-45.
3. Alur, R., Dill, David L.: A theory of timed automata. Theoretical Computer Science (1994), 126: 183-235.
4. Behrmann, G., David A., Larsen, K.: A Tutorial on Uppaal. Formal Methods for the Design of Real-Time Systems (2004), 3185: 200-236.
5. Kovalov, A., Bowles, J.: Avoiding medication conflicts for patients with multimorbidities. In: 12th International Conference on Integrated Formal Methods. pp. 376–392. LNCS 9681, Springer (2016)
6. Bowles, J., Bordbar, B., Alwanain, M.: Weaving true-concurrent aspects using constraint solvers. In: Application of Concurrency to System Design (ACSD 2016). IEEE Computer Society Press (June 2016)
7. Bowles, J.K.F., Caminati, M.B.: Mind the gap: addressing behavioural inconsistencies with formal methods. In: 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE Computer Society (2016)
8. Demner-Fushman, D., Chapman, Wendy W., McDonald, Clement J.: What can natural language processing do for clinical decision support? Journal of Biomedical Informatics, volume 42, no. 5 (2009): 760-772

9. Shah, Nigam H., LePendu, P., Bauer-Mehren, A., Ghebremariam, Yohannes T., Iyer, Srinivasan V., Marcus, J., Nead, Kevin T., Cooke, John P., Leeper, Nicholas J.: Proton Pump Inhibitor Usage and the Risk of Myocardial Infarction in the General Population. (2015) PLoS ONE 10(6): e0124653. doi:10.1371/journal.pone.0124653
10. LePendu, P., Iyer, Srinivasan V., Bauer-Mehren, A., Harpaz, R., Mortensen, J., Podchiyska, T., Ferris, Todd A., Shah, Nigam H.: Pharmacovigilance Using Clinical Notes. Clinical Pharmacology & Therapeutics (2013), 93: 547555. doi:10.1038/clpt.2013.47
11. *Polypharmacy Guidance (2nd Edition).* Scottish Government Model of Care Polypharmacy Working Group, 2015.
12. Imler, T. D., Morea, J., Kahi, C., Cardwell, J., Johnson, C. S., Xu, H., Imperiale, T. F. (2015). Multi-center colonoscopy quality measurement utilizing natural language processing. American Journal of Gastroenterology, 110(4), 543-552. DOI: 10.1038/ajg.2015.51
13. Fillmore, Charles J.: The Case for Case. Universals in Linguistic Theory (1968)
14. Hoare, Charles Antony Richard.: Communicating Sequential Processes. The origin of concurrent programming (1978).
15. Gibson-Robinson, T., Armstrong, P., Boulgakov, Alexandre., Roscoe, A.W.: FDR3 — A Modern Refinement Checker for CSP. Tools and Algorithms for the Construction and Analysis of Systems (2014).
16. Carvalho, G., Falcão, D., Barros, F., Sampaio, A., Mota, A., Motta, L., Blackburn, M.: NAT2TEST$_{SCR}$: test case generation from natural language requirements based on SCR specifications. Science of Computer Programming (2014)
17. Carvalho, G., Barros, F., Lapschies, F., Schulze, U., Peleska, J.: Model-Based Testing from Controlled Natural Language Requirements. Formal Techniques for Safety-Critical Systems: Second International Workshop (2013).
18. Silva, B., Carvalho, G., Sampaio, A.: Test Case Generation from Natural Language Requirements Using CPN Simulation. Formal Methods: Foundations and Applications: 18th Brazilian Symposium (2016).
19. Diamantopoulos, T., Roth, M., Symeonidis, A., Klein, E.: Software requirements as an application domain for natural language processing. *Language Resources and Evaluation*, 51(2):495–524, 2017.
20. Norman, G., Parker, D., and Sproston, J.: Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164-190, 2013.