Automated Verification of Care Pathways Using Constraint Programming

Renan Pereira de Figueiredo, João Bosco Ferreira Filho, Flávio R. C. Sousa, Philip Weber, and Ian Litchfield

***Abstract*—Bad construction of modeled care pathways can lead to satisfiability problems during the pathway execution. These problems can ultimately result in medical errors and need to be checked as formally as possible. Therefore, this study proposes a set of algorithms using a free open-source library dedicated to constraint programming allied with a DSL to encode and verify care pathways, checking four possible problems: states in deadlock, non-determinism, inaccessible steps and transitions with logically equivalent guard conditions. We then test our algorithms in 84 real care pathways used both in hospitals and surgeries. Using our algorithms, we were able to find 200 problems taking less than 1 second to complete the verification on most pathways.**

***Index Terms*—Clinical pathway, constraint programming, data-dependent transition system, DSL, satisfiability problems.**

I. Introduction

The development of new technological resources for the health sector is reshaping medical practice and expanding the range of possible treatments. The behavioral variability, the exponential growth of scientific information, and the advent of evidence-based medicine [1], [2], have driven health care stakeholders to search for standardization in treatments, provided by well-defined care pathways. Among their many benefits, care pathways reduce possible errors of diagnosis and improve the quality of service [3], [4], [2].

One way of implementing care pathways is to mirror the structure of industrial and business processes. These process-based guidelines assist health professionals with decision making when treating patients, using best practices, facilitating communication and sequencing the activities of multidisciplinary health care teams [5]. They are increasingly being used, in the United Kingdom, for example, they were introduced in the early 1990s and are used in combination with national guidance and local National Health Service policy to provide appropriate care in a local context [6], [7], and currently, most health areas in the UK already have a care pathway approach [8]. In the USA, care pathways were used in more than 80% of hospitals in the late 1990s [9]. A study by Lopes and Ramires [10] on the presence of clinical pathways in Central Europe, states that in some fields of medicine such as Oncology and Palliative Care, pathways have become the standard model of care. Medical errors can cost a patient their life and millions in damages to hospitals. This immense responsibility in health care processes warrants IT investment for process automation.

On the one hand, a care pathway can be seen as a business process [11], [12], [13], having a structure of a data-dependent transition system where there are some **states** as steps connected by a set of **transitions** (also called sequences). Some of these transitions have guard conditions whose logical operations define which one will be the next step in the execution. On the other hand, domain experts (doctors, nurses, etc.) are not aware of some structures present in the syntax of business process languages such as gateways in BPMN (Business Process Model and Notation) [14]; medical professionals in charge of defining pathways just end up drawing boxes and arrows between them with a condition labeling these arrows. Considering the importance of the correctness of these pathways and the fact that specialists will not follow a formal language to draw them, in this work, we develop a set of algorithms to check for four basic problems in pathways that follow the basic structure of boxes and arrows with labels. We follow the model-driven engineering paradigm [15], using a DSL (Domain Specific Language) [16] with a Constraint Programming (CP) Solver [17] to analyze all transitions and their guard conditions that could present satisfiability problems. The four possible logical problems we search for are:

- **Deadlock.** Finding deadlocks will prevent the pathway execution system getting blocked at any time while treating a patient;
- **Non-determinism.** Finding non-determinism will ensure that there is always one, and only one path to be followed given the current state, avoiding ambiguity during the treatment;
- **Inaccessible Steps.** Finding inaccessible steps prevents states that will never run in a pathway;
- **Equivalent Transitions.** Finding transitions with logically equivalent guard conditions helps to find redundancy in pathways, avoiding rework;

We take a set of 84 real care pathways to test the algorithms modeled with specific characteristics by a group of stakeholders. The main contribution of this work is a tool that can help specialists in the modeling of safer pathways, mainly with regards to the transitions guard conditions.

R. P. Figueiredo and J. B. Ferreira Filho are with the Group of Computer Networking, Software Engineering and Systems (GREat), Departament of Computer, Universidade Federal do Ceará, Fortaleza, Ceará, BR. (email: renanpdef@gmail.com; bosco@dc.ufc.br)

F. R. C. Sousa is with Teleinformatics Engineering Department (DETI), Universidade Federal do Ceará, Fortaleza, Ceará, BR. (email: flaviosousa@ufc.br)

P. Weber is with Aston University, Birmingham, Birmingham, UK. (email: p.weber1@aston.ac.uk)

I. Litchfield is with University of Birmingham Institute of Applied Health Research, Birmingham, West Midlands, UK. (email: I.Litchfield@bham.ac.uk)

Manuscript received X X, X; revised X X, X.

The remainder of this paper is divided as follows. Section II presents the related work. Section III gives background and motivation about the structure of the care pathways, the possible problems that can be found and the computational tools we used to address them, which are the model-driven engineering and the constraint programming paradigm. Section IV contains the actual algorithms to find the problems. Section V evaluates the algorithms, Section VI concludes the paper, and Section VII gives thanks.

II. Related Work

There are papers in the literature focusing on error detection in business process models [18], [19], [20], [21], [22], [23], [24]. Many of these papers try to find errors in the data-flow, such as missing, redundant or unused data, as can be seen in [19], [20] and [21]. Also, there has been an effort to find structural errors as deadlocks and other problems in workflows [22], [23], [24]. In [22] Kherbouche, Ahmad and Basson propose an approach to automate the checking of some structural errors in BPMN process models [14] based on model checking. BPMN is a standard for process modeling that provides support for modelling control flow, data flow, and resource allocation. They map the BPMN process model to Kripke structures [25] that provide semantics and allow checking the validity of a specific property holds or not. In our work, we bring this verification capacity closer to the clinical pathway domain by abstracting structures that are not familiar to stakeholders from the medical domain, such as gateways from the BPMN notation. Our algorithms are ready to check any process model that uses only boxes as steps and arrows as transitions between them.

There are also similar studies that analyze clinical pathways. In [6], the authors present a method for detecting execution paths in two Business Process models that violate a set of constraints. They extended BPMN to become more appropriate for modelling care pathways, and after, transformed this extended BPMN to CPN (Couloured Petri Nets) [26], aiming to simplify the analysis since CPN is normally used when the process behavior is heavily influenced by the data to model concurrent systems by analyzing their properties. In the mentioned work, CPN models are enhanced with logical constraints to represent potential conflicts. Another similar approach is the integrated framework developed in [27] which detects and resolves conflicts in the pathways used for patients with multimorbidities. They also use BPMN to model the guidelines that is transformed into an intermediate formal model for a better analysis, then using a constraint solver Z3 [28] to check the satisfiability of a set of assertions expressed in first-order logic, together with the theorem prover Isabelle [29], a proof assistant which provides a framework to accommodate logical systems to compute the validity of logical deductions, to combine treatment plans and check the correctness of the approach. Both papers have different objectives than the one presented in this work. The problem addressed in Weber's paper is to identify conflicts between clinical pathways when they are followed concurrently in treating patients with multiple morbidities. And Bowles' aims at finding a combination of formalisms able to capture pathways, highlighting the problems using an event-based approach.

TABLE I
Features used in related works.

Features	Our Work	Kherbouche	Weber	Bowles
Automated verification	X	X	X	X
Specific to medical field	X	O	X	X
DSL	X	O	O	O
CP Solver	X	O	O	X
Simplistic and powerful notation	X	X	O	O
Straightforward verification	X	O	O	O

Our approach focuses on the practice, by reading existing pathways as they are (drawing of boxes and arrows with labelled transitions) and calculating the existence of 4 problems, without requiring a previous formalization of the pathways in, for example, an event based system. Being closer to the practice, we are able to validate our approach within a real set of pathways and actually find modelling errors.

Table I shows the main features present in our work and which ones are also present in the works of Kherbouche, Weber and Bowles. We use the letter "X" for applied features in the work, and the letter "O" when it is not applied. According to Table I, DSL and Straightforward Verification are the biggest differentials of our work because they are the two features of what is not used by any of the other works. The use of a DSL allows us to make a straightforward verification of care pathways, which makes the error checking process easier and faster for modelers.

Therefore, the main contribution of this work is to present a set of algorithms to verify the bad construction of modeled care pathways without using well-known approaches that have, for example, scalability limitations, such as Petri Nets [30]. Besides, it contributes to the use, auditing and management of care pathways in a more practical way, as we do not consider verbose process languages to describe pathways, but rather a simplistic and powerful notation containing only states, transitions and guards. The verification of these pathways become straightforward; instead of having to translate pathways from practice to computer science modeling languages and then finding the formalism to map to, we simply get the already existing pathways and give as input to our approach.

III. Material and Methods

A. Care Pathway Structure

In this work, we will consider the most basic way of defining pathways: boxes and arrows between them; this is the way that most doctors, specialists use to define their workflows, see for example the book of knowledge of care pathways in the UK[1]. Considering this, we take care pathways as data-dependent transition systems [31] as can be seen in Figure 1. They are directed graphs where nodes represent *states* that describe some information about a system at a certain moment of its behavior, and edges model transitions that specify how the system can evolve from one state to another.

[1]https://pathways.nice.org.uk/

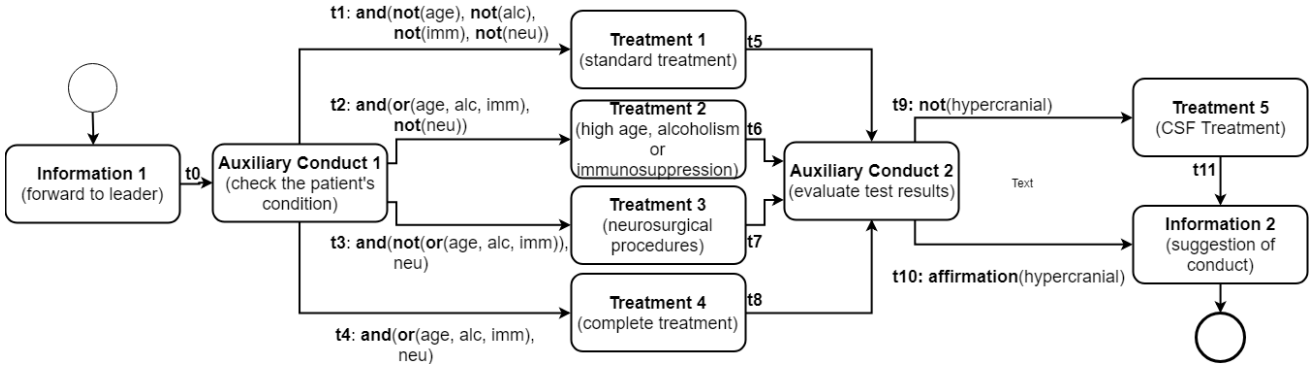

Fig. 1. Meningitis care pathway.

Figure 1 is a representation of a real care pathway where states are represented by rectangular shapes, the conditional transitions by the labeled arrows and non-conditional transition by empty arrows. The thin circle and thick circle represent the initial and final states respectively. In the studied pathways, there is only one initial state, but it may have more than one final state. A state evolves according to a transition relation $\rightarrow$, that can be conditional, as transition $t1$, or not, as transition $t5$. Atomic propositions express simple known facts about the states of the system under consideration, as "$age = 1$", or "$alc = 0$". The labeling function L relates a set $L(s) \in 2^{AP}$ of atomic propositions to any state s (AP is the set of atomic propositions). $L(s)$ intuitively stands for exactly those atomic propositions $a \in AP$ which are satisfied by state s. Conditional transitions have an operation as the label, the guard condition g, a boolean expression composed by a set of operands O related by operators that could be logical, relational, arithmetic or a unary operator, i.g., $and(or(age, alc, imm), neu)$. An operand is a variable or a constant from an operation that can have a weight, an integer value that can be assigned as the value of the operand. The operand can be Numeric, YesOrNo (Boolean), Choice, or another operation. A Numeric operand is an integer constant or a variable whose domain is the set of integers $\mathbb{Z}$. A YesOrNo operand is a Boolean variable with only two values as its domain that could be 0 and 1, or 0 and the operand weight. The Choice operand is an integer variable composed by components called Option. Each Option has a weight that is added to the variable's value when chosen by the doctor. It is important to emphasize that, in the pathways analyzed, once a variable assumes a value during the pathway execution, it maintains that same value throughout the entire execution.

Intuitively, the behavior of the pathway represented in Figure 1 can be described as follows. The pathway starts with *Information 1*, which forwards the case to the team leader, which then in *Auxiliary Conduct 1* checks the patient's condition. At this moment the doctor has to analyze 4 variables: *age*, *alc*, *imm* and *neu*. The first three variables represent aggravating conditions for the patient as, respectively, high age, alcoholism and immunosuppression, the last variable "*neu*" represents the need for neurosurgical treatment. Then the pathway evolves to one of the four treatments: *Treatment 1*, the standard treatment, *Treatment 2* if there are some aggravating conditions, *Treatment 3* if neurosurgical procedures are necessary and there is no aggravating condition or *Treatment 4* if there are some aggravating conditions and the need for neurosurgical procedures. Each of the four treatments is connected to another *Auxiliary Conduct* by a non-conditional transition. In the *Auxiliary Conduct 2* the doctor evaluates the test results and checks the presence of cranial hypertension in the patient, represented by the variable *hypercranial*. If the patient does not have cranial hypertension, the doctor submits it to a CSF (Cerebral Spinal Fluid) treatment, the *Treatment 5*, and then goes to the final state *Information 2* with suggestions of conduct. Otherwise, the pathway evolves directly into the final state with another suggestion. *Information 2* has more than one suggestion of conduct that might be used according to the patient's condition.

Therefore, this pathway presents nine states, $|S| = 9$, with the initial state s_0 = *Information 1*. We have six operations as guard condition of some transitions in all pathway with the set of operands O= {*age, alc, imm, neu, hypercranial*}. As the set of atomic proposition we have $AP = \{age = 0, age = 1, alc = 0, alc = 1, imm = 0, imm = 1, neu = 0, neu = 1, hypercranial = 0, hypercranial = 1\}$. Through the labeling function we can see which atomic propositions must be satisfied to reach a certain state, i.g. $L(Treatment\ 3) = \{\{age{=}0, alc{=}0, imm{=}0, neu{=}1\}\}$. It is important to point out that subsequent states depend on the occurrence of previous states.

The pathways used as objects of study in this work are used by a single doctor treating a single patient per time. Therefore, in this scenario, incorporating parallelism is a complicating factor, as all operations are sequenced. We can imagine situations in which concurrency may be important (e.g., orchestration of surgery by multiple professionals), however, this was not the reality in the health organizations we had as partners. We acknowledge this as a possible feature and we consider it as a possibility of future work.

B. Possible Problems in a Clinical Pathway

We can define the four possible issues addressed in our work as follows. We assume that: S is the set of states in a pathway; the semantics of the model only allows a patient to be in one state at a time; and that the problems refer to a given state and not to the whole pathway.

- **Deadlock:** being T_{out} a set of the output transitions of a state $s \in S$ and g_t the guard condition of $t \in T_{out}$. $\forall t \in T_{out}\ g_t = 0 \rightarrow s$ *is in deadlock*.
- **Non-Determinism:** being T_{out} a set of the output transitions of a state $s \in S$ and g_t the guard condition of $t \in T_{out}$. $\exists t_i \wedge \exists t_j \in T_{out} \mid t_i \neq t_j,\ g_{t_i} = g_{t_j} = 1 \rightarrow s$ *has a non-determinism problem*. Therefore it is not possible to resolve the next step.
- **Inaccessible Steps:** being T_{in} a set of the input transitions of a state $s \in S$ and g_t the guard condition of $t \in T_{in}$. $\nexists t \in T_{in} \mid g_t = 1 \rightarrow s$ *is inaccessible step*.
- **Logically Equivalent Sequences:** being T_{out} a set of the output transitions of a state $s \in S$ and g_t the guard condition of $t \in T_{out}$. $\exists t_i \wedge \exists t_j \in T_{out} \mid t_i \neq t_j,\ g_{t_i} \equiv g_{t_j} \rightarrow t_i \equiv t_j$.

To ease the understanding of these problems, we have created 4 examples of possible problems using the data-dependent transition system notation presented in the book by Baier and Katoen [31], which can be accessed at additional material [2].

Figure 1 is an example of a well-designed pathway; there is no problem of non-determinism. If we analyze the labeling functions of every state, we would check that there is no equivalent set of propositions in the labeling functions, that is, there is no set of propositions in which more than one state can be reached at the same time.

To check for a deadlock, we need to make some observations. In the example of Figure 1, to know if a deadlock can occur in a state such as the *Auxiliary Conduct 1*, we verify if we can reach at least one of the possible 4 next states (Treatment 1...4). This verification is done by iterating through the possible values of the 4 Boolean variables: age, alc, imm and neu; the possibilities are therefore equal to 2^4. For this particular example, we can check that any of the expressions guarding the treatments can be satisfied for at least one set of value attribution. To reach *Auxiliary Conduct 2*, it is only necessary one of the four treatments (1 to 4) is executed, as *Auxiliary Conduct 2* is not guarded by any expression, implying that t5, t6, t7 and t8 are resolved to true. In a third moment, to reach the final state or *Treatment 5*, we have a new variable, *hypercranial*, then, considering the possibility of the previous state being reached, we have now 32 sets of possible values for the variables. We can also notice that transitions *t9* and *t10* contemplate all these possibilities. Therefore, there is no set of propositions that is not satisfied by at least one of the guard conditions, that is, there are no deadlocks.

The inaccessible step problem takes a different approach. It is possible to detect an inaccessible step problem through a contradiction in the guard condition causing the labeling function to return an empty set, for example: making a small change in the guard condition of transition *t3* to *and(not(or(age, alc, imm, neu)), neu)* we get a new labeling function of *Treatment 3* which returns an empty set, $L(Treatment3) = \emptyset$, that is, there is no set of propositions satisfying the guarding condition of *t3*, since it is not possible to have the variable *neu* to be false and true at the same time. Meningitis pathway has no inaccessible step problem.

Now, we look at the problem of logically equivalent transitions, which generates a situation of non-determinism. In the pathway of Figure 1 there is also no state with this problem. For a non-determinism to occur, we would have to have the following situation. Suppose g_{t1} = *and(not(age), not(alc), not(imm), neu)* and we have g_{t3} = *and(not(or(age, alc, imm)), neu)*, then $g_{t1} \equiv g_{t3}$ and $L(Treatment1) = L(Treatment3) = \{\{age{=}0, alc{=}0, imm{=}0, neu{=}1\}\}$. Whenever the guarding condition from one of this transitions is satisfied, the other one is satisfied as well. Therefore we have $t1 \equiv t3$, causing the state *Auxiliary Conduct 1* to have a non-determinism problem.

Although the problem of logically equivalent transitions is a problem of non-determinism, it is interesting to analyze it separately because it is a more aggravating situation. Detecting logically equivalent transitions separately is important because it is the worst case of a non-determinism problem. Non-determinism may or may not occur, it will depend on the values that the variables in the guard will assume; however if there are equivalent transitions, there will always be non-determinism, regardless of the values assumed by variables. Detecting this situation separately allows for earlier correction by modelers.

The 4 specific problems were motivated by a high-level manual analysis of existing pathways at partner hospitals. However, there may exist other possible problems, such as the existence of cycles. Cycles may sometimes cause pathways to get stuck and execute continuously without ever reaching a final state. Detecting cycles can be challenging and should be considered as the next step for our solution. Although we recognize it may occur, we have not found any cycles in the pathways used in this work during our manual inspection.

C. Model-Driven Engineering and Constraint Programming

Model-Driven Engineering (MDE) is a software development methodology that creates domain models, combining domain-specific modeling languages (DSML) with transformation engines and generators. These models help to understand complex systems and obtain results through a low level of abstraction improving the maintenance and evolution of the system [32]. For this work the care pathways were modeled using a DSL [16] developed by a multidisciplinary team of medical professionals and computer scientists with the Eclipse Modeling Framework to generalize the clinical pathway structure, modeling it with all its elements and flow conditions to ease the analysis process; its abstract syntax (i.e., metamodel) is close to a business process and can be found at github[3].

Constraint Programming represents a real-world problem in terms of decision variables and constraints, and find an assignment to all the variables that satisfies the constraints. In this work, we use the CP Solver named Choco Solver [33]. It is a free and open-source satisfaction problem solver, easy to use, extend and integrate to other software. The key component from Choco is Choco model. It should be the first instruction,

[2]https://docs.google.com/document/d/10kkocFouHZsSTihf85vxR53-UAIb12iDMyyHzUzhJsA/edit?usp=sharing

[3]https://github.com/CarePathwayModeler/pathwayMetamodel.git

before any other modeling instructions, as it is needed to declare variables and constraints. IntVar and BoolVar are types of variables. An IntVar is an integer variable whose domain is a set of integers, that could be bound through an interval of two integer numbers or enumerated. A BoolVar is a boolean variable, a specific IntVar that take their value in $[0, 1]$. The Constraints are restrictions over variables as a logic formula that must be satisfied to get a feasible solution. A solution to a problem is the assignment of values to variables verifying all the constraints.

IV. Algorithms

The goal of this study is to develop a service to find possible logical problems in the structure of a care pathway. Considering this, this study aims at answering the following research questions: How to find,

- deadlock in a clinical pathway?
- non-determinism in a clinical pathway?
- inaccessible steps in the pathway?
- logically equivalent sequences in the pathway?

Based on these research questions, we developed a set of algorithms to verify the logical problems in clinical pathways with the use of MDE integrated with Choco Solver. There are four main functions, each one aiming at verifying one of the four problems. The subtopics below explain the behavior of these algorithms as well as the use of Choco to perform this verification. The algorithms are implemented and available at github[4].

The 4 algorithms were developed to follow the set of care pathways states with their respective transitions to create a set of constraints. Thus, we build the satisfiability problems to be solved by Choco Solver. According to Choco's developers and our understanding after analysing choco's code, choco is commonly used to solve NP-Hard problems and has exponential time complexity. This makes all our algorithms to have exponential complexity at the worst case.

A. Logical Structure

Before understanding the algorithms we should understand how each state from a pathway is represented logically. Figure 2 and Table II show how the pathway states are logically structured for the set of algorithms. Notice that states, transitions and guard conditions are created as a Choco BoolVar. Each transition is true if and only if its guarding condition is satisfied. A state is true (or accessible) if and only if its input transition and the previous state were also satisfied. We can also see that the problems in a state are represented as a set of constraints.

The algorithms to detect deadlock, non-determinism, and logically equivalent sequences analyze each pathway state and verify if the state presents irregularities, considering only the set of output transitions from the state, checking their guard conditions. That is, each state and its set of output transitions are analyzed individually. The inaccessible step detection algorithm performs a depth-first search in the pathway, constructing multiple paths with the initial state as its first element.

[4]https://github.com/carepathways/Pathway-Verification.git

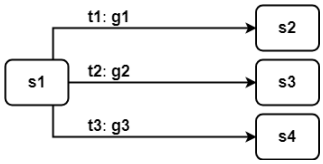

Fig. 2. Simple pathway representation.

TABLE II
Logical Representation of Figure 2

Pathway Objects	Logical Representation
State s	BoolVar $s \mid s = [0, 1]$
Transition t	BoolVar $t \mid t = [0, 1]$
Guard Condition g	BoolVar $g \mid g = [0, 1]$
t_i	$t_i \iff g_i$
s_2	$s_2 \iff t_1 \wedge s_1$
s_3	$s_3 \iff t_2 \wedge s_1$
s_4	$s_4 \iff t_3 \wedge s_1$
s_1 in Deadlock	$t_1 + t_2 + t_3 = 0$
s_1 with Non-Determinism	$t_1 + t_2 + t_3 \geq 2$
s_1 with Equivalent Sequences	$\forall t_1 t_2, t_1 + t_2 \neq 1$ $\vee\ \forall t_1 t_3, t_1 + t_3 \neq 1$ $\vee\ \forall t_2 t_3, t_2 + t_3 \neq 1$

B. Transform Transitions to BoolVars

All four algorithms use a method to transform a list of transitions into a list of BoolVar named SequenceListToBoolVarList. It gets as input a set of transitions and a Choco model m, runs over every transition in the set, transforms each of them to BoolVar, inserts them in a set of BoolVar and returns it.

To accomplish this transformation, we should first transform each operand into an integer variable (IntVar) or Boolean variable (BoolVar) of Choco, and store them in two sets, V_i and V_b, for IntVars and BolVars respectively. We check every operand in an operation, verifying if they are Numeric, Boolean or Choice. We create an IntVar with an enumerated domain for each numeric operand with no default value. If the numeric operand already has a declared value, it is no longer a variable, so the IntVar is created as a constant with the same value of the operand as its domain. To transform the Boolean operands it is necessary to check the operator. If the operator is arithmetic, we describe the operand as an IntVar enumerated with 0 and the operand's weight. Otherwise, it is described as a BoolVar whose domain is 0 and 1. For Choice operands, we create an enumerated IntVar whose domain includes 0, the set of possible weights attributed to Choice Options, and the possible combinations of their sums.

After transforming the operands into IntVar and BoolVar, we have to transform the operations themselves. We analyze the existing operators in operations to transform a transition in a BoolVar reified with constraints created using the BoolVars and IntVars from V_b and V_i sets. Therefore we create the corresponding constraints according to the operators using a method named ARITHM from Choco model, a method to create

constraints. E.g., representing the *AND* logic between two BoolVars we have to create a new BoolVar b that only be true if the sum of those two BoolVars is 2 since a BoolVar can only be 0 or 1, and we call it a Boolvar reified with a *AND* logic constraint. We use the same logic for other logical operators. A relational operation works similarly. An *EQUALITY* operation will only be a BoolVar reified as equality of two IntVars. The unary operation may be an *AFFIRMATION* of a BoolVar A, setting a new BoolVar B to 1 if A is 1 too, or a *NEGATION*, setting B to 1 if the older BoolVar A is 0. For arithmetic operations we use IntVar methods to calculate these operations and create another IntVar with all possible results as its domain, i.e. for a *MULTIPLICATION* operation of two IntVars is generated a new IntVar whose domain is the set of all possible results of this multiplication. Every constraint can be posted on the Choco model m or reified to a new BoolVar that can be a representation of a transition constrained by its operation.

C. Finding Deadlock, Non-Determinism, and Logical Equivalence

Algorithm 1 is suited for finding a situation in which a state is in deadlock. First of all, in line 2, the algorithm iterates over the size of the map M_s with the pathway states as key and their respective output transitions as values. In lines 3 to 5 the Choco model is instantiated, we get a set of transitions T_{out} from the set of map values and we use the function SEQUENCELISTTOBOOLVARLIST to convert T_{out} to a set of BoolVar B. We then iterate over all BoolVar b in B, setting b to 0 (false) and post it as the constraint on the Choco model. In line 8 we get the Solver, an object obtained from Choco model in charge of alternating constraint-propagation with search and learning to compute solutions. In line 9 and 10 we get the origin state of the transitions by the keys set from map M_s and add it to map M_r as a key with the set of possible deadlock cases get by function FINDALLSOLUTIONS, a method from the Solver used to attempt to find all solutions of the declared satisfaction problem. Finally, we return M_r with the states and their respective deadlock cases.

Algorithm 1 findDeadlockSolutions (M_s)

Input: A map of states with the sets of their respective output transitions M_s.
Output: A map M_r of states with the deadlock case for each state.

1: $M_r \leftarrow \emptyset$
2: **for** $i \leftarrow 0$ **to** SIZE(M_s) **do**
3: $m \leftarrow$ MODEL()
4: $T_{out} \leftarrow$ VALUES(M_s)[i]
5: $B \leftarrow$ SEQUENCELISTTOBOOLVARLIST(m, T_{out})
6: **for all** $b \in B$ **do**
7: Post ARITHM($b = 0$)
8: $solver \leftarrow$ GETSOLVER(m)
9: $s \leftarrow$ KEYS(M_s)[i]
10: $M_r \leftarrow M_r \cup \{s,$ FINDALLSOLUTIONS($solver$)$\}$
11: **return** M_r

The algorithm used to find states with non-determinism problem works in the same way as deadlock algorithm. It starts by iterating over the size of the map M_s, instantiates the Choco model, gets a set of transitions T_{out} from the map values and transform it to a set of BoolVar B. The main difference is in the constraints created where we take two BoolVars from B, b_0 and b_1, and set both to true as constraint on Choco model (ARITHM($b_0 + b_1 = 2$)). Then we add it to a map M_r with the state and a case that present a non-determinism problem.

To find logically equivalent transitions we verify the operation of two or more transitions with the same output state. Initially, we get a set of transition T_{out} from values of the map M_s and convert it to a set of BoolVar B. We iterate over all the BoolVars in B, posting as a constraint the sum of two different BoolVars in B is 1 (ARITHM($b_1 + b_2 = 1$)), that is, if one of them is true the other one has to be false. If there are no solutions with this constraint it means that these two BoolVars are logically equivalent, so the transitions represented by these BoolVar are added in a set of logically equivalent transitions. In the end, this equivalent transitions set is added to M_r with their respective origin state as key.

D. Finding Inaccessible Steps

Algorithm 2 uses a different approach to find inaccessible steps. The method gets the set S of all steps in the pathway and tries to find all the accessible steps, only to prune the set S, removing all these accessible steps to get a set of unfeasible steps. First, the sets are initialized: a set S_a to store accessible steps, V for the steps visited, a set named $Stack$ in which the search is performed forming a path, and the set T of transitions that connect the states from $Stack$. All these sets start with the initial step, except for T. The iteration then cycles through the stack.

At lines 4 to 6 a Choco model is instantiated, we take the state s_0 from the top of the stack and get the next output transition to be verified with GETNEXTSEQUENCE(s_0, V). This method adds states not yet visited in V and returns an unverified transition or null if all output transitions from s_0 have already been verified. Getting null as t_{out}, we remove s_0 from the $Stack$ and the last transition from T. Otherwise, we add t_{out} in T, transform T into a set of BoolVars B, and post in Choco Model the constraints to check if the next step is accessible by the path built in T (all transitions labels of T have to be satisfiable). Then we add the next step s_1 to S_a if there is at least a solution with these constraints and add it to the stack if it has at least one output transition, otherwise, remove the last transition from T. The same is done if there is no solution to the set of constraints because if the state is not accessible, it is not necessary to check the subsequent states. In the end, we use the GETINACESSIBLEELEMENTS function to remove all accessible steps from S and return the result.

V. EVALUATION

The algorithms seen in Section IV were run on a personal computer with a quad-core processor Intel Core i7-2670QM and 8GB RAM. They were tested with a set of 84 real care pathways already used in over 211,000 patient care services since October 2017. In the tests performed, we found several cases of deadlocks (103) and non-determinism (58), and some cases of inaccessible steps (39). Table III shows 7 pathways from a total of 84 analyzed in this study; they are ordered by time spent to search for the four problems. Table III also

Algorithm 2 findInaccessibleSteps (S)

Input: A set S of the pathway states.
Output: A set of inaccessible states S_i.

```
1: S_a, V, Stack ← getInitialElement(S)
2: T ← ∅
3: while Stack is not empty do
4:     m ← model()
5:     s_0 ← lastElement(Stack)
6:     t_out ← getNextSequence(s_0, V)
7:     if t_out = null then
8:         Remove s_0 from stack and the last transition from T
9:     else
10:        T ← T ∪ t_out
11:        B ← sequenceListToBoolVarList(m, T)
12:        for all b ∈ B do
13:            Post arithm(b = 1)
14:        solver = getSolver(m)
15:        if findSolution(solver))} ≠ null then
16:            s_1 ← imputStep(t_out)
17:            S_a ← S_a ∪ s_1
18:            Add s_1 in stack
19:            if outputSequences(s_1) ≠ null then
20:                Add s_1 in stack
21:            else
22:                Remove the last transition from T
23:        else
24:            Remove the last transition from T
25: S_i ← getInaccessibleElements(S, S_a)
26: return S_i
```

TABLE III
THE PATHWAYS AND THE FOUND PROBLEMS.

Pathway	S	Dl	ND	IS	ET	RT
Stroke	8	2	0	0	0	792293.07
Pneumonia Influenza	26	0	0	0	0	6801.2
Abdominal Pain	57	2	2	4	0	47.79
Diarrhea	34	0	0	0	0	23.16
Low Back Pain	46	7	9	3	0	19.18
Dermatological Disorders	30	3	3	0	0	18.95
Headache	14	1	1	0	0	15.41

S = Number of States; Dl = Deadlock
ND = Non-Determinism; IS = Inaccessible Steps
ET = Equivalent Transitions; RT = Average runtime in millisecond

lists the number of states and the number of deadlocks, non-determinism, inaccessible steps and equivalent transitions. See the full table[5].

The last column of Table III shows the average runtime. These time values were achieved by running the four main algorithms for each pathway 12 times. We then excluded the highest and the lowest execution time values and calculate the average of the 10 remaining values. We use time values to define the complexity of the pathways. For the sake of comparison, when the search takes less than 1 millisecond we named it as a low complexity pathway; more than 1 millisecond but less than 1 second we named it as a medium complexity pathway; as high complexity if it takes more than 1 second to check for the problems; and as of very high complexity if more than 1 minute is needed. Table IV shows the pathways grouped by the level of complexity. The runtime is less than one second in most cases (97,6%), except for two pathways (Stroke and Pneumonia Influenza) that have more complex operations and require more time to run.

To exemplify the problems found, Figure 3 shows a fragment of the Low Back Pain pathway that presents a problem of

[5]https://docs.google.com/spreadsheets/d/1zJrsYDqnD2tbOKErvq9CYZOaT5ahnThw4UwQ-lE8UPc/edit?usp=sharing

TABLE IV
PATHWAY COMPLEXITY LEVEL TABLE.

Complexity Level	Pathways Nº	Percentage (%)
Low	16	19.3
Medium	66	78.3
High	1	1.2
Very High	1	1.2

non-determinism. The *Auxiliary Conduct* has three transitions, the transitions *t24* and *t25* have different operands, *stone-suspicion* and *stone-present* respectively. If both operands assume the value *true* (*stone-suspicion* = 1 and *stone-present* = 1), then both transactions are valid (evaluate to true) and then we have a non-determinism problem, as it would not be possible to decide which step will be executed next. We can notice that in this case there is no deadlock problem, since transition *t26* is always true when the other two transitions are false, *stone-suspicion* + *stone-present* = 0.

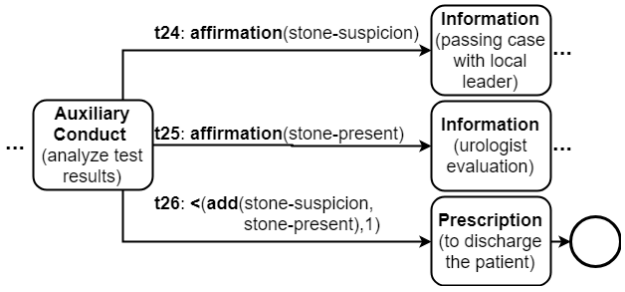

Fig. 3. A fragment of low back pain care pathway.

Another example that presents problems is the *exposed fracture* pathway in Figure 4. It is a small pathway with only four steps but presenting Deadlock and Non-Determinism problems. It has two different operands, *fract-degree12* and *fract-degree34*, and both can assume true or false values at the same time.

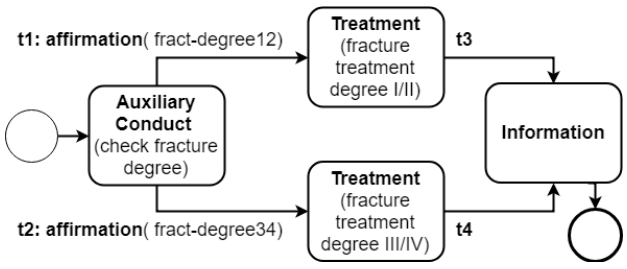

Fig. 4. Exposed fracture care pathway

All the inaccessible steps found are states isolated in the pathway, not having a set of transitions that connect them to the initial state. There were no cases of equivalent transitions in any of the tested pathways. Although it may be interesting to verify the existence of equivalent transitions, it is a very specific case of non-determinism with little possibility of an incident, as it is a very straightforward verification that can be done at modeling time.

We notice that some of the problems are due to poor definition of variables. For instance, in Figure 4, instead of defining

2 boolean variables (*fract-degree12* and *fract-degree34*), the modeler could have created one single numeric variable for expressing the degree of the fracture, in this way, we would avoid the problem of assigning true for both variables.

All the results obtained with the execution of the algorithms in the set of tested pathways were analyzed manually and individually. Also, some pathways were redesigned and still run in real scenarios. Some models were built with deliberate errors to test the algorithms. However, it is hard to ensure that there are no more errors from the execution of the algorithms, as well as no false positives. An induction proof would be needed to guarantee that given any model as input, our algorithms would find all the errors. We are aware that our approach would benefit from such a theoretical proof and will investigate further in future work.

VI. Conclusion and Future Work

This work offers a solution for finding possible logical problems in the structure of a clinical pathway. It contributes to the audit and management of these pathways, helping in the process of correction. The algorithms were tested in real pathways and we were able to find problems and report them accordingly. We believe that our approach may prevent serious mistakes that can ultimately affect patient's treatment.

As future work, we would like to investigate theoretical proofs in a full theoretical analysis for the correctness and completeness of our algorithms and also improve their performance. We may also address other possible pathway problems, such as the existence of cycles and data inconsistency caused by concurrent operations. Finally, we have to embed these algorithms into the modeling practice, checking in real-time if a pathway under construction is correct. This allows a quick fix by the modeler before the pathway goes into use in the practice.

VII. Acknowledgment

We would like to thank the Brazilian National Scientific and Technological Development Council (CNPq) for partly funding this research via scholarship. We would also like to thank IntMed Software company for providing the care pathways we analyzed in this work, and doctors from Hapvida's hospitals for clarifying questions regarding the pathways used in the hospitals.

References

[1] B. Djulbegovic and G. H. Guyatt, "Progress in evidence-based medicine: a quarter century on," *The Lancet*, vol. 390, no. 10092, pp. 415–423, Jul 2017.

[2] L. Cosgrove, A. F. Shaughnessy, and T. Shaneyfelt, "When is a guideline not a guideline? the devil is in the details," *BMJ Evidence-Based Medicine*, vol. 23, no. 1, pp. 33–36, 2018. [Online]. Available: https://ebm.bmj.com/content/23/1/33

[3] H. S. G. Caballero, A. Corvò, P. M. Dixit, and M. A. Westenberg, "Visual analytics for evaluating clinical pathways," in *2017 IEEE Workshop on Visual Analytics in Healthcare (VAHC)*, Oct 2017, pp. 39–46.

[4] J. Shi, Q. Su, and Z. Zhao, "Critical factors for the effectiveness of clinical pathway in improving care outcomes," in *2008 International Conference on Service Systems and Service Management*, June 2008, pp. 1–6.

[5] K. Vanhaecht, M. Panella, R. van Zelm, and W. Sermeus, "An overview on the history and concept of care pathways as complex interventions," *International Journal of Care Pathways*, vol. 14, no. 3, pp. 117–123, 2010. [Online]. Available: https://doi.org/10.1258/jicp.2010.010019

[6] P. Weber, J. B. F. Filho, B. Bordbar, M. Lee, I. Litchfield, and R. Backman, "Automated conflict detection between medical care pathways," *Journal of Software: Evolution and Process*, vol. 0, no. 0, p. e1898, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1898

[7] K. Zander, "Integrated care pathways: Eleven international trends," *Journal of integrated Care Pathways*, vol. 6, no. 3, pp. 101–107, 2002. [Online]. Available: https://doi.org/10.1177/147322970200600302

[8] CPA, "The effectiveness of care pathways in health and social care," *Centre for Policy on Ageing - Rapid Reviews*, May 2014.

[9] K. Vanhaecht, M. Panella, R. van Zelm, and W. Sermeus, "What about care pathways?" 01 2011.

[10] J. Lopez and B. Ramirez, "Cost savings through clinical care pathways in austria, poland, and the slovak republic," *Clinical Social Work and Health Intervention*, vol. 8, no. 2, pp. 38–43, 2017. [Online]. Available: https://www.clinicalsocialwork.eu/wp-content/uploads/2017/02/08-Lopez.pdf

[11] J. Poelmans, G. Dedene, G. Verheyden, H. Van Der Mussele, S. Viaene, and E. Peters, "Combining business process and data discovery techniques for analyzing and improving integrated care pathways," in *Industrial Conference on Data Mining*. Springer, 2010, pp. 505–517.

[12] G. Schrijvers, A. van Hoorn, and N. Huiskes, "The care pathway: concepts and theories: an introduction," *International journal of integrated care*, vol. 12, no. Special Edition Integrated Care Pathways, 2012.

[13] P. Gooch and A. Roudsari, "Computerization of workflows, guidelines, and care pathways: a review of implementation challenges for process-oriented health information systems," *Journal of the American Medical Informatics Association*, vol. 18, no. 6, pp. 738–748, 2011.

[14] OMG®, *Business Process Model and Notation (BPMN)*, Object Management Group®, January 2011. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/PDF

[15] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb 2006.

[16] S. Anonsen, "Uml modeling languages and applications," N. Jardim Nunes, B. Selic, A. Rodrigues da Silva, and A. Toval Alvarez, Eds. Berlin, Heidelberg: Springer-Verlag, 2005, ch. Experiences in Modeling for a Domain Specific Language, pp. 187–197. [Online]. Available: http://dl.acm.org/citation.cfm?id=2206963.2206984

[17] M. Milano, "Twenty years of constraint programming (cp) research," *Constraints*, vol. 23, no. 2, pp. 155–157, Apr 2018.

[18] M. Hammer, *What is Business Process Management?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 3–16. [Online]. Available: https://doi.org/10.1007/978-3-642-45100-3_1

[19] A. Elajeli Rgibi, S. Zhen Yao, and J. Jun Xu, "Dataflow errors detection in business process model," *Applied Mechanics and Materials*, vol. 130-134, 10 2011.

[20] S. v. Stackelberg, S. Putze, J. Mülle, and K. Böhm, "Detecting data-flow errors in bpmn 2.0," *Open Journal of Information Systems*, vol. 1, no. 2, pp. 1–19, 2014.

[21] M. I. Kabbaj, A. Bétari, Z. Bakkoury, and A. Rharbi, "Towards an active help on detecting data flow errors in business process models." *IJCSA*, vol. 12, no. 1, pp. 16–25, 2015.

[22] O. M. Kherbouche, A. Ahmad, and H. Basson, "Detecting structural errors in bpmn process models," in *2012 15th International Multitopic Conference (INMIC)*, Dec 2012, pp. 425–431.

[23] A. Awad and F. Puhlmann, "Structural detection of deadlocks in business process models," in *Business Information Systems*, W. Abramowicz and D. Fensel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 239–250.

[24] T. Maruta, S. Onoda, Y. Ikkai, T. Kobayashi, and N. Komoda, "A deadlock detection algorithm for business processes workflow models," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, vol. 1, Oct 1998, pp. 611–616 vol.1.

[25] M. Browne, E. Clarke, and O. Grümberg, "Characterizing finite kripke structures in propositional temporal logic," *Theoretical Computer Science*, vol. 59, no. 1, pp. 115 – 131, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397588900989

[26] P. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st ed. Springer-Verlag Berlin Heidelberg, 2009.

[27] J. Bowles, M. B. Caminati, and S. Cha, "An integrated framework for verifying multiple care pathways," in *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, Sept 2017, pp. 1–8.

[28] D. R. Cok, D. Déharbe, and T. Weber, "The 2014 SMT competition," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 207–242, 2014. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/122

[29] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, 1st ed. Springer-Verlag Berlin Heidelberg, 2002.

[30] G. Denaro and M. Pezze, "Petri nets and software engineering," in *Advanced Course on Petri Nets*. Springer, 2003, pp. 439–466.

[31] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[32] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, Sept 2003.

[33] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Documentation*, TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017. [Online]. Available: http://www.choco-solver.org