A Flexible Approach for Finding Optimal Paths with Minimal Conflicts

Juliana K.F. Bowles(✉) and Marco B. Caminati

School of Computer Science,
University of St Andrews, St Andrews KY16 9SX, UK
{jkfb,mbc8}@st-andrews.ac.uk

Abstract. Complex systems are usually modelled through a combination of structural and behavioural models, where separate behavioural models make it easier to design and understand partial behaviour. When partial models are combined, we need to guarantee that they are consistent, and several automated techniques have been developed to check this. We argue that in some cases it is impossible to guarantee total consistency, and instead we want to find execution paths across such models with minimal conflicts with respect to a certain metric of interest. We present an efficient and scalable solution to find optimal paths through a combination of the theorem prover Isabelle with the constraint solver Z3. Our approach has been inspired by a healthcare problem, namely how to detect conflicts between medications taken by patients with multiple chronic conditions, and how to find preferable alternatives automatically.

1 Introduction

In complex systems design, it is common to model components separately in order to facilitate the understanding and analysis of their behaviour. Nonetheless, modelling the complete behaviour of a component is hard [25], and often sets of possible scenarios of execution are captured instead. A scenario describes a particular situation that may involve a component and how it behaves and interacts with other components. In practice, UML's sequence diagrams are commonly used to model scenarios [20]. From such individual scenarios, we then need to be able to derive the complete behaviour of a component. The same ideas apply if we model (partial) business processes within an organisation, for instance using BPMN [19]. In either case, we need a means to compose models (scenarios or processes), and when this cannot be done, detect and resolve inconsistencies.

Composing systems manually can only be done for small systems. As a result, in recent years, various methods for automated model composition have been introduced [1,4–6,11,13,22,23,26–28]. Most of these methods introduce algorithms to produce a composite model from partial specifications and assume a formal underlying semantics [11]. In our recent work [4,6], we have used constraint solvers (Alloy [9] and Z3 [17] respectively) for automatically constructing

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

the composed model. This involves generating all constraints associated to the models, and using an automated solver to find a solution (the composed model) for the conjunction of all constraints. Using Alloy for model composition (usually only for structural models), is an active area of research [23,28], but the use of Z3 is a novelty of [6]. In [6] we did not exploit Z3's arithmetic capabilities which we have done more recently in [7]. Most existing approaches can detect inconsistencies, but fail to provide a means to resolve them.

We argue that in some cases it is impossible to guarantee total consistency between scenarios of execution. Instead we want to find execution paths across such models with minimal conflicts with respect to a certain metric of interest. We present an efficient and scalable solution to find optimal paths through a combination of the theorem prover Isabelle [18] with the constraint solver Z3. Our approach has been inspired by a healthcare problem, namely how to detect conflicts between medications taken by patients with multiple chronic conditions, and how to find preferable alternatives automatically. In this paper, we focus on the theoretical foundations required to address the problem and our medical domain, but remind the reader of the more general applicability of our work and considerable practical benefits.

This paper provides a formal statement of the problem, a measure of inconsistency, and shows how the obtained problem can be turned into an optimisation problem. In addition, it illustrates how a SMT solver can be used to find a scalable solution to the proposed problem. As a final contribution, the paper introduces a general technique to combine Z3 with Isabelle in order to ensure that the SMT translation of the problem is formally correct.

Paper structure: Sect. 2 introduces the formalisation of the problem and our solution. Section 3 translates this formulation into an SMT context. In Sect. 4, a concrete application in the healthcare domain is introduced to motivate the approach introduced in this paper, and is used to evaluate our design through a basic implementation featuring simple input and output interfaces. Section 5 exposes the general technique we used to combine the theorem prover Isabelle and the SMT solver to guarantee the correctness of the SMT code illustrated in Sect. 3. Section 6 discusses related work, and Sect. 7 concludes.

2 Description of the Problem and Approach

Our problem is formulated formally as follows: we are given a list of simple directed acyclic graphs $G_i, i = 1, \ldots, n$, each with exactly one source node. Since each of the graphs is simple, each G_i can be thought of as a finite set of ordered pairs of nodes (j, k), each representing a directed edge from node j to node k. Therefore, we can define $G := \bigcup_{i=1}^{n} G_i$, and denote by $V(G')$ the set of nodes touched by any edge in $G' \subseteq G$. Further, we assume that $V(G_{i_1}) \cap V(G_{i_2}) = \emptyset$ for any $i_1 \neq i_2 \in \{i_1, \ldots, i_n\}$: i.e., distinct graphs have no nodes in common.

We want to obtain a list of paths, one for each given graph, each leading from the source of the corresponding graph to one of it sinks (we recall that a source is a node with no incoming edges and a sink one with no exiting edges). Such a list of paths must be determined so as to maximise a given *score*, which can be thought of as a metric of the compatibility of the resulting execution paths. Before defining how this score is computed, we need to describe how the needed input data and the output are encoded. Examples later clarify the need for these notions.

2.1 Score Model

To compute the score, we assume to be given further input, besides the list of graphs $G_1, \ldots, G_n$, as follows.

1. A map $t : G \to \mathbb{N} \times \mathbb{N}$ associating to each edge e a pair $(t^-(e), t^+(e))$, where $t^-(e)$ is the minimal time and $t^+(e)$ is the maximal time e has to wait before e can occur. We require $t^-(e) \leq t^+(e)$ for any $e \in G$. It can be used to bound the occurrence of tasks associated to nodes in a graph. For example, t^- expresses that the next task cannot start before t^- time units, and t^+ expresses that it must occur before t^+ time units.
2. A list $\tau := \{\tau_1, \ldots, \tau_n\}$ of integers, where τ_i specifies the instant at which the source node of the graph G_i is executed. This list can be used to express the requirement that different models start their execution at different times.
3. A finite set R of *resources*.
4. A map $M : V(G) \to 2^R$. $M(j)$ specifies a subset of resources among which one can be chosen in order to perform the task corresponding to node j.
5. A map $g : R \to \mathbb{Z} \times \mathbb{N}$ associating to each resource r an *effectiveness score* $g_1(r)$ and an *amount* $g_2(r)$. The effectiveness is a measure of how well a given resource performs a task, and the amount is how much of the resource is consumed for performing the task. For example, a hardware resource is needed for a given time, a medication must be taken at a given dosage, etc.
6. A map $I : R \times R \to \mathbb{Z}$ yielding an *interaction*. The interaction is an integer expressing how much two resources mutually boost or interfere, where a negative interaction means a counter-productive effect (i.e., diminishing the overall effectiveness of the two interacting resources).
7. A map $f : \mathbb{Z} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ combining an interaction between two resources, a time distance, and the amounts of the two resources to yield the component of the overall score for a given pair of resources. f takes into account the fact that the actual interaction between resources occurring at distinct nodes depends not only on the interaction between the resources (as defined at the previous point), but on their amount and on how much time passes between their occurrences. We will refer to the integer values returned by f as *interaction scores*.

2.2 Output

Given a list $G_1, \ldots G_n$ of directed acyclic graphs, a set of resources R and maps f, g, I, M, t as introduced in the previous section, the output is a triple of functions (F, c, m), each defined on the set of all nodes, $V(G)$.

F is a boolean function telling us which nodes are executed, $c\left(j\right)$ returns the instant at which node j is executed, while $m\left(j\right)$ is the resource picked to perform the task associated to the node j. We will use the notation $P\left[X\right]$ to indicate the image of the set X through the relation P; for example, $F\left[\{\text{true}\}\right]$ is the set of executed nodes. With this notation in place, we require that F, c and m satisfy all the following conditions:

1. the set on which F is true determines one path for each G_i, starting from the source of G_i and ending at one of its sinks; this is a way to represent the paths that we anticipated as our main goal at the beginning of this section. More formally, F satisfies the following requirement: for any $i \in \{1, \dots, n\}$ there is a finite sequence w^i of nodes of G_i such that (a) w_0^i is the source of G_i, (b) $w_{|w^i|-1}^i$ is a sink of G_i, (c) $\forall j \in \left\{1, \dots, \left|w^i\right| - 1\right\}$ $\left(w_{j-1}^i, w_j^i\right) \in G_i$, and (d) $\left\{w_0^i, \dots, w_{|w^i|-1}^i\right\} = F^{-1}\left[\{\text{true}\}\right] \cap G_i$.
2. $\forall j \in V\left(G\right),\ F\left(j\right) \rightarrow m\left(j\right) \in M\left(j\right)$;
3. for any $i \in \{1, \dots, n\}$, if j is the source node of G_i, then $c\left(j\right) = \tau_i$;
4. if there is an edge going from the node j to the node k, and j and k are executed, then
$$t^{-}\left(j,k\right) \leq c\left(k\right) - c\left(j\right) \leq t^{+}\left(j,k\right);$$
5. the global score:
$$\sum_{j \in F^{-1}[\{\text{true}\}]} g_1\left(m\left(j\right)\right) + \sum_{\substack{i_1, i_2 \in \{1,\dots,n\}, i_1 < i_2 \\ j \in F^{-1}[\{\text{true}\}] \cap V\left(G_{i_1}\right) \\ k \in F^{-1}[\{\text{true}\}] \cap V\left(G_{i_2}\right)}} f\left(I\left(m\left(j\right), m\left(k\right)\right), \left|c\left(k\right) - c\left(j\right)\right|, g_2\left(m\left(j\right)\right), g_2\left(m\left(k\right)\right)\right) \tag{1}$$
is maximal.

The first term in (1) sums the effectiveness of each picked resource for each executed node, irrespectively of whether distinct picked resources interact. The second term sums together the interaction scores of each pair of resources picked in distinct graphs. The interaction score for each of such pairs depends on the absolute interaction of the two resources (specified by I), the time distance separating the occurrence of the two resources, and the amount.

2.3 An Illustrative Example

We use a simplified example to motivate our formal problem. A more realistic example and our solution is given in Sect. 4.

Assume that a patient with an acute condition is hospitalised on day 0. There are two possible treatments for the condition: a non-surgical treatment and surgery. The two alternatives are represented by the two branches in the directed graph of Fig. 1 (left), where the source node represents the hospitalisation. The right branch represents the choice of surgery with nodes n_3 and n_4 denoting the steps implied by this choice (n_3: pre-surgical testing and n_4: the surgery itself). Each node in a treatment graph may have one or more ways of performing it. For example, in the case of pre-surgical testing it involves administering one of two drugs (d_1 or d_2), while in the case of the surgery, only one resource is present (we assume here that there is only one way to perform it). The left branch (with n_2) models the non-surgical choice, here associated to the prescription of drug d_0 (with no other choice available). The weights on the edges are the time constraints for the subsequent step: for example, after pre-surgical testing was performed (n_3), surgery cannot happen before 2 days have passed, but should happen within 4 days (for illustration purposes only).

Additionally, this particular patient suffers from a chronic condition, which requires him to take drug d_3 on even days and d_4 on odd days. This can be represented by a path graph, unfolding the alternation of d_3 and d_4 for a given finite number of days (Fig. 1 right).

It is known that surgery is preferred, but d_1 interacts negatively with d_3 and d_2 with d_4. We ignore drug dosages here, which our formalisation can handle as well. The problem we want to solve is to find how to best schedule steps in the hospitalisation, and how to choose between the non-surgical and the surgical treatment, taking into account the respective effectiveness and the interaction with the treatments for the chronic condition.

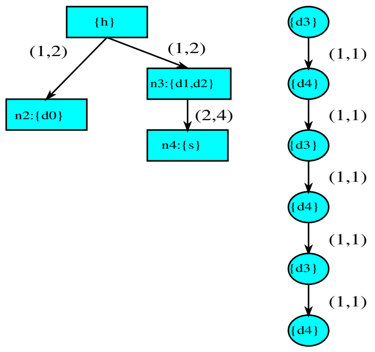


Fig. 1. A simple example problem.

3 SMT Translation

We now need to represent the notions presented in Sect. 2 in a way amenable to SMT computations. In doing so, we will write formulas close to the first-order logic language used by SMT solvers; for the sake of readability, however, we will employ some notational simplifications. In particular, we adopt infix notation instead of prefix notation, we omit type specifications, and we use subscripts and other typographic features not available in plain-text syntax of SMT-LIB.

To represent the output F, we introduce one boolean variable node_i for each node i (i.e., the truth value of node_i will yield $F(i)$). First, we must pick all the sources of the G_i's, so that the corresponding node variables will be asserted to be true. Afterwards, for each source, we must assert that exactly one of its children must be true. Then, for every child set to true, we must ask that exactly one of its children must be true, and so on, until no node has children (we reached a sink). Besides doing that, we want to make sure that no other node is selected.

Correspondingly, we generate, for the node i, the assertions

$$\text{node}_i \rightarrow \bigvee_{j\,|\,(i,j)\in G} \left(\left(\bigwedge_{\substack{k\neq j,\\ k\,|\,(i,k)\in G}} (\neg\, \text{node}_k) \right) \wedge \text{node}_j \right) \tag{2}$$

$$\left(\bigwedge_{(j,i)\in G} (\neg\, \text{node}_j) \right) \rightarrow \neg\, \text{node}_i, \tag{3}$$

where we adopted an informal intentional implosion of the arguments of connectives for reader's convenience, rather than writing the extensional form actually needed for SMT-LIB. For example, the expression $\bigwedge_{(j,i)\in G} (\neg\, \text{node}_j)$ appearing in (3) is a shortcut for scanning the input G to take all the first components of pairs having i as a second component (let us assume they are $\text{node}_{i_1}, \ldots, \text{node}_{i_p}$), and then writing the SMT expression *and* $\text{node}_{i_1} \ldots \text{node}_{i_p}$.

3.1 Scores in SMT

To represent the output c, we introduce one numerical variable clock_j for each node j. For each j, k such that $(j,k) \in G$, we create the following assertion:

$$\text{node}_j \wedge \text{node}_k \rightarrow \text{clock}_k - \text{clock}_j \geq t^-(j,k) \wedge \text{clock}_k - \text{clock}_j \leq t^+(j,k)\,.$$

Additionally, we impose that each source node happens at the time specified by τ; therefore, if j is the source node of G_i, we assert:

$$\text{clock}_j = \tau_i. \tag{4}$$

To represent the output m, we introduce one variable label_j for each node j, whose value is the name of the drug picked for the node j. Now we can introduce two kinds of scores, represented by integer SMT variables score_j and $\text{score}_{j,k}$, respectively: the first represents the effectiveness of the prescription associated to node j, while the second is the score generated by possible conflicts between the prescriptions picked for node j and for node k. We first zero out the scores for the non-executed nodes:

$$\neg\, \text{node}_i \rightarrow \text{score}_{i,j} = 0 \wedge \text{score}_i = 0 \wedge \text{score}_{j,i} = 0.$$

Since the variables node_j describe whether a node is executed the final optimisation, we generate the following assertions for each possible j, k:

$$\begin{gathered}\text{node}_j \wedge \text{node}_k \rightarrow \text{score}_{j,k} = \\ f\left(I\left(\text{label}_j, \text{label}_k\right), \left|\text{clock}_k - \text{clock}_j\right|, g_2\left(\text{label}_j\right), g_2\left(\text{label}_k\right)\right).\end{gathered}$$

We finally assign the sum of all $\text{score}_{i,j}$'s and score_i's to a variable, and ask for an SMT solution to all the assertions which maximises that variable; this requires the optimizing version of the SMT solver Z3, νZ (also known as μZ or Z3Opt) [2].

The map I assigns a degree to the possible conflicts; this assignment, together with f, determines how the interactions influence the resulting choice of resources, timing, and the execution paths in the different models. These choices add flexibility to the whole approach but, on the other hand, need to be done by a domain expert.

4 Evaluation and Use Cases

In healthcare management and practice, as in other domains, clinical and medical procedures are streamlined by adopting standardised guidelines. In particular, treatments for common chronic conditions have been subject to various clinical trials, and the outcomes documented in *clinical pathways* specifying accepted treatment steps, possible alternatives, and recommendations to follow. Clinical pathways are informal flowcharts, with natural language annotations, and as such they can be formalised as directed acyclic graph structures, usually with one initial node (the source), and each node representing a medication prescription. Applying a single clinical pathway to a given patient is subject to a number of variable aspects and requirements, for example:

1. pathways typically present alternatives from which one is chosen;
2. there is often a choice to be taken among equivalent drugs in a group;
3. the time separating subsequent steps in a pathway is typically not fixed, being liable to be adapted to the context and the patient situation;
4. the dosage of the chosen drugs influences how drugs mutually interact;
5. the set of chronic conditions typically changes over the patient's life span: when this happens, even the treatment of the pre-existing conditions must be reconsidered.

On patients suffering from multiple chronic conditions, several pathways have to be applied concurrently, so that the number of possible combinations of these parameters increases dramatically. Our goal is to present the clinician suggestions about which choice of the parameters above is the best. The model presented in the previous sections allows us to capture all these aspects: the resources correspond to the single treatments (e.g., drugs, or surgical procedures), the effectiveness score expresses how well a single treatment performs, the amount can be used to express the dosage of a drug, and I can model, for example, drug-drug interaction. For consistency and comparison, we evaluate our design on a well-known case [8] of a hypothetical 79-year-old woman with five diseases: chronic obstructive pulmonary disease (COPD), diabetes mellitus (type 2), hypertension, osteoarthritis, and osteoporosis. To this end, we extracted data to represent pathways and scores from two sources, respectively: NICE pathways[1], publicly available as informal flowcharts with accompanying text, and the website drugs.com to derive the scores.

Given the form in which pathways from NICE are presented, we extracted the data needed as input to our design manually, for each of the five conditions we are considering (see [12]). We pruned the nodes not liable to cause conflicts (e.g., "ongoing monitoring of HbA_{1c}"), generated an adjacency relation describing the underlying graph, and attached a list of possible medications for each node (for example, when a medication group such as *Sulfonylurea* was associated to a node, we inserted the list of all the medications in the group in the corresponding node of the graph we generated). The resulting graphs are visible in Figs. 2 and 3,

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

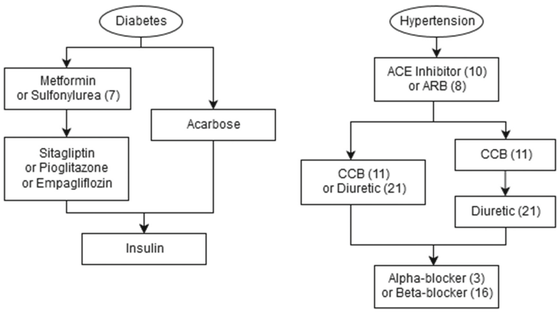

Diabetes
Metformin or Sulfonylurea (7)
Acarbose
Sitagliptin or Pioglitazone or Empagliflozin
Insulin
Hypertension
ACE Inhibitor (10) or ARB (8)
CCB (11) or Diuretic (21)
CCB (11)
Diuretic (21)
Alpha-blocker (3) or Beta-blocker (16)

Fig. 2. Pharmaceutical graphs for Diabetes and Hypertension based on NICE pathways. ACE – angiotensin-converting-enzyme, ARB – angiotensin receptor blocker, CCB – calcium channel blocker.

where numbers in brackets represent the number of individual medications in a group. They contain a total of 127 distinct medications. Some of the graph parameters, such as the duration limits (described by the functions t^- and t^+ introduced in Sect. 2) can depend on the single patient and on the context, and are therefore not present in the data provided by NICE. For evaluation purposes, we generated suitable data reasonable for our purposes to fill the gaps (this will be done by clinicians in the future). To retrieve all the possible drug conflicts, we used the interaction engine on the mentioned site drugs.com[2], and obtained a classification (minor, moderate and major): 178 minor conflicts, 3033 moderate conflicts, 270 major conflicts, for a total of 3481 conflicts, an amount too large to analyse manually.

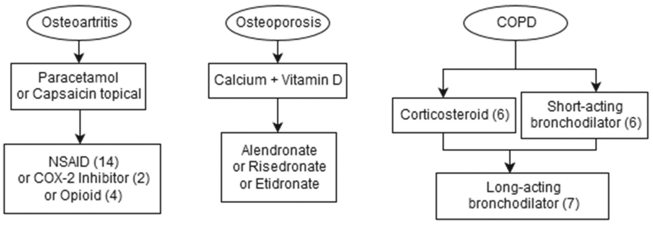

Osteoartritis
Paracetamol or Capsaicin topical
NSAID (14) or COX-2 Inhibitor (2) or Opioid (4)
Osteoporosis
Calcium + Vitamin D
Alendronate or Risedronate or Etidronate
COPD
Corticosteroid (6)
Short-acting bronchodilator (6)
Long-acting bronchodilator (7)

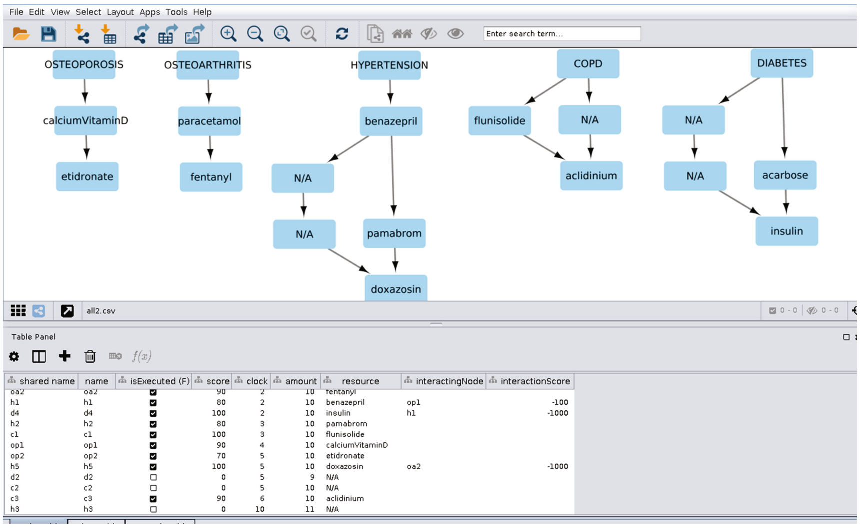
Fig. 3. Pharmaceutical graphs for three additional conditions. NSAID – nonsteroidal anti-inflammatory drug, COX-2 – Cyclooxygenase-2.

4.1 Results

The data extracted as explained above was represented in a textual, comma-separated value (CSV) format, and we built a simple implementation of our

[2] http://www.drugs.com/drug_interactions.html.

design, parsing these files, generating SMT code, producing text representations of (F, c, m), together with a graphical representation of the output using Cytoscape[3] [24]. For computations, we assigned the values of -100, -1000 and -5000 to minor, moderate and major conflicts, respectively. Another important parameter we had to set is f, which weights the conflicts according to time separation and dosage. We chose a simple form, which takes the conflict and zeroes it as soon as the time distance is greater than 8 time units or one of the dosages is less than 10 dosage units.



shared name	name	isExecuted (F)	score	clock	amount	resource	interactingNode	interactionScore
oa2	oa2	✓	90	2	10	fentanyl		
h1	h1	✓	80	2	10	benazepril	op1	-100
d4	d4	✓	100	2	10	insulin	h1	-1000
h2	h2	✓	80	3	10	pamabrom		
c1	c1	✓	100	3	10	flunisolide		
op1	op1	✓	90	4	10	calciumVitaminD		
op2	op2	✓	70	5	10	etidronate		
h5	h5	✓	100	5	10	doxazosin	oa2	-1000
d2	d2	☐	0	5	9	N/A		
c2	c2	☐	0	5	10	N/A		
c3	c3	✓	90	6	10	aclidinium		
h3	h3	☐	0	10	11	N/A		

Fig. 4. Time- and dosage- aware optimal solution generated by Z3 for a hypothetical five-comorbid patient, as represented in the graphical front-end.

The output is shown in Fig. 4: the text in the nodes of Figs. 2 and 3 have been substituted by the drug ultimately picked in the maximised solution, except for the non-executed nodes, which are marked with "N/A". In the lower part of the interface, the user can see the node list with all the relevant information: the picked drug, the clock (i.e., the value returned by c introduced in Sect. 2.2) for the corresponding event, the score, the conflicting nodes (if any) and the conflict score (if any). The user can also click on single node to highlight only the relevant data. For this example, the total score is -1220, while the total conflict is -2100, composed as follows:

[3] Cytoscape is a software platform for the visualisation of complex network integrated with any type of attribute data, with a focus on molecular interaction networks. It was chosen among several other visualisation platforms because of the variety of layout algorithms available, the simple data import/export format available, and the relatively moderate adoption effort it required from us.

- the insulin-benazepril interaction contributes -1000,
- the doxazosin-fentanyl interaction contributes -1000,
- the benazepril-calciumVitaminD interaction contributes -100.

To assess the performance of the proposed approach, we timed the presented implementation. The average running time is 28.1 s, including the SMT code generation from the input data (which, however, takes a negligible amount of time); these results were obtained on an off-the-shelf laptop with 4GB RAM and a dual-core 2GHz CPU, running a 32-bit Linux OS. The actual run-times would likely be less, since in real situations a number of possibilities are excluded and additional restrictions often apply. For example, a portion of drugs could be excluded a priori because not available, or because known to have too many interactions; what is more, the doctor might impose manually a choice in a graph branch, thereby reducing the combinations.

4.2 Introducing Time Offsets

Suppose a patient is being treated for a number of conditions, when she gets diagnosed an additional condition (requirement (5) of Sect. 4). We want to show how the time-awareness of our design helps when facing such a situation. Let the instant in which the additional condition gets diagnosed be denoted by 0. Starting from time 0, we want to re-assess the patient's situation to get suggestion about which changes in the therapy should be implemented, assuming that the treatment for the pre-existing conditions started at some time in the past $-x$. To achieve that, it will suffice to change condition (4) (Sect. 3.1) whenever j refers to an initial node of the pre-existing conditions:

$$\text{clock}_j = -x,$$

while we assert $\text{clock}_j = 0$ when the index j refers to the new condition.

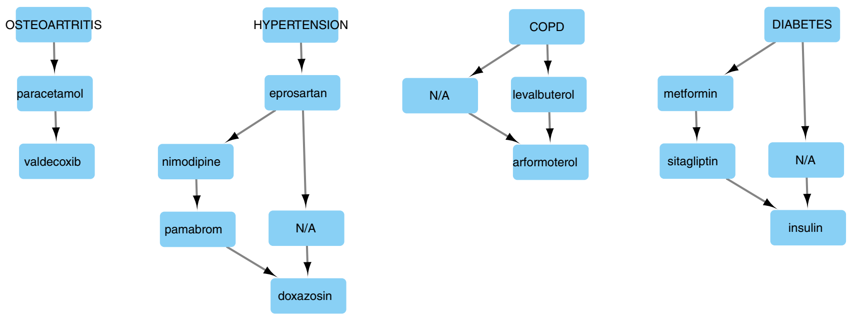


Fig. 5. Time- and dosage- aware optimal solution generated by Z3 with four concurrent morbidities.

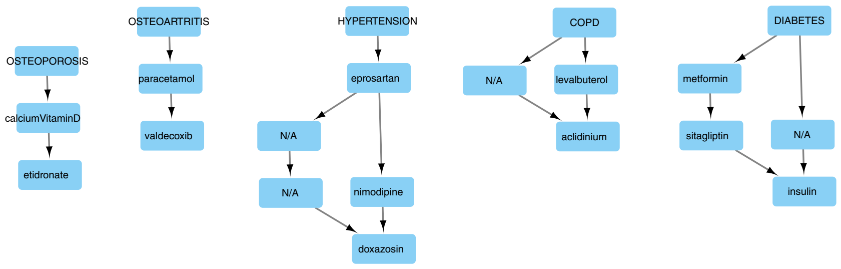


Fig. 6. The optimal solution after a fifth condition was added with a time offset.

We evaluated how this can change the recommendation given by our system in a very simple case. First, we ran the simulation as in the preceding section, except that we did not include osteoporosis, and obtained the solution shown in Fig. 5. Then we added back osteoporosis with a starting time 0, while changing the starting time of the remaining four conditions to −6. The result is shown in Fig. 6. As it can be seen by comparing the results, the addition of the new condition (osteoporosis) modified the suggested pathway for one of the existing conditions (hypertension). This means that the clinician could consider back-tracking and changing the previously established therapy in view of the modified clinical situation. It should be noted that the running time for these examples is substantially the same as that for the example in the previous section ($\leq$30 s).

5 Formal Verification

Formulas (2) and (3) correspond to SMT assertions whose number quickly grows even for small graphs. There are more immediate ways of expressing the same problem as an SMT problem, but they turn out to be significantly less efficient. Our idea (extending the approach taken in [7]) is to exploit Isabelle's SMT-LIB generator to automatically produce SMT code from Isabelle definitions that we can formally prove to be correct via formal Isabelle theorems. This SMT code will typically be not as efficient as the SMT code that we will effectively run; however, we can use an SMT solver to prove that the two are equivalent. This allows us to infer (if we trust the SMT solver) that the formal correctness theorems proved in Isabelle apply to the SMT code that we will effectively run.

This approach is illustrated in Fig. 7. While this scheme can be applied generally, we use formulas (2) and (3) (Sect. 3) to illustrate it. We rewrite them as the following equivalent Isabelle/HOL definitions:

```
abbreviation "conditionTwo' G F ==
(∀ p. (F p & ¬ isSink' G p) → (∃! c. (G p c & F c)))"

abbreviation "conditionThree' G F ==
∀ c. (¬ isSource' G c & (∀ p. G p c → ¬ F p)) → (¬ F c)"
```

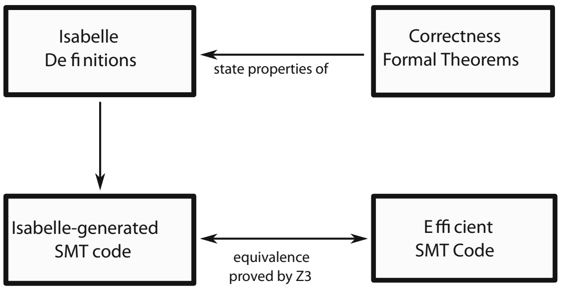


Fig. 7. Overview of the formal verification of the SMT code.

Here, G is a graph and F is a set of nodes expressed as predicates (i.e., there is an edge from m to n if and only if $G\ m\ n$ is true, and the node m is selected if and only if Fm is true), and `isSink' G s` is a predicate telling whether s is a sink for the graph G; similarly for `isSource'`. To these definitions, we add another, corresponding to the requirement that all the sources must be selected:

```
abbreviation "conditionOne' G F ==
              (∀ s. isSource' G s → F s)"
```

Now, we put together these conditions:

```
abbreviation "formalConditions' G F == conditionOne' G F &
                   conditionTwo' G F & conditionThree' G F"
```

and use Isabelle's SMT generator to produce corresponding SMT code:

```
lemma assumes "formalConditions' G F" shows False
sledgehammer[provers=z3, minimize=false,
                 timeout=1, overlord=true] (assms).
```

`sledgehammer` is an Isabelle tool for theorem proving, and it works by negating the thesis of the considered theorem, and to challenge an SMT solver (or other tools) to consider whether the obtained problem is satisfiable. If it is not, it can use the information to find a proof. In this case, we are not interested in theorem proving, but only in the generated SMT code. More details can be found in [7], where we directly use the generated code to compute. Here, we make an indirect use of it: the generated code is used to define a boolean SMT variable `formal`, containing all the corresponding assertions. That is, `formal` will be true if and only if the automatically generated `formalConditions' G F` is true. We do the same for the assertions illustrated at the beginning of Sect. 3, to obtain an SMT variable that we call `efficient`. In other words, `efficient` will be true if and only if each node$_j$ obeys the group of assertions in Sect. 3. We add the requirement that Fj = node$_j$ for any node j. If this group assertions and the group of assertions enclosed in `formalConditions'` introduced above were not equivalent, there should be a choice of F satisfying one but not the other group. Therefore, we express the following SMT assertion:

```
(assert (or (and formal (not efficient))
            (and (not formal) efficient))).
```

If the SMT solver returns `(unsat)`, this means that the two conditions are indeed equivalent. It should be noted that this check is only needed once; for the example of Sect. 4, we obtained an `(unsat)` answer in about 5 s.

Once we have the guarantee of the equivalence between the SMT code that we run and the SMT code generated from Isabelle definitions, we can start proving formal theorems in Isabelle about these definitions, in order to make sure they present the intended properties.

As an example, we discuss how to prove that the nodes selected by the SMT assertions labelled as `formal` is indeed a path leading from a source to a sink for the corresponding graph. To do so, we start from defining in Isabelle/HOL the canonical notion of walk in a simple directed graph:

```
definition "isWalk G w == w≠[] →
                  (∀i∈{1..<size w}. (w!(i-1),w!i)∈G)",
```

which amounts to asking that any consecutive entries of the node list `w` are joined by an edge (note that `w!0`, `w!1`, ... are the entries of the list `w`). We want to prove that any selection of nodes of a given graph `G` obtained from the assertions `formal` are the entries of a walk which starts from the source of `G`, ends in a sink of `G` and has no repetitions. To this end, we formally proved the following theorem:

```
theorem assumes "finite G" "card (sources G)=1"
"irrefl (trancl G)" "conditionOne (set2pred G) (unset F)"
"conditionTwo (set2pred G) (unset F)"
"conditionThree (set2pred G) (unset F)" shows
"∃ w. distinct w & isWalk G w & last w∈sinks G &
              {hd w}=F∩sources G & set w=F∩nodes G"
```

The converter `set2pred` serves to pass from `G` represented as a set of pairs to its representation as a binary boolean predicate. In Isabelle, they have different types, and using sets is more expressive than using predicates, leading to more readable statements; on the other hand, sets are not present natively in the SMT-LIB language, and using predicates is therefore preferable. Similarly, `unset` passes from the set `F` to its representation as a unary boolean predicate.

The theorem assumes that the graph is finite, and that there is only one source. Given that it is represented as a set of ordered pairs, it is inherently directed, while `irrefl (trancl G)` states that its transitive closure is not reflexive. This is the set-theoretical way of requiring that `G` is acyclic. Under these assumptions, plus the requirements that the SMT solver proved to be equivalent to the SMT assertions that we execute, the theorem states that the set `F∩nodes G` is the entries set of some walk `w` for `G` which starts at the source (`hd w` is the first entry of `w`), ends at a sink, and has no repetitions (keyword `distinct`). The conditions about `w` in the thesis of the theorem above correspond to requirements at the point (1) of the list of conditions at the beginning of Sect. 2.2.

6 Related Work

The problem of model composition has been treated by using SMT solvers or constraint solvers in previous works [4,6,23,27,28]. To the best of our knowledge, no other existing approach to the problem adopts a metric to be maximised, as we did here. From an abstract viewpoint, there are some similarities between the model we proposed in Sect. 2 and that presented in [14] where, however, there is only one graph involved, and the semantics associated to it and the problem to be solved are entirely different. In [12], the specific problem of the composition of pharmaceutical graphs is solved by the usage of SMT solvers in a way which inspired the present paper. Here, we addressed two of the main limitations conceded in [12], namely the lack of dosage and timing information, presenting an SMT-based design which adds those information coordinates to obtain a more flexible and realistic solution to the problem of minimising conflict in multiple pathway applications on multimorbid patients.

The issue of verifying SMT code, a solution to which we discussed in Sect. 5, is underexplored: indeed, SMT solvers are typically used the other way around, i.e., to aid in the verification of software or in formal proofs [3].

The problem of automated detection and resolution of pathway conflicts in patients with multimorbidity is gaining considerable attention. In [10], ontologies are used to represent pathways, with one additional ontology (Merge Representation Ontology, MRO) created by interviewing clinicians to identify merging criteria, and instantiated to find merge points of two given pathways. No information is given, however, on how adaptable the approach is and whether the instantiation process is automatic. Similarly, recent work proposing extensions to the existing GLARE project [21] also uses ontologies, but focuses on soliciting clinicians' input for conflicts solution, rather than automating it. Constraint logic programming (CLP) to express and deal with conflicts is proposed in [16]. While CLP solvers and SMT solvers have fundamental similarities, their expressiveness, background technologies and domains of application differ. With the efficiency of the current, mutually competitive, SMT solvers growing at a steadfast pace, their lower expressiveness is getting more and more effectively compensated. Another route has been brought forward in [15], where an ad-hoc algorithm is proposed based on formal rules expressing the actions to be taken upon the happening of given premises expressing conflicts. Whereas this solution has the advantage of accurately solving conflicts, based on the actual medical meaning of the rules themselves, it relies on specific software, which looks problematic in view of extensions and further development.

7 Conclusions

We introduced a general way of expressing the problem of execution paths across combined behavioural models of complex systems as an optimisation problem on a suitably defined score metric, expressed the latter problem in SMT language and evaluated the whole approach on a concrete, well-known healthcare problem.

Additionally, we formally proved the correctness of our SMT code through a novel application of theorem prover Isabelle, exploiting the latter's ability of generating SMT code. This is important because writing SMT code directly is time-consuming and error-prone while, and the existing interfaces of SMT solvers with higher-level languages (e.g., APIs) are currently (as far as we know) not formally verified.

Our approach can be adapted to a wide range of conditions: for example, by imposing manual restrictions on the number of resources, by manually imposing a chosen path in some of the graphs, by changing how f operates, or by offsetting the application of distinct models time-wise; in the latter case, we showed how this can be exploited in a simple way to manage the changes to a patient's therapy following the diagnose of a new condition.

While the current execution time is tolerable for interactive use, we believe that in real situations it can be dramatically lower, on average. Nonetheless, we are experimenting with *scopes*, a feature of Z3 which allows incremental solving by specifying a block of assumptions which are always held, together with a stack of additional assumptions which can be pushed and popped repeatedly. When suitably handled, this feature is likely to allow to reduce the SMT computation time upon modifications of the inputs, since a relevant part of it will remain unchanged throughout the modifications (e.g., the topology of the graphs). As a further, orthogonal code improvement strategy, we are investigating restrictions of the theory known to the solver (using the `(set−logic)` command); this requires expressing our SMT-LIB code into a form allowed by the selected restriction. The code presented here can be run under the `UFLIA` theory, but we are positive about it being adaptable to a fragment of `UFLIA` (e.g., `QF_UFLIA`), which would likely result in better performance.

The presented design has some limitations. First, while it can find an execution minimising the conflict, it cannot suggest additional remedies to neutralise or mitigate the arising conflicts. Secondly, it does not handle, currently, directed graph presenting cycles (we are not aware of any existing approach supporting the presence of cycles in our main application domain, e.g., clinical pathways composition). Third, the score calculation could be made context-aware by considering previously occurred resources in the same graphs. A possible attack to the last issue could consist in generalising the maps f and g by adding arguments to them expressing the context: we are experimenting in this sense.

References

1. Araújo, J., Whittle, J., Kim, D.: Modeling and composing scenario-based requirements with aspects. In: Proceedings of the 12th IEEE International Requirements Engineering Conference, pp. 58–67. IEEE Computer Society Press (2004)
2. Bjørner, N., Phan, A.-D., Fleckenstein, L.: *νZ* - an optimizing SMT solver. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 194–199. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_14
3. Bobot, F., Filliâtre, J.C., Marché, C., Paskevich, A.: Why3: shepherd your herd of provers. In: Boogie 2011: First International Workshop on Intermediate Verification Languages, pp. 53–64 (2011)

4. Bowles, J.K.F., Bordbar, B., Alwanain, M.: A logical approach for behavioural composition of scenario-based models. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 252–269. Springer, Cham (2015). doi:10.1007/978-3-319-25423-4_16
5. Bowles, J., Bordbar, B.: A formal model for integrating multiple views. In: Seventh International Conference on Application of Concurrency to System Design, 2007, ACSD 2007, pp. 71–79. IEEE Computer Society Press (2007)
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: 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE Computer Society (2016)
8. Boyd, C.M., Darer, J., Boult, C., Fried, L.P., Boult, L., Wu, A.W.: Clinical practice guidelines and quality of care for older patients with multiple comorbid diseases: implications for pay for performance. JAMA **294**(6), 716–724 (2005)
9. Jackson, D.: Software Abstractions: Logic, Language and Analysis. MIT Press, Cambridge (2006)
10. Jafarpour, B., Abidi, S.S.R.: Merging disease-specific clinical guidelines to handle comorbidities in a clinical decision support setting. In: Peek, N., Marín Morales, R., Peleg, M. (eds.) AIME 2013. LNCS, vol. 7885, pp. 28–32. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38326-7_5
11. Klein, J., Hélouët, L., Jézéquel, J.: Semantic-based weaving of scenarios. In: Proceedings of the 5th International Conference on Aspect-Oriented Software Development, pp. 27–38. ACM (2006)
12. Kovalov, A., Bowles, J.K.F.: Avoiding medication conflicts for patients with multimorbidities. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 376–390. Springer, Cham (2016). doi:10.1007/978-3-319-33693-0_24
13. Liang, H., Diskin, Z., Dingel, J., Posse, E.: A general approach for scenario integration. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 204–218. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87875-9_15
14. Lombardi, M., Milano, M., Benini, L.: Robust scheduling of task graphs under execution time uncertainty. IEEE Trans. Comput. **62**(1), 98–111 (2013)
15. López-Vallverdú, J.A., Riaño, D., Collado, A.: Rule-based combination of comorbid treatments for chronic diseases applied to hypertension, diabetes mellitus and heart failure. In: Lenz, R., Miksch, S., Peleg, M., Reichert, M., Riaño, D., ten Teije, A. (eds.) KR4HC/ProHealth -2012. LNCS, vol. 7738, pp. 30–41. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36438-9_2
16. Michalowski, M., Wilk, S., Michalowski, W., Lin, D., Farion, K., Mohapatra, S.: Using constraint logic programming to implement iterative actions and numerical measures during mitigation of concurrently applied clinical practice guidelines. In: Peek, N., Marín Morales, R., Peleg, M. (eds.) AIME 2013. LNCS, vol. 7885, pp. 17–22. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38326-7_3
17. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_24
18. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). doi:10.1007/3-540-45949-9

19. OMG: Business Process Model and Notation. Version 2.0. OMG (2011). http://www.omg.org, document id: formal/2011-01-03
20. OMG: UML: Superstructure. Version 2.4.1. OMG (2011). http://www.omg.org, document id: formal/2011-08-06
21. Piovesan, L., Molino, G., Terenziani, P.: An ontological knowledge and multiple abstraction level decision support system in healthcare. Decision Anal. **1**(1), 1 (2014)
22. Reddy, R., Solberg, A., France, R., Ghosh, S.: Composing sequence models using tags. In: Proceedings of MoDELS Workshop on Aspect Oriented Modeling (2006)
23. Rubin, J., Chechik, M., Easterbrook, S.: Declarative approach for model composition. In: MiSE 2008, pp. 7–14. ACM (2008)
24. Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T.: Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res. **13**(11), 2498–2504 (2003)
25. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. IEEE Trans. Softw. Eng. **35**(3), 384–406 (2009)
26. Whittle, J., Araújo, J., Moreira, A.: Composing aspect models with graph transformations. In: Proceedings of the 2006 International Workshop on Early Aspects at ICSE, pp. 59–65. ACM (2006)
27. Widl, M., Biere, A., Brosch, P., Egly, U., Heule, M., Kappel, G., Seidl, M., Tompits, H.: Guided merging of sequence diagrams. In: Czarnecki, K., Hedin, G. (eds.) SLE 2012. LNCS, vol. 7745, pp. 164–183. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36089-3_10
28. Zhang, D., Li, S., Liu, X.: An approach for model composition and verification. In: NCM 2009, pp. 1102–1107. IEEE Computer Society Press (2009)