Chapter 9
Balancing Prescriptions with Constraint Solvers

Juliana K. F. Bowles and Marco B. Caminati

Abstract Clinical guidelines are evidence-based care plans which detail the essential steps to be followed when caring for patients with a specific clinical problem, usually a chronic disease (e.g. diabetes, cardiovascular disease, chronic kidney disease, cancer, chronic obstructive pulmonary disease, and so on). Recommendations for chronic diseases include the medications (or group of medications) to be given at different stages of the treatment plan. We present an automated approach which combines constraint solvers and theorem provers to find the best solutions for treatment according to different criteria, and avoiding adverse drug reactions as much as possible. We extended the approach here to further refine the choice(s) to avoid dangerous or undesirable side effects.

9.1 Introduction

Clinical guidelines are published in the UK by the National Institute of Health and Care Excellence (NICE[1]) for England and Wales, and the Scottish Intercollegiate Guidelines Network (SIGN[2]) for Scotland. Clinical guidelines are evidence-based care plans, which detail the essential steps to be followed when caring for patients with a specific clinical problem and play an important role in improving health care for people with long-term conditions. There are guidelines for managing the treatment for chronic diseases such as diabetes, cardiovascular disease, chronic kidney

[1] NICE www.nice.org.uk.
[2] SIGN www.sign.ac.uk.

This research is supported by EPSRC grant EP/M014290/1 and MRC grant MR/S003819/1.

J. K. F. Bowles (✉) · M. B. Caminati
School of Computer Science, University of St Andrews, St Andrews KY16 9SX, United Kingdom
e-mail: jkfb@st-andrews.ac.uk

M. B. Caminati
e-mail: mbc8@st-andrews.ac.uk

P. Liò and P. Zuliani (eds.), *Automated Reasoning for Systems Biology and Medicine*, Computational Biology 30,
https://doi.org/10.1007/978-3-030-17297-8_9

disease, cancer, chronic obstructive pulmonary disease and so on. Guidelines include recommendations for the medications (or group of medications) to be given at different stages of the treatment plan as well as alternatives. When patients have multiple chronic conditions, aka *multimorbidity*, they are implicitly following several of the clinical guidelines for their individual diseases in parallel. Clinical guidelines make recommendations for treatments of chronic conditions but often do not take into account the possible presence of comorbidities. In fact, in the presence of multimorbidity, current guideline recommendations rapidly lead to polypharmacy without providing guidance on how best to prioritise recommendations [17]. As a result, it is possible for patients to take medications that lead to adverse drug reactions, or for particular combinations of drugs to be less effective if administered at the same time. In precision therapeutics, the aim is to tailor medical treatment to the individual characteristics of each patient which includes finding the right set of drugs for a patient with multimorbidities.

In recent work, we have explored how formal methods can help with the development of an automated framework that combines efficient and formal verification techniques, such as constraint solvers and theorem provers, to identify steps in different guidelines that cause problems if carried out together (e.g. two drugs prescribed for different conditions may interact, food may interact with a drug, health recommendations may contradict with each other) whilst at the same time find the preferred alternative according to a certain criteria (e.g. drug efficacy, prevalent disease, patient allergies, preferences, etc.) [8, 13]. Future integration of such techniques in practice can lead to the development of clinical decision support systems to manage treatments for patients with complex needs and multimorbidities. The need for this has been stressed in Hughes et al. [17].

In Kovalov and Bowles [20], we introduced medication effectiveness (given by drug companies) as the only criteria for finding the best solution. The approach associated a positive score to each medication capturing effectiveness, and a negative score to pairs of medications with known adverse reactions. This score is used by the SMT solver to find the ideal solution with the highest possible score. This paper extends our work further by expanding the search criteria whilst being able to generate the top three alternatives that reduce the identified inconsistencies according to the chosen criteria. We can use modelling languages such as BPMN [16, 25] to capture the details of a clinical guideline, as we did in Bowles et al. [13]. In the present paper, however, we focus on the underlying formalisation of guidelines and our SMT solver based approach for the search. The formal model used is the labelled event structure (LES) [32].

Our approach takes two or more clinical guidelines for patients with multimorbidities (captured as LES), and detects whether when patients are at different stages of their conditions, the combination of medications taken by such patients is safe and if not computes preferable alternatives. The search involves checking other medications in a group and backtracking to previous decision points and reversing a decision to find better solutions according to certain criteria. The list of alternatives can be fine-tuned to suit individual patient's preferences, such as, for instance avoiding specific undesirable side effects, but will otherwise take into account medication dosage

and timing, and the prevalent disease (if applicable). The approach is flexible and we can further explore different parameters as desired (e.g. cost, therapeutic efficacy, number of medications prescribed, genomic biomarkers if known, and so on) to find the ideal set of solutions. We interpret these parameters as integer variables and make use of the arithmetic capabilities of SMT solvers such as Z3 to compute optimal solutions for subsets of parameters of interest. Behind the scenes, the correctness of our approach is established by the theorem prover Isabelle.

This paper is structured as follows. We describe the background, including related work, and contribution of the present paper in Sect. 9.2, and recall our formal model (labelled event structures) as needed for this paper in Sect. 9.3. Section 9.4 describes how Isabelle and Z3 are combined to compute the best treatment paths under certain conditions, whilst Sect. 9.5 illustrates how their interplay allows to verify some aspects of this computation. We conclude the paper with a discussion of future work in Sect. 9.6.

9.2 Context and Contribution

When considering an approach to model treatment plans for patients with multimorbidities, a starting point are the models for the guidelines of individual diseases. Each guideline has a process-like description: after diagnosis a patient follows a sequence of steps, some steps may be carried out in parallel (for instance, a blood test and an X-ray), under some conditions there maybe alternative steps available (for instance, in type 2 diabetes patients may be offered metformin or a sulphonylurea as their first stage medication) and it may be necessary to repeat steps (for instance, for patients with diabetes glycated haemoglobin (HbA1c) is measured regularly). There are many modelling languages with notions of sequence, alternative, iterative and parallel behaviour, which could be used to describe them, such as different notations within UML [26], BPMN [25], Petri nets [28], process algebras, and so on. Composing guidelines for managing conditions such as type 2 diabetes, hypertension, and chronic kidney disease (CKD) would give indications for treating patients with these three conditions.

In recent years, several automated approaches have been developed to simplify the task of composing a variety of models, typically partial specifications from UML models [1, 5, 10–12, 19, 22, 27, 29–31, 33]. In our own work, we have used a combination of SAT and SMT solvers [10–12] to combine behavioural models and have shown the result in a visual manner. The idea is straightforward: behavioural models are formalised as constraints expressed in first-order logic, and the conjunction of the constraints from all models are fed to a solver to generate the solution for the composition. If two or more constraints give rise to a contradiction no solution can be produced and no valid composition model exists.We used a SAT solver based on Alloy [18] in Bowles et al. [10, 11] and the SMT solver Z3 [23] in Bowles et al. [12] and all subsequent work. In Bowles et al. [12] we had shown that Z3 outperforms Alloy as the complexity of the examples increases. Even though the use of Alloy is

common in the literature for model composition (e.g. [29, 33]), our own work was the first to make use of Z3 [12]. Initially, we made little use of Z3's powers such as arithmetics and arithmetic optimisation, and we also did not explicitly deal with inconsistent constraints. We have addressed both points more recently in Kovalov and Bowles [20], Bowles and Caminati [6].

In Bowles and Caminati [6], we used labelled event structures [32] as a true-concurrent semantic model of sequence diagrams [4, 21], and combined the theorem prover Isabelle [24] with the SMT solver Z3 [23] to detect inconsistencies over the model and solve partial specifications. In the context of clinical guidelines, the approach can be adapted not only to identify inconsistencies—in which case the SMT solver is unable to produce a solution but identifies the conflicting events—but to search for optimal treatment paths that minimise the conflicts. We note that for patients with complex conditions and subject to polypharmacy as a consequence it may be unavoidable to have adverse reactions from some of the medications given, but ideally this should be kept to a minimum.

In this paper, we extend earlier work done in Kovalov and Bowles [20] which used integer variables to encode drug effectiveness and drug interactions. We add measures for the likelihood of side effects associated to drugs, and explore the arithmetic capabilities of SMT solvers such as Z3 to search for the ideal solutions which minimise conflicts due to drug interaction and avoid undesired side effects. For example, sulphonylureas can cause hypoglycemia (low blood sugar) and weight gain, and these are relatively common side effects. In addition, some users may suffer an allergic reaction during the initial weeks of treatment, resulting in itchy red skin/skin rashes. Taking into account side effects of medications as a criteria of choice is new to this paper. Also new here is that our approach finds the best three treatment plans for clinicians to choose from.

We deal with side effects in our framework in two ways:

1. A side effect is captured as a Boolean variable. If a side effect `s` is to be avoided, we ignore paths that contain medications which may be associated to `s`. This may sometimes be necessary, but other times be too restrictive.
2. We capture the degree of likelihood of a side effect for a drug: `very common`, `common`, `occasional`, `rare`, `very rare`. We assign a probability bound to a side effect for a drug, where, for instance `rare` may mean a likelihood of occurrence of less than 10%. For example, itchy red skin or skin rash is a rare or possibly even very rare side effect for most sulphonylureas.

Continuing the work started in Bowles and Caminati [6], we exploit the interface between Isabelle and Z3 to obtain a versatile tool for our search for optimal treatment paths in complex scenarios. We have, for instance used Isabelle to check the correctness of our models (LES), obtain their composition (if it exists) and fill any gaps while being able to prove at any point that the models are valid [6]. Here, if different care guidelines for chronic conditions are being applied to the same patient, we consider the following:

- One disease may have a higher *priority*, possibly due to a higher risk.
- Some of the possible medications prescribed at a given step in the guidelines may be known to be more *effective*. For instance, the use of metformin in the treatment of type 2 diabetes tends to be the first medication of choice.
- For a patient, the time of *diagnosis* for his/her different conditions rarely occur at the same time. For example, (poorly managed) hypertension may lead to type 2 diabetes in the future. This makes it possible for a patient to start to follow different care guidelines at different times in a *dephased* manner.
- *Side effects* (or their avoidance) often influence which medications are prescribed to a patient, and sometimes also reflect patient's preferences. For instance, the use of metformin in the treatment of type 2 diabetes is often preferred because it lacks the side effects of drugs like sulphonylureas. Standard metformin can, nonetheless, cause gastrointestinal intolerance.
- Further *constraints* may arise from allergies that a patient has to one or more drugs.

Dephasing is a technique explored first in Bowles and Caminati [7] which adjusts when the different conditions have to be considered together. This means that we can ignore conflicts that cannot arise since the medications that could create problems are no longer reflecting the current treatment. Instead, we focus on the present and imminent medication choices and interactions. Further constraints can be used to add allergies as well as any additional constraints as needed. If a patient has gastrointestinal intolerance as a consequence of standard metformin, then the medication should be avoided and replaced by another. It can be treated as a Boolean variable in the same way that we avoid side effects altogether.

In the context of our work, it is crucial to define flexible automated techniques able to consider all the information above in order to compute the best possible treatment plan for each patient. Patients with similar conditions but, for instance different pace of disease progression may be given different medication combinations. It should also be noted that the best possible solution may still have very severe side effects. It is ultimately the decision of the clinician which treatment to adopt given the information available.

9.2.1 Example

Guidelines published by NICE are usually given in a combination of graphical representation and notes in natural language. There are several problems with some of the diagrams from NICE in their use of ambiguous notation. As mentioned earlier, we do not focus on modelling in this paper, and we present an abstract example, illustrating the guidelines for three hypothetical conditions `D1`, `D2` and `D3`, given as three labelled event structures directly. The formal details of the model will be described later, but the visualisation of the model is simple to describe. It allows us to give a more compact presentation, which is adequate for our purposes here.

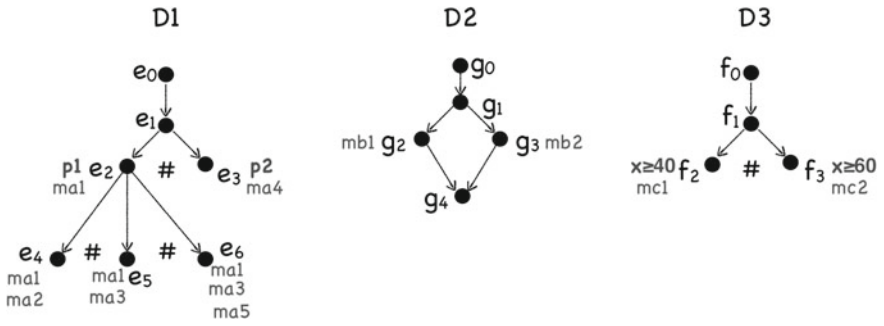

Fig. 9.1 Partial guideline models for three chronic conditions

Figure 9.1 shows three (unfoldings of) treatments for different conditions that a patient may be undergoing. Each circle is an event denoting the occurrence of something (an action, a clinical examination, taking a medication, etc.). The initial events (e_0, g_0 and f_0) indicate the diagnosis of the corresponding disease. At times, there may be a choice between treatment options (e.g. e_2 and e_3). We indicate in red the occurrence associated to an event. Some occurrences have conditions on them, for instance `p1` has to hold for e_2 to be able to occur. For event f_2, variable x has to have a value greater or equal to 40 for medication `mc1` to be prescribed. The arrows suggest the ordering of event occurrences, and the # denotes alternatives (e.g. event e_2 may be picked or event e_3 but not both). Alternative events e_2 and e_3 have associated constraints `p1` and `p2`, respectively, but note that they are not necessarily mutually exclusive. For instance, `p1` can correspond to `no patient weight restrictions` and `p2` can be `the patient is not overweight or obese`, in case `ma1` denotes standard metformin and `ma4` denotes a sulphonylureas. We may want to associate a priority to `p1`, to indicate, for instance that if it holds we will want the corresponding operand to execute (instead of the second operand and regardless of whether `p2` holds or not). We will see in the next section how these notions get formalised.

Assume that we know that the occurrence of `ma1` conflicts with `mc2`, and `ma2` conflicts with `mb2`. This is not encoded directly in the LES of Fig. 9.1, but is domain knowledge contained elsewhere. In order to find our optimal paths, we need to know in addition how effective drugs are considered to be when used for a condition and reported side effects. To simplify, we assume here that a drug is only used in the context of one treatment—which in a way can be inferred by the dosage—but this is not a required restriction of our framework. This information is captured for our example in Table 9.1.

In addition, drugs are known to interact with others. Sometimes additional drugs are added to compensate the interactions as shown in Table 9.2.

We want to combine the diagrams of Fig. 9.1 in a way that the known underlying conflicts are taken into account. To do so, we extend our approach from Bowles and Caminati [6] to find valid paths in the composition that avoid given conflicts and/or side effects as desired.

Table 9.1 Drug effectiveness and side effects

Drug	Effectiveness	Side effects	Likelihood
ma1	ve_1 1000	s0	Rare ($\leq$20%)
ma4	ve_4 900	s1 s2	Common ($\geq$60%) Rare ($\leq$20%)
ma5	ve_5 600	s3	Very common ($\geq$80%)

Table 9.2 Drug interactions

Drugs	Conflict level	Score
ma1, mc2	Severe	$v1$ - 2000
ma1, ma5, mc2	Mild	$v2$ - 600
ma2, mb2	Severe	$v3$ - 1800

9.3 Formal Model

The model we use to capture the semantics of a clinical guideline, or its unfolding, is a labelled (prime) event structure [32]. The choice of the model is merely based on its simplicity and how it is able to convey in a straightforward manner the key notions required: sequence, parallelism and repetition of behaviour (cf. Küster-Filipe [21], Bowles [4]). In addition, the formalism can be captured by our theorem prover and all models checked for correctness.

As the name suggests, event structures consist of sets of events with several binary relations defined over the events. Different variants of event structures are available and define different relations. A prime event structure defines two binary relations: *causality* (to denote a causal dependency between events) and *conflict* (to denote nondeterminism). The former induces a (partial) order among events (e.g. a patient whose diabetes worsens may be given a dual-therapy, that is an additional drug to the one taken previously), whereas the latter captures how the occurrence of some events excludes the occurrence of others (e.g. a patient may be given a choice of drug as a beta-blocker). A further implicit relation of *concurrency* captures any two events not related by causality nor conflict (e.g. a patient with diabetes is independently monitored for high blood pressure). The formal definition below is taken from Küster-Filipe [21].

Definition 1 An *event structure* is a triple $E = (Ev, \rightarrow^*, \#)$ where Ev is a set of events and $\rightarrow^*, \# \subseteq Ev \times Ev$ are binary relations called *causality* and *conflict*, respectively. Causality $\rightarrow^*$ is a partial order. Conflict $\#$ is symmetric and irreflexive, and propagates over causality, i.e. $e\#e' \wedge e' \rightarrow^* e'' \Rightarrow e\#e''$ for all $e, e', e'' \in Ev$. Two events $e, e' \in Ev$ are *concurrent*, $e \; co \; e'$ iff $\neg(e \rightarrow^* e' \vee e' \rightarrow^* e \vee e\#e')$. $C \subseteq Ev$ is a *configuration* iff (1) C is *conflict-free*: $\forall e, e' \in C \neg(\, e\#e')$ and (2) *downward-closed*: $e \in C$ and $e' \rightarrow^* e$ implies $e' \in C$.

We assume a *discrete* structure which guarantees a finite model and is sufficient for our purposes, that is, there are always only a finite number of causally related

predecessors to an event e. This is referred to as the *local configuration* of e and written $\downarrow e$. Discreteness is important here because in our treatment plans for a chronic condition there is always a starting point given by its diagnosis. A configuration as defined above (downward-closed and conflict-free) is a *trace of execution* if and only if it is maximal. Finally, an event e may have an *immediate successor* e' with respect to the causality relation. Immediate causality between two events e and e' is written $e \rightarrow e'$ and indicates that no other event can occur in between. An event can have one or more immediate successors.

Event structures are typically enriched with labels. We define two labelling functions below where L is a set of labels.

Definition 2 A *labelled event structure* over L is a triple $M = (E, \mu, \nu)$ where $E = (Ev, \rightarrow^*, \#)$ is an event structure, and μ and ν are partial labelling functions given by $\mu : Ev \rightarrow 2^L$ and $\nu : Ev \rightarrow \mathbb{N} \times \mathbb{N}$.

The first labelling function, $\mu : Ev \rightarrow 2^L$, maps each event onto a subset of elements of L, where L denotes constraints defined over integer variables (e.g. $x > 2$ or $y = 10$), logical propositions (e.g. `prop1`) or actions (e.g. prescribe a medication $ma1$). If for an event $e \in Ev$, $\mu(e)$ contains an action, then e denotes the occurrence of that action. If $\mu(e)$ contains a formula or logical proposition, then this formula or proposition must hold when e occurs.

The second labelling function, $\nu : Ev \rightarrow \mathbb{N} \times \mathbb{N}$, associates to each event its *priority* and *duration*. For an event e with $\nu(e) = (p, d)$, the highest the value of p the higher the priority associated to e, and d indicates the duration of e. Events in conflict (alternatives) should typically have different priority values. Further labels may be added to the framework as partial functions if required. We refer to M as a model.

Let $\mathscr{L}$ be a set of labels used across a finite number of models $M_1, \ldots, M_n$, where $n \in \mathbb{N}$ and $\mathscr{L} \supseteq \bigcup_{i=1}^{n} L_i$.

Definition 3 Label conflicts associated to $\mathscr{L}$ are given by $\Gamma \subseteq L_{i_1} \times \cdots \times L_{i_p} \times \mathbb{Z}$ where $i_1 \ldots i_p \in [1..n]$ and such that Γ is right unique, i.e. for any $l_1, l_2, \ldots, l_p \in \mathscr{L}$, if $(l_1, l_2, \ldots, l_p, m), (l_1, l_2, \ldots, l_p, k) \in \Gamma$ then necessarily $m = k$.

Here, we assume conflicts of a certain value. For instance, (l_1, l_2, v) indicates that l_1 and l_2 are in conflict with an *interaction score* of value v. The set of tuples in Γ encodes the information of Table 9.2.

Recall the example of Fig. 9.1 introduced in the previous section. The label conflicts are given by

$$\Gamma = \{(ma1, mc2, -2000), (ma2, mb2, -1800), (ma1, ma5, mc2, -600)\}$$

The information shown visually is formally given by the labels as follows: $\mu_{d1}(e_2) = \{p1, ma1\}$, $\mu_{d1}(e_3) = \{p2, ma4\}$, $\mu_{d1}(e_4) = \{ma1, ma2\}$, $\mu_{d1}(e_5) = \{ma1, ma3\}$ and $\mu_{d1}(e_6) = \{ma1, ma3, ma5\}$for the event structure associated to `d1`; $\mu_{d2}(g_2) = \{mb1\}$ and $\mu_{d2}(g_3) = \{mb2\}$ associated to `d2`; and $\mu_{d3}(f_2) = \{x \geq 40, mc1\}$ and $\mu_{d3}(f_3) = \{x \geq 60, mc2\}$ associated to `d3`.

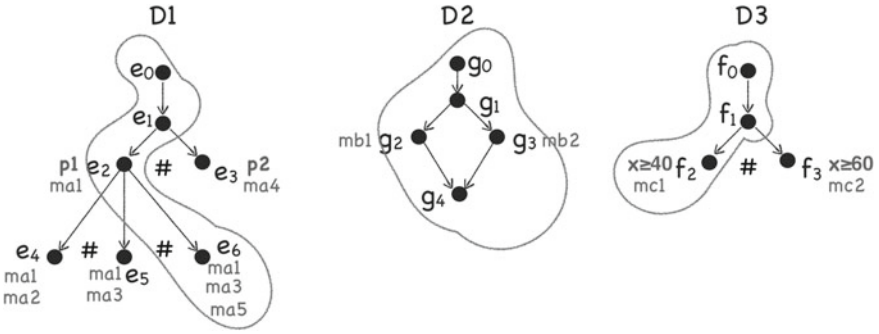

Fig. 9.2 Optimal solution with respect to effectiveness and interaction scores

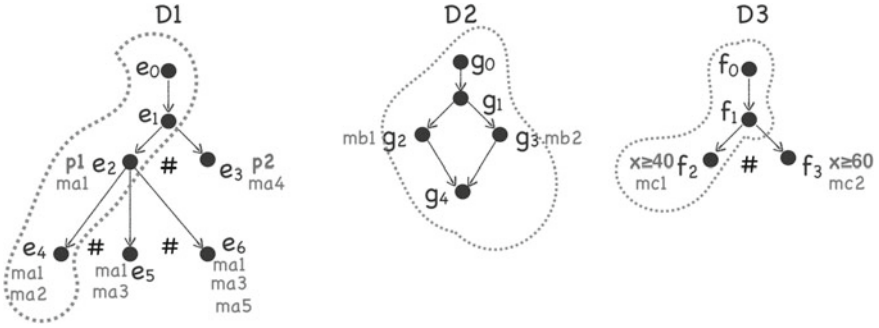

Fig. 9.3 Optimal solution with respect to effectiveness and interaction scores, which avoids side effect `s3`

The labels of some of the events (marked) above are inconsistent/conflicting according to Γ, namely events f_3 conflicts with e_2, e_4, e_5 and e_6; and events e_4 and g_3. When obtaining the composition of the models above we need to make sure label inconsistencies are detected and avoided. A composed model that avoids the labels could reduce the composition to a trace of execution which goes to e_3 and hence avoids most of the conflicts. However, if the search criteria needs to avoid a common side effect `s1` this is not an option.

When searching for the most effective configuration in the event structures, we get the configuration highlighted in Fig. 9.2. However, if we now consider that we want to avoid side effect `s3` (very common for `ma5`) we obtain the configuration shown in Fig. 9.3 instead.

9.4 Searching Optimal Solutions

We use the theorem prover Isabelle [24] and the SMT solver Z3 [23] for our purposes when searching for optimised treatment paths. We start with a description of how we use the theorem prover.

9.4.1 Using a Theorem Prover

Isabelle is a theorem prover (proof assistant) providing a framework to accommodate logical systems (inference rules, axioms), and compute the validity of logical deductions according to the chosen logical system. In this paper, we use Isabelle's library based on *higher-order logic* (HOL): the resulting overall system is referred to as Isabelle/HOL, but here we will use Isabelle and Isabelle/HOL interchangeably. In Isabelle/HOL, the basic notions are type specification, function application, lambda abstraction and equality. Using these notions, mathematical definitions can be formulated; in turn, theorems about these definitions can be proved using the axioms describing the intuitive properties of the basic notions and the inference rules of HOL. An important point is that an Isabelle/HOL definition can be computed if suitably formulated: this allows us to use Isabelle/HOL for performing computations and formally prove their correctness by verifying theorems stating the wanted properties of the corresponding definitions.

This idea is general, and can, in theory, be used to verify any algorithm. In practice, however, the definition of the object we want to compute is often *non-constructive* and therefore, whilst we can still use Isabelle/HOL to prove theorems about it, we cannot directly compute it. One general approach used in Caminati et al. [14] to overcome this limitation is: to keep the given non-constructive definition specifying the given computational problem, to add a *computable* definition, and to prove in Isabelle that they are equivalent through a so-called *bridging theorem*. In this way, any theorem we prove about one of the two will carry over to the other definition and, in particular, we are able to prove the correctness of our implementation (given by the constructive Isabelle definition) with respect to the specification (given by the original, potentially non-constructive, Isabelle definition).

These general considerations can be applied to the problem we are addressing in this paper. Given the description of our formal model in Sect. 9.3, it is clear that a key step in giving a solution to our problem consists of computing the traces of the underlying event structures, which entails the computation of all possible configurations. To this end, given an event structure with causality `cau` and conflict `cfl`, we can easily express the properties which make a set of events `C` a configuration.

```
abbreviation "isDownwardClosed cau C == (C ⊆ events cau
& (∀ e f. e ∈ cau & (f, e) ∈ cau → f ∈ C))"
abbreviation "isConflictFree cfl C == (∀ e e'.
e ∈ C & e' ∈ C → (e,e') ∉ cfl)",
```

where `cau` is the set of all the ordered pairs related by $\rightarrow^*$, and `cfl` is the set of all the ordered pairs related by #.

The problem, however, is that such definitions are not constructive: they describe the properties `C` must have, but not how to compute it. We mentioned that the general solution is to introduce a computable definition, which in this case reads:

```
abbreviation "extension cau C == (C ∪ (cau^{−1}‘‘C))"
abbreviation "restriction cfl C == C − (cfl ‘‘ C)"
abbreviation "configurations cau cfl ==
{C. C ∈ Pow (events cau) & extension cau C ⊆ C & C
    ⊆ restriction cfl C}",
```

where “applies a relation to a set, `^−1` takes the converse of a relation, and `Pow` takes the powerset. The advantage of `configurations` is that it is constructive and can, therefore, be used to actually compute all the configurations. It is not immediate how `configurations` relates to the original definition: the following bridging theorem certifies their equivalence.

```
theorem "(C ∈ configurations cau cfl) ↔
(isConflictFree cfl C & isDownwardClosed cau C)"
```

The next step to the solution to the problem outlined in Sect. 9.3 is taking traces, i.e. those configurations which are maximal. Again, the notion of maximality of a set `X` in a family `XX` of sets:

```
(X ∈ XX & (∀ Y ∈ XX. Y ≠ X → ¬ X ⊆ Y))
```

is descriptive, rather than constructive, and again we supply an equivalent, constructive definition:

```
abbreviation "isMaximal XX X == ({} ∉ (λ Y. X−Y)
    ‘(XX−{X}))"
abbreviation "maximals XX== {X ∈ XX. isMaximal XX X}"
```

along with a bridging theorem:

```
theorem  "X ∈ maximals XX ↔
(X ∈ XX & (∀ Y ∈ XX. Y ≠ X → ¬ X ⊆ Y))"
```

Putting together these first two steps, we are able to compute all traces for a given event structure:

```
abbreviation "traces cau cfl ==
    maximals (configurations cau cfl)"
```

We proceed by further steps each implementing the notions outlined in Sect. 9.3: there, using the small example of Sect. 9.2.1, we explained our need to select, for each event structure, the configurations yielding no incompatibilities (according to Γ) and the preferred combination of priorities (given by the first component of ν, ν_1). To do that, we need to specify how the events in a trace for one event structure overlap timewise with the event in a trace for another event structure. This in turn implies sorting the event in each trace. However, sorting the events of any configuration of an event structure must not disrupt the partial order given by the corresponding causality relation. In other words, taken a list whose entries are the events in a trace, and any

two distinct elements `f`, `s` of such a list which are related by $f \rightarrow s$, the index of f must be smaller than the index of s:

```
(∀ f s. ((f,s) ∈ set G & f ∈ set l & s ∈ set l & f ≠ s)
  → the (findFirstIndex (λ x. x=f) l) <
  the (findFirstIndex (λx. x=s) l)),
```

where `findFirstIndex (λ x. x=e) l` returns the index of the first entry of the list `l` equal to `e`. Since in general such an entry could in some cases be non-existing (this is prevented in the particular clause above by the conditions on `f`, `s`), this function actually returns a value of type *optional*, which provides a special value `None` for these cases: the function `the` appearing above converts back this optional type to a natural number, as it should be since it describes an index. Once again, this condition does not allow to compute all the trace lists we need, so that we introduce a corresponding constructive definition:

```
abbreviation "isOrderPreserving G l ==
(None = (List.find (λ x. x=True)
[let m=findFirstIndex (λ x. x=f) l in
let n=findFirstIndex (λ x. x=s) l in
(m ≠ None & n ≠None & the m > the n ). (f, s) <- G]))"
```

and a bridging theorem granting us that `isOrderPreserving` does what expected:

```
theorem "(∀ f s.
 ((f,s) ∈ set G & f ∈ set l & s ∈ set l & f ≠s) →
 the (findFirstIndex (λ x. x=f) l) <
 the (findFirstIndex (λ x. x=s) l))
↔ (isOrderPreserving G l)".
```

Once we have all the lists representing the traces and respecting the underlying partial order given by causality, it is easy to calculate the temporal configurations of all the events occurring in one such list. This can be implemented in Isabelle as follows:

```
abbreviation "clocks dephasing durations l == map (op +
    dephasing)
[listsum (map durations (take i l)) . i<-[0..<size l]]",
```

where the function `clocks` takes a list l representing a sorted trace and return the list of the abscissas (i.e. the time at which they start) of the corresponding events. This is calculated by summing for each event the `durations` (i.e. the second component of the pair returned by the function ν introduced in Sect. 9.3) of the preceding events and by adding the dephasing of the event structure. Finally, having computed the temporal scope of each event in a sorted trace, and all the sorted traces as from the previous steps, pruning the combination of traces featuring conflicting events overlaps is straightforward. From the remaining traces, the overall priority is computed through the standard Isabelle function `listsum`, then allowing to use the function Isabelle `argmax` to pick the best combination of traces. We omit the details here,

and focus, in the rest of the paper, on the issue of the performance of the obtained implementation.

Up to now, the stress was on correctness, granted, as we have seen, by bridging theorems between specification and implementation definitions in the functional language Isabelle/HOL. Like any implementation, however, the one we have introduced is liable to optimisations: for example, the definition of `configurations` on page 253 starts from the powerset of all the events and proceed to refine it according to the properties `extension` and `restrictions`. Since the computation of the powerset is expensive, one could try to find an equivalent definition for `configuration` not using it, and then proceed to prove a further bridging theorem stating the correctness of the new definition. This extends the bridging theorem approach by introducing chains of equivalent definitions, each more and more efficient, linked by several bridging theorems, leading to a general approach to writing algorithms which are both efficient and formally proven correct [9].

Here, rather than following that approach, we proceed by introducing an alternative technique to improve performance by producing non-Isabelle code which is, however, still amenable to Isabelle proofs. This non-Isabelle code will be, more precisely, satisfiability modulo theories code. An SMT solver checks the satisfiability of a set of formulas, aka *assertions*, expressed in first-order logic and such that arithmetic operations and comparison are understood. Furthermore, additional relations and functions can be evaluated in order to make the problem satisfiable. The next section reformulates our problem into SMT terms, while Sect. 9.5 introduces a general technique to apply Isabelle correctness proof to the SMT code.

9.4.2 Using an SMT Solver

SMT code is typically not very readable due to the first-order logic underpinning it, which features a limited number of native notions and structures. For example, any computation involving elementary operations on sets (e.g. union, intersection, cartesian product, domain, range and so on) has to be rewritten because sets are not directly available in first-order logic and must be represented as predicates. Furthermore, to increase efficiency, usually a number of transformations are applied to the code making it even less readable: for example, a universally quantified assertion over a finite type is often replaced by multiple non-quantified assertions, each for one element of the type (quantifier elimination). Finally, SMT-LIB, the standard specifying a common language for SMT solvers [2], consistently employs polish notation, aggravating the problem on the readability.

Our solution to this expository problem is to write formulas close to the first-order logic language used by SMT solvers; for the sake of readability, however, we will employ some simplifications. In particular, we adopt infix notation instead of prefix notation, we use set-theoretical styling instead of predicates (e.g. writing $(a, b) \in R$ instead of $R\ a\ b$), we will use set-theoretical operations (e.g. union, intersection, cartesian product, domain, range, etc.) instead of the corresponding first-order logic

renditions, we will omit type specifications, and we will use the universal quantifier $\forall$ even when in the actual code it has been eliminated.

Recall from Sect. 9.3 that the input describing our problem is a n-tuple of models

$$M_1 := (E_1, \mu_1, \nu_1) \ldots M_n := (E_n, \mu_n, \nu_n)$$

where

$$E_1 = \left(Ev_1, \rightarrow_1^*, \#_1\right) \ldots E_n = \left(Ev_n, \rightarrow_n^*, \#_n\right)$$

are event structures, together with a set of label constraints Γ. Ev_i is the set of events, $\rightarrow_i^*$ is the causality relation (a partial order), and $\#_i$ is the conflict relation of the ith event structure, $L_i := \bigcup \text{Range}\,(\mu_i)$ is its set of labels, ν_i specifies the priority and duration associated to each event of E_i, and Γ describes inter-structure label conflicts. We assume that the sets of events are pairwise disjoint, and denote M_i's immediate causality relation by G_i. We define the overall immediate causality and conflict relations by the union of the respective relations of the models, that is

$$G := \bigcup_{i=1,\ldots,n} G_i \quad \text{and} \quad \# := \bigcup_{i=1,\ldots,n} \#_i$$

Given a relation R over a set Y and a set $X \subseteq Y$, we introduce the notation $R^{\rightarrow}(X)$ to denote the image of X through R.

We will now proceed in steps: first, we show how to compute traces, then we show how to use ν to obtain the preferred one, depending on the duration and priority assigned to single events.

9.4.3 Representing Traces of Execution

To represent a trace of execution, we need to know which events belong to the trace and in which order. We define a Boolean function isSelected over the set of all events.

This function is computed by the SMT solver, and we illustrate how it works for a given event structure Ev_i. Since traces are maximal configurations we need to first make sure that isSelected is conflict-free and downward-closed in accordance to Definition 1. This is expressed by the following two conditions.

The conflict-free condition is given by

$$\forall j, k \in Ev_i.\ \text{isSelected}\,(j) \wedge \text{isSelected}\,(k) \rightarrow \neg\,(j \# k)$$

where j and k are events and if both are selected they cannot be in conflict.

The downward-closed condition is given by

$$\forall j \in \text{Range}\,(G_i)\,.\ \text{isSelected}\,(j) \rightarrow \bigwedge_{k \in \left(G_i^{-1}\right)^{\rightarrow}\{j\}} \text{isSelected}\,(k)$$

where if j is selected so is k for all k predecessor of j.

Together, these two conditions express the definition of configuration as introduced in Definition 1, and can be translated straightforwardly into corresponding SMT-LIB assertions. This is no longer true for the notion of *maximal* configuration, which is needed to compute the traces of execution as introduced in Sect. 9.3. In fact, finding a maximal configuration implies quantifying over all configurations (that is, sets of events). Since the logic of SMT solvers is essentially first order, this means that one can quantify over the elements of sets, but not over sets themselves. For this reason, we cannot directly express maximality in SMT-LIB. The solution to this technical hurdle is to notice that, in this particular case, the definition of a configuration can be reformulated in an equivalent way which does not require quantification over higher level entities:

$$\forall z \in Ev_i \; \exists y \in Ev_i \qquad ((y \# z \wedge \text{isSelected}(y)) \vee ((y, z) \in G_i \wedge \neg \text{isSelected}(y))) \tag{9.1}$$

We will prove that the aforementioned equivalence holds, and that therefore we can use (9.1) in our SMT-LIB code to effectively compute traces, in Sect. 9.5. More precisely, we will prove that the set on which isSelected is true satisfies the definition of trace, and that any set of events obeying the definition of a trace must satisfy the assertions above. This means that the SMT code expressing those assertions implements an algorithm to compute traces. This algorithm will return a set of events, as required by the definition of a trace. However, we will see in Sect. 9.4.4 that, for our purposes, it will be useful to endow this set with an ordering induced by the causality partial orders $\rightarrow_1^* \ldots \rightarrow_n^*$.

More precisely, given a trace isSelected we consider its subset given by $Ev_i \cap$ isSelected (that is, the portion of the trace made of events of a given E_i, $i \in \{1, \ldots, n\}$); we want to sort this subset or, more formally, we want to obtain a total linear order s_i over it, in such a way that s_i respects $\rightarrow_i^*$. The first step, therefore, is to obtain $\rightarrow_i^*$ from G_i; to conform with the fact that it is more convenient to restrict to the ASCII character set when writing SMT code, we will indicate, here, $\rightarrow_i^*$ with P_i (just as we previously used G_i for $\rightarrow_i$). This means that we want to obtain the partial order relation P_i given its transitive reduction G_i or that, equivalently, we want to obtain the transitive–reflexive closure P_i of the covering relation G_i (see, e.g. [15, Sect. 2]).

The standard definition of transitive closure of a relation consists of considering the family of all the supersets of the given relation satisfying the definition of transitivity and then taking the minimal. This presents the same problem posed by the computation of a maximal configuration, because to do so we have to quantify over a set of sets. We solve it in the same manner: the parts of the definitions not involving minimality can be expressed directly in first-order logic. In this case, we require that P_i is a transitive and reflexive extension of G_i, which is straightforward:

$$\begin{aligned} \forall j,\ k.\ (j,k) \in G_i \rightarrow (j,k) \in P_i, \\ \forall j,\ k,\ l.\ (j,k) \in P_i \wedge (k,l) \in P_i \rightarrow (j,l) \in P_i, \\ \forall j \in Ev_i.\ (j,j) \in P_i, \\ \forall j,\ k.\ (j,k) \in P_i \wedge (k,j) \in P_i \rightarrow j = k. \end{aligned} \tag{9.2}$$

Now we want to impose that P_i is minimal among all the relations satisfying (9.2). The following expression reformulates the minimality condition without quantifying over relations:

$$\forall l,\ n.\ (l,n) \in P \rightarrow l = n \vee (l,n) \in G \vee \exists m.\ ((l,m) \in G \wedge (m,n) \in P), \tag{9.3}$$

where we set $P := \bigcup_{i=1}^{n} P_i$. The idea behind (9.3) is that, if P_i is minimal, any two events related by P_i will be joined by a finite path of events in which every event is related to its successor by G_i.

We recall now that we wanted P_i in order to produce a total linear order s_i respecting P_i. To this end, it is sufficient to ask for s_i to be an injective monotone map between the ordered sets Ev_i and $\{1 \ldots, |Ev_i|\}$ (where the latter set is ordered by the canonical order for natural numbers); this will automatically imply that s_i is surjective and that it induces a total linear order on Ev_i, which will be useful in Sect. 9.4.4.

All these requirement are translated into the following SMT formulas:

$$\forall j,\ k.\ (j,k) \in P_i \rightarrow s_i(j) \leq s_i(k), \tag{9.4}$$

$$\forall j,\ k \in Ev_i.\ j \neq k \rightarrow s_i(j) \neq s_i(k), \tag{9.5}$$

$$\forall j \in Ev_i.\ s_i(j) \geq 1 \tag{9.6}$$

$$\forall j \in Ev_i.\ s_i(j) \leq |Ev_i| \tag{9.7}$$

$$\forall j,\ k \in Ev_i.\ \text{isSelected}(j) \wedge \neg\,\text{isSelected}(k) \rightarrow s_i(j) < s_i(k). \tag{9.8}$$

(9.4) specifies that s_i is monotonic (i.e. order-preserving), (9.5) requires s_i to be injective, while (9.6) and (9.7) specify the range of s_i. Finally, (9.8) impose to sort non-selected events before the events being part of the trace, a feature which will be convenient for the purposes of Sect. 9.4.4.

Finally, we need to take into account the influence of side effects into the determination of a solution. To this end, we preliminarily define a function isAdministered defined on all the drugs, and impose that if isSelected is true for a given node, then isAdministered is true for all the drugs in that node. Then, we introduce two functions, both defined over the set of all side effects S: isOccurring (yielding a boolean) and likelihood (yielding a number). The latter function is given as input, while the former is determined by the SMT solver to be true for any side effect linked to a drug for which the function isAdministered yields true. This allows to impose constraints regarding side effects; for example, imposing that no 'very likely' side effect occurs:

$$\forall s \in S.\ \text{likelihood}(s) \geq 80 \ \rightarrow\ \neg\,\text{isOccurring}(s)$$

More elaborate constraints can be formulated, the possibilities being limited only by the expressiveness of SMT syntax. For example, one could require that at most one in a subset of side effects occurs, or that if a given side effect occurs, then another does not, etc.

9.4.4 Selecting the Best Traces

In Sect. 9.4.3, we introduced SMT assertions to obtain a trace for each Ev_i independently. Now we want to restrict those assertions in order to pick one trace for each Ev_i so as to minimise the possible negative effects arising from interactions between events in distinct Ev's overlapping in time. To achieve this, we need to determine the time occupied by each event in a trace: this is the reason why, in Sect. 9.4.3, rather than limiting ourselves to compute traces of Ev_i merely as sets $Ev_i \cap \text{isSelected}$ of events of Ev_i, we additionally endowed those sets with a linear order (described by the map s_i). Indeed, this now allows a straightforward determination of a function clock yielding, for each event in a trace, its starting time: we just impose that each event starts when its predecessor is in the same trace (according to s_i) finishes.

$$\forall j,\ k \in Ev_i$$
$$(\text{isSelected}\ j \wedge \text{isSelected}\, k \wedge s_i\,(j) \leq |Ev_i| \wedge s_i\,(k) \leq |Ev_i| \wedge s_i\,(k) - s_i\,(j) = 1)$$
$$\rightarrow \text{clock}\,(k) = \text{clock}\,(j) + \nu_2\,(j)$$

Here, ν_2 is the second element of the pair returned by ν, as from the definitions in Sect. 9.3. For events having no predecessors, i.e. the sources of the directed acyclic graphs underlying the causality partial orders of the event structures, the formula above does not impose anything. In other words, we are left with the freedom of deciding when each event structure starts. This can be taken advantage of, as an example, for representing the addition of a new long-term condition to others already affecting a patient: the dephasing concept mentioned beforehand in Sect. 9.2. Now, we are ready to use the information from clock to determine a two-argument function Score yielding the degree of interaction of a pair of events and taking into account their temporal overlapping.

The idea is to combine the absolute interaction (i.e. irrespective of their time location) between two events, as given by a known map D, with their mutual position in time (given by clock) to determine the value of Score for that particular pair of events. The way we combine these two pieces of information is given by a known function f, which has the role of a parameter; in other words, it can be any function specified by the user in SMT-LIB, and therefore is quite arbitrary:

$$\forall j \in Ev_m,\ k \in Ev_{m'}.\, \text{isSelected}\,(j) \wedge \text{isSelected}\,(k) \rightarrow \text{Score}\,(j,k) =$$
$$f\,(\text{clock}\,(j), \text{clock}\,(k), \nu_2\,(j), D\,(\mu\,(j), \mu\,(k)))\,,$$

The formula above is replicated for each m, m' indexing distinct event structures; i.e. $m \in \{1, \ldots, n\}$, $m' \in \{1, \ldots, n\} \setminus \{m\}$. For the examples in this paper, we used a simple threshold function as f:

$$f\,(x_1, x_2, y, z) := \begin{cases} z, & \text{if } x_2 - x_1 \in [0, y] \\ 0, & \text{otherwise.} \end{cases}$$

Score can be used to guide the solver towards selecting traces so that their mutual interaction is minimised. However, we also want the priority of the events in the computed traces to be high: therefore, we need to combine these two criteria. To do so, we first need to compute the overall priority score of a trace:

$$\begin{aligned} \forall j.\, \text{isSelected}\,(j) &\rightarrow \text{priority}\,(j) = \nu_1\,(j) \\ \forall j.\, \neg\, \text{isSelected}\,(j) &\rightarrow \text{priority}\,(j) = 0, \end{aligned}$$

where ν_1 is the first component of ν, yielding the priority.

Then, we need to combine the two criteria; in other words, we need to combine Score and priority thus computed in some way. This way, similarly to what happens for f, can be given by any known, arbitrary map. In this paper, we used a very simple function; i.e. we merely added them into an integer variable `totScore`. The SMT solver was then challenged to take, among all the traces satisfying all the assertions, the one maximising `totScore`. Note that this requires an SMT solver supporting maximisation (which is outside of the SMT-LIB standard), such as Z3 [3].

9.4.5 The Example Revisited

We test the output of our approach with respect to the simple example of Fig. 9.1. To this end, we first check that our approach computes the expected solution depicted in Fig. 9.2 for the example introduced in Sect. 9.2.1.

Table 9.3 (left) shows that this is indeed the case, while Table 9.3 (right) was obtained from the same input, but with the additional requirement that *very likely* side effects (in this case, `s3`, do not occur). The result matches with what is represented in Fig. 9.3 and was discussed in Sect. 9.2.1.

9.5 Verification

In Sect. 9.4.2, we condensed the actual SMT code under higher level formulas at the theorem prover level motivating the reason for doing so. We now return to the topic of the expressiveness of the SMT code, noting that our actual SMT code consists of

Table 9.3 Computation results for Figs. 9.2 and 9.3 on the left and right, respectively

Clock	Event	Order	Duration	Clock	Event	Order	Duration
0	e0	1	1	0	e0	1	1
0	f0	1	1	0	f0	1	1
0	g0	1	1	0	g0	1	1
1	e1	2	1	1	e1	2	1
1	f1	2	1	1	f1	2	1
1	g1	2	1	1	g1	2	1
2	e2	3	3	2	e2	3	3
2	f2	3	3	2	f2	3	3
2	g2	3	2	2	g2	3	2
4	g3	4	1	4	g3	4	1
5	e6	7	3	5	e4	6	2
5	g4	5	4	5	g4	5	4

265 assertions for the example of Sect. 9.2.1. To illustrate the SMT code consider the following small excerpt.

```
(assert (=> (not (isSelected f1)) (exists ((y nodeType))
 (or (and (domain_coverRelations/f.txt y) (isSelected y)
       (< (interactionDatabase (label y) (label f1)) 0))
     (and (immediate_coverRelations/f.txt y f1)
     (not (isSelected y)))))))

(assert (=> (not (isSelected f2)) (exists ((y nodeType))
 (or (and (domain_coverRelations/f.txt y) (isSelected y)
       (< (interactionDatabase (label y) (label f2)) 0))
     (and (immediate_coverRelations/f.txt y f2)
     (not (isSelected y)))))))

(assert (=> (not (isSelected f3)) (exists ((y nodeType))
 (or (and (domain_coverRelations/f.txt y) (isSelected y)
       (< (interactionDatabase (label y) (label f3)) 0))
     (and (immediate_coverRelations/f.txt y f3)
     (not (isSelected y)))))))

(define-fun DAG_coverRelations/f.txt () Bool
(and (= true (isSelected f0))
  (=> (isSelected f1) (and (isSelected f0)))
  (=> (isSelected f2) (and (isSelected f1) ))
  (=> (isSelected f3) (and (isSelected f1)))))

(declare-fun transitive_coverRelations/f.txt
      (nodeType nodeType) Bool)
(assert (forall ((arg0 nodeType) (arg1 nodeType))
     (=> (immediate_coverRelations/f.txt arg0 arg1)
         (transitive_coverRelations/f.txt arg0 arg1))))
```

We can compare the Isabelle definition of `traces` given in Sect. 9.4 with the code above, which is only a small part of the SMT code pursuing the same computation as `traces`. On the one hand, Isabelle exploits the expressiveness of higher-order logic for both computing and stating theorems, whereas on the other hand, SMT solvers can handle millions of variables and assertions very efficiently. Furthermore, in Isabelle's higher-order logic, one has (or can define) whatever mathematical notions and datatypes may be needed for the problem at hand, and can define functions of any order (such as functions operating on other functions, and so on). Conversely, an SMT solver has natively only very low-level objects: Booleans, numbers, quantifiers (for all, exists), functions and relations defined over these types, and maybe the possibility of defining further datatypes. However, the objects obtained by quantifying, function/relation application, and so on, cannot, in general, be subject themselves to further quantification, function/relation application. In essence, this is because such solvers operate on first-order logic only. We have seen the consequences of this restriction when facing the notion of maximality in Sect. 9.4.2.

Our approach proposes to combine both approaches and make use of the best from each. In other words, we address the following question: Is it possible to write efficient SMT code and to apply fully formal, rigorous Isabelle theorems on it, expressed using the powerful higher-order logic of the latter? This can be useful in concrete SMT implementations, where the efficient code can blow up to hundreds or thousands of little-readable assertions, and therefore the assurance, by a formal theorem, that what we are computing is indeed what we meant, becomes important. In this section, we introduce a general method to achieve this, articulated in the following ideal steps:

s1 Isabelle's HOL incorporates first-order logic; therefore, it is possible to generate SMT code from Isabelle definitions restricting to entities in HOL's first-order logic fragment.

s2 These 'restricted' definitions are still Isabelle objects: hence, we can formally prove correctness theorems about them (maybe by proving their equivalence to higher order or more expressive Isabelle definitions which are easier to be stated theorems about).

s3 Parallel to such Isabelle-generated SMT code, we will usually have efficient SMT code obtained in some other way.

s4 We can use the SMT solver to prove the equivalence of the Isabelle-generated SMT code and of the 'efficient' SMT code by challenging it to find a model satisfying the assertions specified by one but not the assertions specified by the other.

We will illustrate this method through a couple of examples related to the computational problems presented up to now.

Let us start from Definition 1, that of event structure: it is mainly built on elementary properties of relations (e.g. irreflexivity, transitivity, symmetry, propagation). In first-order logic, and therefore in an SMT solver, we do have relations available,

and therefore we can express the relevant requirements. However, in mathematics it is quite common to regard relations (and functions) as sets of pairs, which allows to reduce properties and operations on relations to set-theoretical properties and operations: indeed, in Isabelle, many of the properties we need (irreflexivity, transitivity, symmetry, etc.) are already defined using this second approach, with a number of useful theorems already provided in the Isabelle library using this representation. Therefore, a first Isabelle definition for Definition 1 is straightforward:

```
abbreviation "isLes causality conflict ==
propagation conflict causality & sym conflict &
irrefl conflict & trans causality &
antisym causality & reflex causality"
```

where `propagation` was the only auxiliary definition we had to introduce ourselves, the others appearing in the definiens above being already available:

```
abbreviation "propagation conflict causality ==
∀ x y. (x,y) ∈ causality  → conflict ``{x} ⊆ conflict ``{y}"
```

Typically, the set-theoretical representation will be more convenient than the other for proving theorems involving the definition of event structure, but, of course, Isabelle can use both representations and pass easily from one to another: this means that any theorem proved about one of the definition can be restated for the other, by applying the following Isabelle theorem to the target Isabelle theorem:

```
theorem "IsLes causality conflict ↔
    (isLes (pred2set Causality) (pred2set Conflict))"
```

where `pred2set` converts from relations represented as predicates into relations represented as sets, and the definition of `IsLes` is similar to that of `isLes`, but with native relations, rather than relations as sets of pairs. However, an SMT solver can only use the non-set-theoretical representation, so that we can generate SMT code expressing Definition 1 using `IsLes`, because it restricts to those higher order notions which are also available in first-order logic. We did not even need to write our own translator from Isabelle syntax to SMT-LIB syntax, because it is already provided by the existing Isabelle tool called *sledgehammer* (although the latter uses the translator for goals totally different than ours):

```
lemma assumes "IsLes Causality Conflict" shows False
sledgehammer run [provers=z3, minimize=false,
          overlord=true, timeout=1] (assms)
```

Sledgehammer invokes SMT solvers to automatically find proofs to given lemmas (a goal we are not interested in, here), and to do so the first step is translating the lemma statement into SMT-LIB: since we only care about the first step, we placed the definition we want to translate as a premise of a dummy lemma (keyword `assumes`), providing a dummy thesis (`shows False`), and then invoked sledgehammer. The SMT code obtained is directly usable by an SMT solver (because `IsLes` has been

carefully restricted to first-order logic, as discussed above). We can, therefore, use it to test whether a given SMT solution or definition is indeed an event structure. This grants that all the Isabelle theorems we proved about event structures automatically apply to that piece of SMT code (for example, Isabelle theorems seen in Sect. 9.4).

As a more elaborate application of the method introduced in this section, we show how we can prove the correctness of the SMT code underlying the formula (9.1) (Sect. 9.4.3). That formula will result in a multitude of SMT assertions, corresponding to the fact that i and z range over the appropriate sets. The correspondence of this portion of SMT code with the original goal (that is, calculating maximal configurations) is hard to spot upon code inspection and even harder to establish with acceptable certainty. Although the reformulation in (9.1) of the concept of maximality gives rise to such problems, we cannot avoid it, because the original definition of maximality is not expressible in first-order logic (we discussed this problem in Sect. 9.4.3). This kind of situation is pretty common when using SMT or SAT solvers: the object needed to be computed is often obtained through assertions possibly not quite resembling the original definition of that object, typically because that definition is either inefficient or not expressible in first-order logic. Therefore, we proceed according to the ideal steps outlined at the beginning of this section and start by introducing Isabelle definitions which are close to pen-and-paper definitions and therefore easier to state theorems about (step (**s2**)). We need the definitions of configuration and of maximality. The first is given straightforwardly by the following three Isabelle definitions:

```
abbreviation "isConflictFree Cf C ==
    (∀ e e'. e ∈ C & e' ∈ C → (e,e') ∉ Cf)"
```

```
abbreviation "isDownwardClosed Ca C =
    (C ⊆ events Ca &
    (∀ e f. e ∈ C & (f, e) ∈ Ca → f ∈ C))"
```

```
abbreviation "isConfiguration Ca Cf C =
  isConflictFree Cf C & isDownwardClosed Ca C"
```

The first two are restated from Sect. 9.4.2 for the reader's convenience. We add another straightforward Isabelle statement of the notion of maximality to obtain an Isabelle definition of trace as introduced in Sect. 9.3:

```
abbreviation "isTrace Ca Cf C =
  isConfiguration Ca Cf C &
  (∀ Y. Y ⊃ C → ¬ (isConfiguration Ca Cf Y))"
```

where the last line says that any strict superset `Y` of a trace `C` must not be a configuration, which is what one means by maximality of `C`. Now, we want to prove that this simple definition of maximality is the same as that expressed by (9.1). Therefore, we reproduce the latter in Isabelle as follows.

```
abbreviation "isMaximalConfSmt Ca Cf C ==
  (∀ z ∈ events Ca − C.
     z ∈ Cf``C ∨ (immediatePredecessors Ca {z})−C ≠ {})"
```

where `immediatePredecessors Ca {z` returns all the events e satisfying $e \rightarrow z$ (we recall that $\rightarrow$ is the immediate causality obtained from $\rightarrow^*$). This allows to link the two definitions through the following Isabelle theorem (we omit the formal proof here):

```
theorem correctness: assumes "finite Ca" "isLes Ca Cf"
 "isConfiguration Ca Cf C" shows
 "(isTrace Ca Cf C) ↔ isMaximalConfSmt Ca Cf C"
```

The theorem establishes in its thesis (which is the statement following the keyword `shows`) that the two definitions are indeed equivalent, as soon as some obvious hypotheses are satisfied. One of these hypotheses requires that the set `C` is actually a configuration. This does not pose issues because, contrary to maximality, configuration can be straightforwardly expressed in first-order logic.

Above, `isMaximalConfSmt` is restricted to first-order logic, and can therefore generate SMT code through `sledgehammer`, as discussed above for `IsLes`. It is the gateway definition allowing us to bring correctness proof into SMT code. This does not mean, however, that the corresponding SMT code will be the one actually used for computations. The actual code is carefully written to optimise performance, for example through the elimination of universal quantifiers (`forall`) by writing one assertion for each possible quantified value, or through rewriting assertions in equivalent forms, etc. We now show how the correctness of the Isabelle-generated SMT code can be brought over to the SMT code actually used for computations. We start by collecting the portion of the latter which expresses maximality under an SMT boolean function `maximality`. If `maximality` were not correct, there would be some model satisfying exactly one between `isMaximalConfSmt` and `maximality`. This possibility is excluded by the SMT solver returning `(unsat)` for the following assertion:

```
(assert (or (and (not maximality) isMaximalConfSmt)
        (and maximality (not isMaximalConfSmt))))
```

9.6 Conclusions

We presented a powerful approach to search for optimal treatment plans, and hence reduce the risk of adverse drug reactions for patients with multiple chronic conditions, under different criteria using SMT solvers. To the best of our knowledge, the use of SMT solvers in this setting and for our purpose is novel. For the purposes of this paper we kept the presentation of the notions and approach abstract, but clearly motivating the need of all concepts with clinical guidelines in mind. In future work, we want to analyse real data to obtain models for the treatment of individual diseases

that takes into account medical practice, and determine more accurate values for our parameters. Only then will be able to evaluate the framework with clinicians and realistic scenarios.

References

1. Araújo J, Whittle J, Kim D (2004) Modeling and composing scenario-based requirements with aspects. In: RE 2004. IEEE Computer Society Press, pp 58–67
2. Barrett C, Stump A, Tinelli C (2010) The SMT-LIB standard: version 2.0. In: Gupta A, Kroening D (eds) Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)
3. Bjørner N, Phan AD, Fleckenstein L (2015) νz-an optimizing SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 194–199
4. Bowles J (2006) Decomposing interactions. In: Johnson M, Vene V (eds) Algebraic methodology and software technology: 11th international conference, Kuressaare, Estonia, 5–8 July 2006. Lecture notes in computer science, vol 4019. Springer, Berlin, pp 189–203
5. Bowles J, Bordbar B (2007) A formal model for integrating multiple views. In: ACSD 2007. IEEE Computer Society Press, pp 71–79
6. Bowles J, Caminati M (2016) Mind the gap: addressing behavioural inconsistencies with formal methods. In: 23rd Asia-Pacific software engineering conference (APSEC). IEEE Computer Society, pp 313–320
7. Bowles J, Caminati M (2017) Correct composition of dephased behavioural models. In: Proença J, Lumpe M (eds) Formal aspects of component software (FACS 2017). Lecture notes in computer science, vol 10487. Springer, Berlin, pp 233–250
8. Bowles J, Caminati M (2017) A flexible approach for finding optimal paths with minimal conflicts. In: International conference on formal engineering methods. Lecture notes in computer science, vol 10610. Springer, Berlin, pp 209–225
9. Bowles J, Caminati M (2017) A verified algorithm enumerating event structures. In: Geuvers H, England M, Hasan O, Rabe F, Teschke O (eds) Intelligent computer mathematics (CICM 2017). Lecture notes in computer science, vol 10383. Springer, Berlin, pp 239–254
10. Bowles J, Alwanain M, Bordbar B, Chen Y (2015) Matching and merging scenarios automatically with Alloy. In: Hammoudi S et al (eds) Model-driven engineering and software development. Communications in computer and information science, vol 506. Springer, Berlin, pp 100–116
11. Bowles J, Bordbar B, Alwanain M (2015) A logical approach for behavioural composition of scenario-based models. In: Butler M, Conchon S, Zaïdi F (eds) Formal methods and software engineering: 17th international conference on formal engineering methods. Lecture notes in computer science, vol 9407. Springer, Berlin, pp 252–269
12. Bowles J, Bordbar B, Alwanain M (2016) Weaving true-concurrent aspects using constraint solvers. In: Application of concurrency to system design (ACSD 2016). IEEE Computer Society Press, pp 35–44
13. Bowles J, Caminati MB, Cha S (2017) An integrated framework for verifying multiple care pathways. In: 2017 international symposium on theoretical aspects of software engineering (TASE). IEEE Computer Society Press, pp 1–8
14. Caminati M, Kerber M, Lange C, Rowat C (2015) Sound auction specification and implementation. In: Proceedings of the 16th ACM conference on economics and computation, ACM EC '15, pp 547–564
15. d'Amore F, Franciosa P, Giaccio R, Talamo M (1997) Maintaining maxima under boundary updates. In: Italian conference on algorithms and complexity. Lecture notes in computer science, vol 1203. Springer, Berlin, pp 100–109

16. Dijkman RM, Dumas M, Ouyang C (2008) Semantics and analysis of business process models in BPMN. Inf Softw Technol 50(12):1281–1294
17. Hughes L, McMurdo MET, Guthrie B (2013) Guidelines for people not for diseases: the challenges of applying UK clinical guidelines to people with multimorbidity. Age Ageing 42:62–69
18. Jackson D (2006) Software abstractions: logic, language and analysis. MIT Press, Cambridge
19. Klein J, Hélouët L, Jézéquel J (2006) Semantic-based weaving of scenarios. In: AOSD'06. ACM, pp 27–38
20. Kovalov A, Bowles J (2016) Avoiding medication conflicts for patients with multimorbidities. In: 12th international conference on integrated formal methods. Lecture Notes in Computer Science, vol 9681. Springer, Berlin, pp 376–392
21. Küster-Filipe J (2006) Modelling concurrent interactions. Theor Comput Sci 351:203–220
22. Liang H, Diskin Z, Dingel J, Posse E (2008) A general approach for scenario integration. In: MoDELS 2008. Lecture notes in computer science, vol 5301. Springer, Berlin, pp 204–218
23. Moura LD, Bjørner N (2008) Z3: an efficient SMT solver. In: TACAS 2008. Lecture Notes in Computer Science, vol 4963. Springer, Berlin, pp 337–340
24. Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL – a proof assistant for higher-order logic. Lecture notes in computer science, vol 2283. Springer, Berlin
25. OMG (2011) Business process model and notation. Version 2.0. OMG. http://www.omg.org, Document ID: formal/2011-01-03
26. OMG (2011) UML: superstructure. Version 2.4.1. OMG. http://www.omg.org, Document ID: formal/2011-08-06
27. Reddy R, Solberg A, France R, Ghosh S (2006) Composing sequence models using tags. In: Proceedings of the MoDELS workshop on aspect oriented modeling
28. Reisig W (1985) Petri nets. EATCS monograph, vol 4. Springer, Berlin
29. Rubin J, Chechik M, Easterbrook S (2008) Declarative approach for model composition. In: MiSE 2008. ACM, pp 7–14
30. Whittle J, Araújo J, Moreira A (2006) Composing aspect models with graph transformations. In: Proceedings of the 2006 international workshop on early aspects at ICSE. ACM, pp 59–65
31. Widl M, Biere A, Brosch P, Egly U, Heule M, Kappel G, Seidl M, Tompits H (2013) Guided merging of sequence diagrams. In: SLE 2012. Lecture notes in computer science, vol 7745. Springer, Berlin, pp 164–183
32. Winskel G, Nielsen M (1995) Models for concurrency. In: Abramsky S, Gabbay D, Maibaum T (eds) Semantic modelling, vol 4. Handbook of logic in computer science. Oxford Science Publications, Oxford, pp 1–148
33. Zhang D, Li S, Liu X (2009) An approach for model composition and verification. In: NCM 2009. IEEE Computer Society Press, pp 1102–1107