Towards a Flexible Approach for Understanding and Comparing Traces*

Juliana Küster Filipe Bowles[1,2][0000−0002−5918−9114]

[1] SCCH, Softwarepark 32a, 4232 Hagenberg, Austria
[2] School of Computer Science, University of St Andrews, St Andrews, KY16 9SX, UK
jkfb@st-andrews.ac.uk

Abstract. In this paper we present our vision towards a formal flexible approach to search for optimal traces of execution with respect to different parameters of interest (e.g., time, efficacy, cost and/or concurrency), enriched with a mechanism to understand the difference between (sub-optimal) traces and their formalisation as explanations. The underlying idea of the approach is to be generalisable and hence suited to tackle practical problems in a variety of domains from aerospace, automotive and automation industries, as well as systems and applications developed for healthcare. Our examples here come from a healthcare based context.

Keywords: Event Structures · Traces· Optimisation · SMT solvers

1 Introduction

Complex software systems have become the heart of the world's economy, society and infrastructure. Consequently, software vulnerabilities and errors in these systems can have substantial global impact and need resilient approaches and solutions able to tackle them.

Formal verification techniques, including model checking, can be used to analyse properties over such systems but require the existence of system models (labelled transition systems, variants of automata, Markov-based models, and so on) which can be checked against properties of interest written in a suitable temporal logic (LTL, CTL, timed/stochastic extensions, etc). Given a system model, model checking explores exhaustively all possible states in the model (either explicitly or implicitly) to find executions that violate certain constraints or conversely satisfy certain properties, and if the approach permits, within certain timed bounds and probabilities. Despite the benefits of using logic and formal techniques for finding bugs in the hardware and software industries, it remains challenging in practice: techniques often do not scale, and system behavioural models may be incomplete or poorly understood.

By contrast, there are many complex problems in modern systems that can be encoded in first-order logic and benefit from the use of efficient SMT solvers:

* This work has been supported by the Austrian Funding Council under FWF Meitner M-3338-N.

finding design errors in the logical functioning of modern digital electronic chips; solving scheduling and planning problems; validating cyber-physical systems; generating test inputs for safety-critical embedded software; detecting security vulnerabilities through an SMT-based technique called *Automatic Exploit Generation* [1]; amongst many others.

In this paper, we take the formalisation of (parts of) models of computation as sets of constraints given in first-order logic and enriched with parameters (e.g., an integer variable x representing a measure in the context of the computation), to search for executions that satisfy particular properties. More concretely, we use SMT solvers to search within models of computation for executions that optimise a given parameter (e.g., maximise x) whilst always satisfying a certain property (e.g., a boolean variable that must always be true). Our models of computation are abstractions that can be extracted from specifications, processes, code and/or data depending on the application. The flexible and scalable nature of SMT solvers such as Z3 [23] make them a suitable choice in practice.

We envisage an approach which reflects on how to understand the obtained optimal computation (trace of execution aka solution) given by the solver, whether there are several solutions that should be considered and whether a notion of *distance* can help to characterise and explain them, be used to select a preferred solution, and so on. This paper presents the background, context and a practical problem for our envisioned approach.

2 Background

A Boolean satisfiability problem, or SAT for short, is the problem of determining whether there is an assignment for all variables in a propositional formula such that the formula evaluates to true, that is, the formula is *satisfiable*. Many problems can be formulated in this way and are solved very efficiently by modern SAT solvers. Furthermore, modern SAT solvers have become core technology of many model checkers and are particularly suited to prove invariants as well as reachability properties through what is known as *bounded model checking* [4] and further enhanced through powerful approaches such as k-induction which, much like mathematical induction, use induction to increase the depth of the bound [14]. Efficient modern SAT solvers (and SMT solvers) are expected to continue to contribute to essential redevelopments of a new generation of model checkers which scale better and can handle infinite-state models.

Some problems require more expressive logics, such as first-order or higher-order logics, and include non-Boolean variables, functions and predicate symbols and quantifiers. To keep such problems decidable, it is usually necessary to constrain the interpretations of the functions and/or predicate symbols. This means that we assume some logical background *theory* (e.g., theory of equality, of integer numbers, of real numbers, and so on), and we are concerned with *satisfiability modulo theories* (SMT) [2, 3], that is, determining the satisfiability of a formula φ with respect to an interpretation in a given theory T. To determine the satisfiability of formulae, SMT solvers make use of efficient reasoning techniques

specific to the selected theory T. SMT solvers are particularly powerful because they can handle and combine many different kinds of theories, and can also be extended to include new theories to deal with new application domains. In addition, the standardisation efforts of SMT-LIB[3] have contributed to driving progress in SMT-based technology further.

Searching for traces of execution involves the use of a solver. Introducing non-Boolean variables, functions, predicate symbols and quantifiers, forces us to go beyond SAT solvers, and use SMT solvers for finding solutions which optimise parameters of interest. There is a wide range of possibilities for solvers that can be used to solve constraints involving non-Boolean variables, including, among others, Minion[16], FlatZinc and MiniZinc[25]. Modern SMT solvers such as Z3 allow the inclusion of heuristics also known as strategies [22]. In addition, when reasoning about the model of choice it can be useful to use a theorem prover Isabelle [27] to generate SMT-Lib code and link it to solvers such as Z3 and CVC5. For the underlying chosen model, we need to define notions of computation distance and explainability. Reasoning on those may be facilitated through the use of a theorem prover.

For a given abstraction from code, we may want to find executions that are computationally more efficient, have the highest degree of concurrency or consume less energy, using criteria such as performance, time, energy, efficacy, cost, and so on. We assume that statements in code may have annotated measures which can be used to search for execution runs that are optimal wrt those measures. The ability to evaluate and compare different optimal computations for different purposes can be used to gain further insights on the abstraction itself.

Let us consider the setting of improving treatment plans for patients with multiple chronic conditions where it is important to be certain that medications prescribed for different purposes do not lead to adverse reactions or undesired side effects: *is there a better combination of medications that avoids harmful adverse reactions and improves the quality of life of the patient?* Or even more fundamentally: *what is the difference between two solutions provided by a solver in terms of treatment efficacy or side effects?* It may be important to understand the difference between proposed treatments in cases where certain medications may not be available.

3 Approach

3.1 Overview

The vision of our approach is sketched in Fig. 1. For a given input, here seen as one or more models of computation (e.g., event structures), converted into an SMT encoding (step ①) and passed to an SMT solver with arithmetic optimisation such as Z3 (step ②), we obtain (if solutions exist) one optimal solution (computation or trace of execution) with respect to the parameters of interest, and through a number of calls to the SMT solver (step ③), this may lead to one or more solutions: sol1, sol2, sol3, etc. Our earlier work (cf. [8, 10, 9, 11]) offers

[3] `http://smtlib.org`

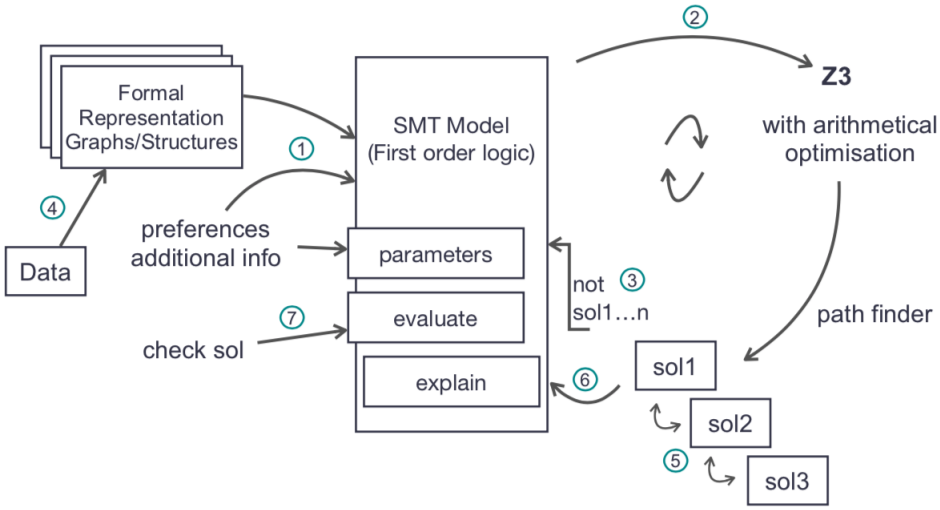

Fig. 1: Overview of the approach

partial contributions to steps ①, ② and ③, but needs to be extended to address preferences and additional information more accurately in step ①. In our earlier work, we have also used the theorem prover Isabelle [27] to generate SMT code automatically and provide additional aid in proving the correctness of properties as needed.

Challenges arise when there are no models available from which we can derive a formal representation or only partial information is available. Step ④ addresses this by enabling the inference of additional information from data, such as additional structures/paths, probabilities associated to different choices, etc. One option here is to follow a similar approach to CP-nets [5] where preferences are represented as a partial order with added priorities and optimisation criteria, but adapted to our richer event structures, and combine them with the remaining models when obtaining the SMT encoding (step ①). The advantage of our approach is that it will use the same abstract representation to capture different aspects and sources of information required for the search.

To create practical solutions of real value, we need to be able to explain generated solutions as well as their difference. Step ⑤ deals with the formalisation and understanding of differences between solutions, and how solutions are explained in step ⑥. Of added value is also to be able to evaluate how good a particular solution may be in the overall spectrum of possible solutions (step ⑦), making use of the notion of *solution distance* from step ⑤. Concerning step ⑥, we have explored the use of argumentation theory for explaining medication choices in[29, 30]. Instead we may take a simpler approach and focus on

the first-order logical statements associated to paths obtained and notions of computation distance.

3.2 Models of Computation

The model of computation we use is a labelled (prime) event structure [32], or *event structure* for short. Event structures have been widely used and studied in the literature, and have been used to give a true-concurrent semantics to process calculi such as CCS, CSP, SCCS and ACP (e.g., [31]). The advantages of prime event structures include their underlying simplicity and how they naturally describe fundamental notions present in behavioural models and code including sequential, parallel and iterative behaviour (or the unfoldings thereof) as well as nondeterminism (cf. [19, 6]), and are hence our model of choice. Event structures also have well-defined composition operators and can be used to represent unfoldings of graphs and models such as Petri nets (cf. e.g., [26]).

To use event structures as a model capturing the behaviour of a process, piece of code or component, we add labels to the events, associating the events to actions that have to be performed, conditions that have to be checked, statements that have to be executed, and so on. Further enriching an event structure with labelling functions gives us a mechanism to introduce further measures which can reflect properties of interest for our computations. However, existing composition mechanisms for event structures ignore labels and are hence inadequate for our use here. By converting our problem to an SMT problem, we circumvent this and we can search for the computations that satisfy our criteria without the need to create a composite model.

The formal definition of an event structure below is taken from [19].

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* $e' \in C$*.*

An event structure consists of a set of event occurrences together with binary relations for expressing causal dependency (called *causality*) and nondeterminism (called *conflict*). The causality relation implies a (partial) order among event occurrences, while the conflict relation expresses how the occurrence of certain events excludes the occurrence of others. From the two relations defined over the set of events, a further relation is derived, namely the *concurrency* relation *co*. Two events are concurrent if and only if they are completely unrelated, i.e., neither related by causality nor by conflict. 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 computations always have a starting point. Immediate

causality between two events e and e' is written $e \to e'$ and indicates that no other event can occur in between. An event can have one or more immediate successors. Finally, a configuration C as defined above (downwards-closed and conflict-free) is a *trace of execution* if and only if it is maximal.

Depending on what interpretation we want to give to an event structure, we define a set of labels, and a labelling function is a partial function that associates subsets of labels to events. We can define several labelling functions to add different measures and parameters of interest. We use LES to refer to labelled event structures in the following, and we assume that the labelling functions are defined for different concrete problems as needed.

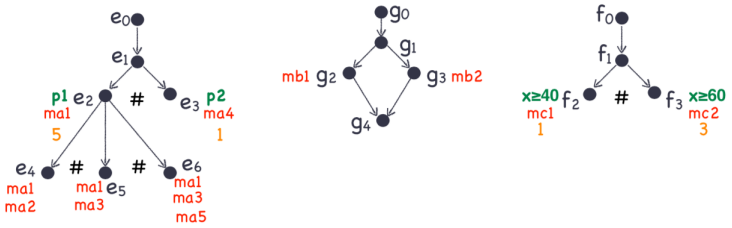

Fig. 2: Three event structures with labels shown.

Consider the three structures shown in Fig. 2 with events given as $e_0, e_1, \ldots$, $g_0, g_1, \ldots$, $f_0, f_1, \ldots$, explicit immediate causality shown (e.g., $e_0 \to e_1$), direct conflict between events at the same level (e.g., $e_2 \# e_3$), and so on. In this example events g_2 and g_3 are concurrent. Additional labels (given in different colours) are shown next to the events. Let p_1 and p_2 indicate conditions associated to events e_2 and e_3 respectively, where each condition has to hold for the corresponding event to occur; priorities are given next to events as natural numbers; and additional actions (ma_1, mb_1, etc) are shown as well. The search across these three very simple models can give us 8 possible traces of execution, but different traces may be required under different conditions. Furthermore, we may have additional information that tells us that certain actions should not be done together (e.g., ma_1 and mc_2), even though they are associated to events with higher priority.

Additional information, such as actions that should not be done together, preferences over the use of certain resources, probabilistic information contained in data, etc, can be added to the solver as well. To keep the overall formalism cohesive, we will represent this information as a LES. We will explore the models of preferences defined for CP-nets [5] for our context here, and how these can be adapted to LES. In CP-nets preferences are represented as a partial order with added priorities and optimisation criteria. Since our LES contains partial orders (in the binary relation causality), the approach taken will be a natural extension which needs to incorporate conflict and concurrency as well. The process of generating SMT encodings from our LES-preferences will hence be the same.

3.3 Computation Distance, Comparison and Evaluation

An SMT solver will generate one optimal solution with respect to the parameter we are trying to optimise. In some cases there may be more than one solution for the same optimal value. In these circumstances, it is worth to be able to characterise the *distance* between the different solutions/computations with the same optimal value, between an optimal solution and a second best, or between two solutions that optimise different parameters. In our case solutions are traces of execution across our input structures. One approach to consider for the distance between two solutions can be how many changes have to be made from one solution to reach the other. The notion of Levenshtein distance [20] for strings, which measures the number of changes between two strings required to make them equal, may be useful when comparing traces of executions. Since sequences of tasks representing a process can been seen as graphs, graph-based edit-distances-algorithms, which extend Levenshtein distance, may be more appropriate [15]. A further useful notion for our model is that of a CP-net distance as described in [21] which is based on distances between partial orders. We expect the later notion to be particularly relevant because event structure causality is a partial order. Whilst *causality* is one of the relations contained in event structures, we also need to consider conflict and concurrency as other relations that may have an impact on our possible distance metrics, as well as any information contained in the event labels (depending on which criteria is being used in the search). Indeed, several notions of computation distance may be necessary.

For instance, when observing the models in Fig. 2, the path that leads to e_5 is very similar to the path that leads to e_6, and the difference is contained in the labels: the choice of e_6 implies a further action ma_5 not taken in e_5. Further information is needed to understand and explain the difference that this additional action may entail.

We can capture the steps taken to go between solutions (traces of execution), and this will allow us to compare a given trace with an optimal solution obtained by the solver with respect to a given criteria, or for two solutions for different criteria to be compared. A further search could be used to get the sequence of steps to go between solutions, and this could be represented as a logical formula not just to formalise their distance but to explore how to justify their difference.

3.4 Explanations

In our case, the solver will return one or more possible solutions (if any exists) for different criteria we may wish to optimise. In addition to understanding why two traces may be different as given by the notion of distance, we also need to be able to explain a computation on its own which involves justifying the events in the trace as well as any inherent concurrency.

We presented an argumentation model to justify the choices on a clinical pathway, as identified by the SMT solver for multimorbid patients in [30]. While the solver only encoded drug information in the optimal path, we can consider all additional information available in our models (and formulated into our SMT

encoding in step ①) to justify an event in a path. Even though our earlier work used simple *pharmaceutical graphs* in the medical context (where nodes were associated to drugs prescribed for treatment in different stages of the condition), further information can be used if available. In addition this approach can be generalised to explanations of similar structures such as our LES. To explain a given trace, we can extract from the event labels all the information that describes that trace and formulate it in first-order logic. Explanations can focus on particular parameters to keep the explanations (their associated formulae) more succinct.

We note that we also need notions of trace equivalence with respect to the arguments (parameters) used to explain different computations. In other words, two computations may be different (the events and relations are different) but their associated explanations can be reduced to the same first-order logical statement and can hence be treated as equivalent under the arguments used. We want to explore the significance of such equivalence notions in practice and we will be inspired by reflections on equivalence such as those given in [28] concerning equivalences between constraint satisfaction problems.

3.5 A Clinical Context

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 healthcare for people with long-term conditions. To think about guidelines from a computer science point of view, we can see them as process descriptions, behavioural models, graphs, etc. There are guidelines for managing the treatment for chronic diseases such as diabetes, cardiovascular disease, chronic kidney 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, and are revised regularly.

When patients have multimorbidity, they are implicitly following several clinical guidelines for their individual diseases in parallel. Clinical guidelines offer treatment recommendations for chronic conditions, but often do not take into account the possible presence of comorbidities. Concretely, for patients with multimorbidity current guideline recommendations rapidly lead to *polypharmacy* (i.e., the prescribing of 5 or more medications) without providing guidance on how best to prioritise recommendations [17]. The risk of medication harm is exacerbated, that is, 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 earlier work, we explored the combination of 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 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). In the initial proposed approach [18], 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 an SMT solver [24] to find the ideal solution with the highest possible score. The approach was illustrated with an example of a patient with five comorbidities from a much cited medical paper [13], and refined in subsequent work establishing an algorithm that searches for optimal medication combinations across treatments maximising medication efficacy, minimising adverse reactions, avoiding intolerances and undesired side effects [12, 8, 10, 7]. At present there is no attempt at deriving explanations for the solutions obtained or a justification as to how different solutions compare; patient preferences are also not modelled to a great extent; and side-effect exploration is currently restricted to a Boolean characterisation.

Recall the models shown in Fig. 2, and assume that they denote the (unfoldings of) treatments (derived from guidelines) for three 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; $x \geq 40$ must hold at for event f_2 for medication `mc1` to be prescribed.

Assume that we know that the occurrence of `ma1` conflicts with `mc2`, and `ma2` conflicts with `mb2` (domain knowledge). 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 the presentation here, assume here that a drug is only used in the context of one treatment. Drug effectiveness, side effects and likelihood of side effects are captured for our example in Table 1. In addition, drugs are known to interact with others. Sometimes additional drugs are added to compensate the interactions as shown in Table 2. We want to combine the diagrams of Fig. 2 in a way that the known underlying conflicts are taken into account. To do so, we need to search valid paths across the three structures that avoid given drug interactions and/or side effects as desired. Intuitively, an optimal solution which focuses on maximising drug effec-

Table 1: Drug Effectiveness and Side Effects.

Drug	Effectiveness	Side Effects	Likelihood
`ma1`	ve_1 1000	`s0`	rare ($\leq 20\%$)
`ma4`	ve_4 900	`s1`	common ($\geq 60\%$)
		`s2`	rare ($\leq 20\%$)
`ma5`	ve_5 600	`s3`	very common ($\geq 80\%$)

Table 2: Drug Interactions.

Drugs	Conflict Level	*Score*
`ma1`, `mc2`	severe	$v1$ -2000
`ma1`, `ma5`, `mc2`	mild	$v2$ -600
`ma2`, `mb2`	severe	$v3$ -1800

tiveness and minimising interaction score, would give us the execution path that includes events e_6 and f_2. If a patient prefers to avoid side effect `s3` then the optimal solution would include event e_4 instead which excludes the medication more likely going to contribute to the side effect.

4 Concluding Remarks

We presented a vision investigating the use of logic as a unifying tool for *modelling*, *reasoning*, *searching* and *explaining* optimal traces of execution. Understanding a notion of distance between traces of execution (with the same optimal value or similar) can be important to decide on what solution to choose, what trade-offs may need to be made as a consequence, and so on. We are currently working towards the formalisation of different notions of distance in order to be able to: *compare* arbitrary solutions; *evaluate* a particular execution with respect to the optimal solution; and *explain* solutions. In other words, the idea is to explore how the interplay between a distance measure and logic can be used to explain different solutions adding domain knowledge (e.g., side effects, patient preferences) to clarify them.

References

1. Avgerinos, T., Cha, S.K., Rebert, A., Schwartz, E.J., Woo, M., Brumley, D.: Automatic exploit generation. Communications of the ACM **57**, 74–84 (2014)
2. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 26, pp. 825–885. IOS Press (February 2009)
3. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, chap. 11, pp. 305–344. Springer (2018)
4. Biere, A., Kröning, D.: Sat-based model checking. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, chap. 10, pp. 277–304. Springer (2018)
5. Boutilier, C., Brafman, R., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. Journal of Artificial Intelligence Research **21**, 135–191 (2004)
6. Bowles, J.: Decomposing Interactions. In: Johnson, M., Vene, V. (eds.) Algebraic Methodology and Software Technology: 11th International Conference, Kuressaare, Estonia, 5-8 July 2006. LNCS, vol. 4019, pp. 189–203. Springer (2006)

7. Bowles, J., Caminati, M.: A flexible approach for finding optimal paths with minimal conflicts. In: International Conference on Formal Engineering Methods. LNCS, vol. 10610, pp. 209–225. Springer (2017)
8. Bowles, J., Caminati, M.: Balancing prescriptions with constraint solvers. In: Liò, P., Zuliani, P. (eds.) Automated Reasoning for Systems Biology and Medicine, Computational Biology, vol. 30, pp. 243–267. Springer (2019)
9. Bowles, J., Caminati, M.: An integrated approach to a combinatorial optimisation problem. In: Ahrendt, W., Tarifa, S.L.T. (eds.) Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11918, pp. 284–302. Springer (2019)
10. Bowles, J., Caminati, M.: Correct composition in the presence of behavioural conflicts and dephasing. Sci. Comput. Program. **185** (2020)
11. Bowles, J., Caminati, M.: A formally verified SMT approach to true concurrency. In: Calimeri, F., Perri, S., Zumpano, E. (eds.) Proceedings of the 35th Italian Conference on Computational Logic - CILC 2020, Rende, Italy, October 13-15, 2020. CEUR Workshop Proceedings, vol. 2710, pp. 357–371. CEUR-WS.org (2020)
12. Bowles, J., Caminati, M., Cha, S., Mendoza, J.: A framework for automated conflict detection and resolution in medical guidelines. Science of Computer Programming **182**, 42–63 (2019)
13. Boyd, C., Darer, J., Boult, C., Fried, L., Boult, L., Wu, A.: Clinical practice guidelines and quality of care for older patients with multiple comorbid diseases: implications for pay for performance. JAMA **294**(6), 716–24 (2005)
14. Donaldson, A., Haller, L., Kroening, D.: Strengthening induction-based race checking with lightweight static analysis. In: VMCAI. LNCS, vol. 6538, pp. 169–183. Springer (2011)
15. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Analysis and Applications **13**, 113–129 (2010)
16. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: ECAI?06. pp. 98–102. IOS Press (2006)
17. Hughes, L., McMurdo, M.E.T., Guthrie, B.: Guidelines for people not for diseases: the challenges of applying uk clinical guidelines to people with multimorbidity. Age and Ageing **42**, 62–69 (2013)
18. Kovalov, A., Bowles, J.: Avoiding medication conflicts for patients with multimorbidities. In: 12th International Conference on Integrated Formal Methods (iFM 2016). LNCS, vol. 9681, pp. 376–392. Springer (2016)
19. Küster-Filipe, J.: Modelling concurrent interactions. Theoretical Computer Science **351**, 203–220 (2006)
20. Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics-Doklady **10**, 707–710 (February 1966)
21. Loreggia, A., Mattei, N., Rossi, F., Venable, K.B.: On the distance between CP-nets. In: Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018). pp. 955–963. ACM (2018)
22. Moura, L., Passmore, G.: The strategy challenge in SMT solving. In: Automated Reasoning and Mathematics. LNCS, vol. 7788, pp. 15–44. Springer (2013)
23. Moura, L.D., Bjørner, N.: Z3: An efficient smt solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
24. Moura, L.D., Bjørner, N.: Satisfiability modulo theories: Introduction and applications. Commun. ACM **54**(9), 69–77 (2011)

25. Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In: PPCP'07. pp. 529–543. Springer (2007)
26. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part i. TCS **13**, 85–108 (1981)
27. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
28. Rossi, F., Petrie, C., Dhar, V.: On the equivalence of constraint satisfaction problems. Tech. Rep. ACT-AI-222-89, MCC Technical Report (December 1989)
29. Shaheen, Q., Toniolo, A., Bowles, J.: Dialogue games for explaining medication choices. In: Gutiérrez-Basulto, V., Kliegr, T., Soylu, A., Giese, M., Roman, D. (eds.) Rules and Reasoning - 4th International Joint Conference, RuleML+RR 2020, Oslo, Norway, June 29 - July 1, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12173, pp. 97–111. Springer (2020)
30. Shaheen, Q., Toniolo, A., Bowles, J.: Argumentation-based explanations of multimorbidity treatment plans. In: PRIMA 2020: Principles and Practice of Multi-Agent Systems. LNCS, vol. 12568. Springer (2021)
31. Winskel, G.: Event structure semantics for CCS and related languages. In: Nielsen, M., Schmidt, E. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 561–576. Springer (1982)
32. Winskel, G., Nielsen, M.: Models for Concurrency. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science, Vol. 4, Semantic Modelling, pp. 1–148. Oxford Science Publications (1995)