On Exploiting Progress for Memory-Efficient Verification of Diagrammatic Workflows

Lars Michael Kristensen[1], Yngve Lamo[1], Wendy MacCaull[3], Fazle Rabbi[1] and Adrian Rutle[2*]

[1] Bergen University College, Bergen, Norway
Lars.Michael.Kristensen@hib.no, Yngve.Lamo@hib.no, fra@hib.no
[2] Alesund University College, Alesund, Norway
adru@hials.no
[3] St. Francis Xavier University, Antigonish, Canada
wmaccaul@stfx.ca

1. Introduction: Workflow management tools may be used in many domains, to guide and direct processes, to support monitoring activities and to increase organizational efficiency. Clinical practice guidelines are textual guidelines describing treatments for specific health problems. Problems can arise if the guideline is misunderstood by the user, if the guideline itself is incomplete, inconsistent or ambiguous, or, if two more guidelines are being followed simultaneously for a patient with several problems. In safety critical applications such as healthcare, it is essential that the workflow is error-free, that is, for every execution of the workflow, necessary behavioural requirements are satisfied and unwanted behaviours do not occur. In earlier work [5], we have proposed a model-driven engineering (MDE) based approach to workflow modelling, with the goals to provide a framework that can model typical healthcare protocols, by means of a visual tool which can be easily understood by the users (usually clinicians), and to articulate and model check behavioural properties. With this approach, the user can input a workflow model and workflow properties which are defined diagrammatically; the model is automatically transformed to DVE code (the DiVinE model checker's language) and the properties to LTL-formulae [6]. If the workflow model is not valid wrt. a property, the tool provides a visual representation of a path which is a counter-example that can be easily analysed for debugging purposes. The verification technique applied is based on explicit state space exploration where all states and all execution paths are explored to check whether the property holds. Most workflow models are so complex that their analysis lead to state-space explosion and memory overflow. In this paper, we propose to use the sweep-line method to exploit inherent progress present in health-care workflows to combat the state explosion problem.

2. Background: Workflow modelling: The syntax and semantics of the workflow modelling language which we use in our approach may be found in [5, 6]. The modelling language is defined using the Diagram Predicate Framework (DPF) [4] and implemented using the DPF Workbench [3]. In DPF, a modelling language is given by a metamodel and a diagrammatic predicate signature (see Fig. 1). The metamodel defines the types and the signature defines the predicates that are used to formulate constraints. A metamodel in DPF consists of an underlying graph, and a set of constraints. We say that a model conforms to (is an instance of) a metamodel if the model's underlying graph is typed by the metamodel's underlying graph, and if the model satisfies the constraints defined in the metamodel. In DPF, the semantics of a (meta)model is given by the set of its instances. DPF supports a multi-level metamodelling hierarchy, in which a model at any level can be regarded the metamodel for models at the level below it.

*in alphabetical order by last name

In the design of our workflow modelling language we have three modelling levels: M2, M1 and M0 (see Fig. 1). The metamodel of our workflow modelling language (which is at level M2) consists of a node Task and an arrow Flow. Simply put, this means that we can define a set of tasks together with the flow relations between these tasks. The signature Σ_2 of the workflow modelling language consists of a set of routing predicates such as `[and_split]`, `[and_merge]`, `[xor_split]`, and `[xor_merge]`. In addition, `[NodeMult,n]` is used to restrict the number of instances (n) a task can have; i.e., it controls how many times a task can be performed, and is used as an upper bound in loops (cycles) in workflow models.

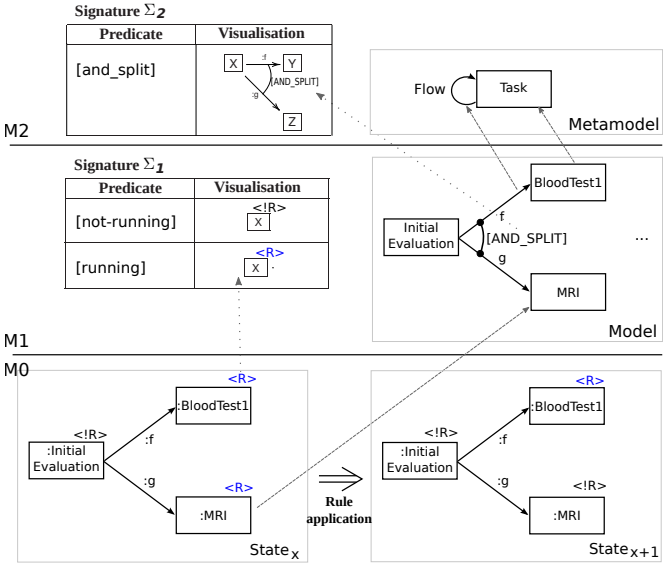

Figure 1: Workflow modelling hierarchy: the dashed arrows indicate the types of model elements, the dotted arrows indicate the relation between the signatures and the models

Given a specific workflow model at level M1 (like the one in Fig. 1) and the predicates `[running]` and `[not-running]` (denoting, respectively, that a task instance is running or not running) collected in a signature Σ_1, we create another modelling language which we use to define *workflow states*, or, equivalently, *instance of a workflow model*. The workflow states are located at level M0 we generate states by applying model transformation rules.

Sweep-line method: The sweep-line method is a state space reduction technique based on the paradigm of on-the-fly state deletion [2]. In order to determine the subsets of states that are to be stored in memory, the sweep-line method exploits a notion of progress exhibited by many systems. The method explores all reachable states while storing only small subsets of the state space in memory at a time. The basic idea of the sweep-line method is when it observes states with a higher progress value, it deletes the states with a lower progress value. It optimistically assumes that the system does not regress; if it turns out that the system does regress to a state with a lower progress value, the method will mark those states as *persistent* (i.e., make them permanently stored in memory) in order to ensure termination [2]. Formally, the progress in a system is formalised by a progress mapping that maps each state into a progress value, and a total order on progress values which is used to determine when states can be deleted.

3. Initial Workflow Verification with the Sweep-Line Method: We performed some verification experiments with workflow models with a small number of tasks. Fig. 2 illustrates a simplified scenario for a cancer treatment. After an initial examination, the patient has an MRI examination and a blood test. These tasks are performed concurrently. An evaluation of the results of the two tests is performed when both tests are completed so the physician can determine which procedure the patient should follow (*either* ProcedureA *or* ProcedureB).

After finishing this procedure, a second evaluation occurs to determine if the patient should continue with a drug treatment or if this workflow should end. The tasks Evaluation2, TakeDrug and BloodTest2 will be repeated up to 5 times, indicated by the loop counter. To specify a

progress measure for a state in the current sweepline implementation for workflow verification, we considered the ordering of tasks; $P_1, P_2, ..$ in Fig. 2 shows the positions of tasks ordering. The ordering shown in the figure is a total order that we obtained from the structural partial ordering of tasks (e.g., $(P_1 < P_2 < P_4)$, $(P_1 < P_3 < P_4)$, $(P_4 < P_5 < P_9)$, etc,.). Clearly, it is also possible to have other total ordering such as $P_1, P_3, P_2, P_4...$ for the given workflow model.

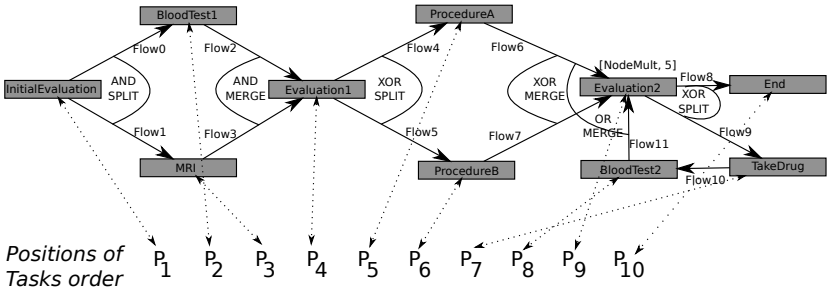

Figure 2: Cancer treatment workflow model

For a specific total ordering if we use $0/1$ in $P_1, P_2, ..$ to represent *not-running/running* task states and an integer number for the loop counter (0-5 for P_9) then the reverse of the number (i.e., right to left ordering) represents the progress value for a workflow state. In situations when we iterate on a loop in our workflow (i.e., Evaluation2 task), the sweep-line method makes some states persistent during the second sweep iteration.

For the cancer treatment workflow model we could define a non-monotonic progress measure by exploiting the loop counter (e.g., the value of P_9 increases from 0 to 5) in the progress measure. If we do not exploit the loop counter (that means if we use $0/1$ for P_9) then we get a monotonic progress measure; but of course still explore the entire state space (albeit some states may be visited multiple times). For the cancer treatment workflow model in Fig. 2, we obtained a reduction of 85% in peak memory usage with the sweep-line method.

In [1] authors presented a sweep-line algorithm for Buchi automata based model checking. The algorithm can detect accepting cycles and can reduce states on the fly; therefore sweep-line methods can be used efficiently for the verification of LTL properties. In future we will evaluate the sweep-line method for the verification of large workflow models and will investigate: i) further reduction using external memory, ii) automated discovery of progress measure, iii) investigate the time/space trade-off when using non-monotonic progress measures.

References

[1] S. Evangelista and L. M. Kristensen. A sweep-line methods for buchi automata-based model checking. *In Fundamenta Informaticae, 2013, to appear.*

[2] K. Jensen, L. M. Kristensen, and T. Mailund. The sweep-line state space exploration method. *Theor. Comput. Sci.*, 429:169–179, 2012.

[3] Y. Lamo, X. Wang, F. Mantz, W. MacCaull, and A. Rutle. DPF Workbench: A Diagrammatic Multi-Layer Domain Specific (Meta-)Modelling Environment. In *Computer and Information Science*, volume 429 of *Studies in Computational Intelligence*, pages 37–52. Springer, 2012.

[4] A. Rutle. *Diagram Predicate Framework: A Formal Approach to MDE.* PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.

[5] A. Rutle, W. MacCaull, H. Wang, and Y. Lamo. A metamodelling approach to behavioural modelling. In *Proceedings of the Fourth Workshop on Behaviour Modelling - Foundations and Applications*, BM-FA '12, pages 5:1–5:10. ACM, 2012.

[6] A. Rutle, F. Rabbi, W. MacCaull, and Y. Lamo. A user–friendly tool for model checking healthcare workflows. *The 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2013), to appear.*