Equivalence Based Cut-off in the Sweep-Line Method

Thomas Mailund 28th February 2001

The nature of the progress measures needed for the sweep-line method lends itself toward infinite state spaces. For most reactive/cyclic models, the progress in the system is carried over to the states resulting in infinitely many states, only distinguished through the progress. In this note we examine the use of equivalences to factor out the progress, without compromising the progress property essential for the sweep-line method.

1 Introduction

A major obstacle for the practical application of the sweep-line method [1] is that most reactive/cyclic models with progress have infinite state spaces. The progress is carried over into the state space, distinguishing otherwise identical states. Equivalence reduction [3] can be used to factorise the infinite number of states, resulting from the progress, into a finite graph. This was done for the special case of timed Petri nets in [2].

Unfortunately, combining equivalence reduction and the sweep-line method is not straightforward. If the progress measure and equivalence relation are compatible, in the sense that $s \in [s'] \Rightarrow \psi(s) = \psi(s')$, which implies that if one equivalence class $[s']$ is reachable from another $[s]$, the elements in the second class have progress less than or equal all states in the first class: $[s] \rightarrow^* [s'] \Rightarrow \psi(s) \sqsubseteq \psi(s')$, combining the two methods are trivial. However, to be useful, the equivalence relation must map states with different progress into the same equivalence class. For this to be possible, in a system with infinite progress, the progress measure and the equivalence relation cannot be compatible.

The approach we suggest here, for systems where the relation and measure are not compatible, is using equivalence relations to define *cut-off points,* states from which we calculate no further. Similarly to the equivalence reduction method, we store representatives for equivalence classes that we have seen during state space exploration, and when we later find a state in the same equivalence class as a previously seen state we stop processing from that point. Since we have already explored the states reachable from an equivalent state, processing states reachable from the new states will bring no further information.

The new method differs from the equivalence reduction method in the fact that we do note store representatives from all states that we see during processing. Doing that would reduce the cut-off method to the original equivalence reduction, and if we had time/space resources to use the equivalence reduction,

there would be no need for the sweep-line method. Instead, we only store representatives of classes that we expect to see in each infinite occurrence sequence.

2 Background

Our model for systems is *labelled transition systems:*

Definition 1 (Labelled Transition System)
A labelled transition system *(LTS) is a tuple* $\mathcal{L} = (S, \Sigma, \Delta, s_I)$*, where* S *is a finite set of* states, Σ *is a finite set of* transition labels, $\Delta \subseteq S \times \Sigma \times S$ *is the* transition relation, *and* $s_I \in S$ *is the* initial state. □

The sweep-line method is based on *progress measures.* A progress measure specifies a *partial order* $(O, \sqsubseteq)$ (i.e., a reflexsive, antisymmetric, and transitive relation) on progress values O of the LTS under consideration, and a *progress mapping* ψ assigning a *progress value* $\psi(s) \in O$ to each state s. We require that the progress measure preserves reachability.

Definition 2 (Progress Measure)
A progress measure *on an LTS* $\mathcal{L} = (S, \Sigma, \Delta, s_I)$ *is a tuple* $\mathcal{P} = (O, \sqsubseteq, \psi)$ *such that* $(O, \sqsubseteq)$ *is a partial order and* $\psi : S \to O$ *is a progress mapping from states into* O *satisfying* $\forall s, s' \in reach_{\mathcal{L}}(s_I) : s \to^*_{\mathcal{L}} s' \Rightarrow \psi(s) \sqsubseteq \psi(s')$. □

The sweep-line method uses progress measures to garbage collect states, as seen in Figure 1.

We will use the following notation: Let $\approx$ be an equivalence relation over the set of states S of some model. We let $S_\approx$ denote the set of all equivalence classes for $\approx$. For a state $s \in S$, $[s]_\approx \in S_\approx$ denotes the equivalence class of $\approx$ containing s. We will write $[s]$ instead of $[s]_\approx$ when $\approx$ is known from context. In the more general equivalence reduction method (as the one found in [3]) the reduction is specified by both an equivalence relation on the set of model states and an equivalence relation on the set of transitions, but for our purpose, the equivalence relation on transitions is not strictly needed. There is nothing in the new method that prevents it though, and the only reason it has been left out is to simplify the presentation.

Not all equivalence relations are useful for equivalence reduction. To ensure that the equivalence graph can be used to reason about the model, we require that the equivalence is *consistent* with the model:

Definition 3 (Consistent, Def. 2.2 [3])
An equivalence relation $\approx \subseteq S \times S$ *is* consistent *iff the following condition holds for all states* $s_1 \in S$, $s_2 \in [s_1]_\approx$, *and transitions* $t \in T$ *:*

$$s_1 \xrightarrow{t} s_1' \;\Rightarrow\; \exists s_2' \in [s_1']_\approx : s_2 \xrightarrow{t} s_2'.$$

□

```
unprocessed = {s_I}
nodes = {s_I}
while ¬ unprocessed.empty()
    s = unprocessed.get_min_element()
    for each t, s' such that s -t-> s'
        if ¬ nodes.contains(s')
            nodes.add(s')
            unprocessed.add(s')
    nodes.garbage_collect()
```

Figure 1: The sweep-line method. The structure `unprocessed` keeps track of the states for which successors are still to be calculated. In each iteration a new unprocessed state is selected, such that this state has a minimal progress value among the states in `unprocessed`. The nodes explored so far are stored in `nodes`, and nodes are only added to `unprocessed` if they are not already in `nodes`. After a node has been processed, nodes with a progress value strictly less that the minimal progress value among the states in `unprocessed` can be deleted. The states are examined in an order determined by the progress measure in order to be able to garbage collect a node as soon as possible.

The equivalence reduced state space of a model is then constructed from the equivalence classes of a consistent equivalence relation:

Definition 4 (Equivalence reduced state space, Def. 2.3 [3])
Let $\approx$ be a consistent equivalence relation. The equivalence reduced state space *of a model with state space (S, T, Δ, s_0) is the tuple $(S_\approx, T, \Delta_\approx, [s_0]_\approx)$, where $([s]_\approx, t, [s']_\approx) \in \Delta_\approx$ iff $(s, t, s') \in \Delta$.* □

The equivalence reduced state space has a node for each reachable equivalence class, and it has an edge between two nodes iff there is a state in the equivalence class of the source node in which a transition is enabled, and whose occurrence leads to a state in the equivalence class of the destination node. The algorithm for constructing the equivalence reduced state space is shown in Fig. 2.

3 The Cut-off Method

For the cut-off method we need at predicate to select the equivalence classes to save during the state space sweep. We therefore require a predicate on states $p : S \to \{\mathtt{tt}, \mathtt{ff}\}$. For states s with $p(s)$, we store $[s]$, for states s with $\neg p(s)$ we do note store $[s]$.

Selecting the right predicate for determining which classes to store is crucial for the method. If too many classes are stored, there may not be enough computer

```
unprocessed = {s_I}
nodes = {[s_I]}
while ¬ unprocessed.empty()
    s = unprocessed.get_next()
    for each t, s' such that s -t-> s'
        if ¬ nodes.contains([s'])
            nodes.add([s'])
            unprocessed.add(s')
```

Figure 2: Exploration of the equivalence reduced state space. The algorithm resembles the usual state space exploration algorithm, but instead of storing the previously explored *nodes*, the corresponding *equivalence classes* are stored, and a state is only considered unprocessed if its equivalence class is not stored among the nodes.

resources to explore the state space. On the other hand, if too few classes are stored, there might exist reachable paths with no cut-off point, in which case the exploration will not terminate. We need at least one cut-off point per infinite occurrence path. This is captured in the following definition:

Definition 5 (Cut-off Predicate)
A predicate $p : S \to \{\mathtt{tt}, \mathtt{ff}\}$ is said to be a cut-off predicate *if all infinite occurrence sequences reachable from the initial state contains an infinite number of states satisfying p.* □

Under the assumption that there are only finitely many equivalence classes (a reasonable assumption given that this is the setting we are working with), this means that eventually all paths must hit upon a cut-off equivalence class previously seen, and exploration along that sequence will terminate. Given a cut-off predicate, the cut-off method combines the sweep-line method and equivalence reduction as shown in Figure 3.

Unfortunately, automatically deriving a cut-off predicate is not immediately possible. Even checking that a given predicate satisfy the property will in general require analysis similar to the analysis we need the predicate for in the first place. Since the property is a "global" property, in the sense that "p is a cut-off predicate" can be expressed as $\Box\Diamond p$, abstractions might be a direction for checking whether a predicate can be used for the method.

References

[1] S. Christensen, L.M. Kristensen, and T. Mailund, *A Sweep-Line Method for State Space Exploration*, Proceedings of TACAS'01 (Tiziana Margaria and Wang Yi, eds.), LNCS, vol. 2031, Springer-Verlag, 2001, pp. 450–464.

```
unprocessed = {s_I}
nodes = {s_I}
equiv_classes = ∅
if p(s_I)
    equiv_classes.add([s_I])
while ¬ unprocessed.empty()
    s = unprocessed.get_min_element()
    for each t, s' such that s -t-> s'
        if ¬ (nodes.contains(s')
              ∨ equiv_classes.contains([s']))
            nodes.add([s'])
            unprocessed.add(s')
            if p(s')
                equiv_classes.add([s'])
    nodes.garbage_collect()
```

Figure 3: The processing of new states in the new method. The sweep-line method is extended with a set `equiv_classes` of equivalence classes, in which all equivalence classes of states satisfying the predicate p is inserted. A new node is only processed if it is not already processed, nor has an equivalent class which is already processed.

[2] ———, *Condensed State Spaces for Timed Petri Nets*, Proceedings of ICATPN'01 (José-Manuel Colom and Maciej Koutny, eds.), LNCS, vol. 2075, Springer-Verlag, 2001, pp. 101–120.

[3] K. Jensen, *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. - Volume 2: Analysis Methods*, Springer-Verlag, 1994.