1 Procrastination Is an Option

Imagine the following scenario: You have a job to do. There is no deadline, but the job has to be done eventually. Clearly, you can either do it today or procrastinate. If you do not do it today then you can do it tomorrow or procrastinate again, etc. Let \( \aleph \) stand for “do the job eventually”. Then we have

\[ \aleph \text{ if } \aleph \text{ today or } \aleph \text{ tomorrow}. \]

This recursive behaviour can be modelled as a fixed point. Why least fixed point? Otherwise, we could procrastinate indefinitely.

2 Propositional Dynamic Logic

We would like to reason about graphs as given in Fig. 1. We call these graphs Kripke models. Basically, a Kripke model is a set of worlds, where each world makes some atomic formulae true. Additionally, we have atomic actions which take us from one world to another. Starting with atomic actions, we can inductively compose more complex actions. If \( \alpha \) and \( \beta \) are actions then so are the sequential composition \( \alpha;\beta \), the non-deterministic choice \( \alpha \cup \beta \), and the repetition \( \alpha^\ast \). This should remind you of regular expressions.

To be able to reason formally about Kripke models, we extend classical propositional logic with formulae of the form \( \langle \alpha \rangle \phi \) and \( \langle \alpha > \phi \) where \( \alpha \) is an action and \( \phi \) is a formula.

The general idea is: if we are in a world \( w \) and we can reach a world \( v \) by executing action \( \alpha \) such that \( \langle \alpha \rangle \phi \) holds in \( v \) then \( \langle \alpha \rangle \phi \) is true in \( w \); if \( \alpha \) holds in all worlds which can be reached from \( w \) by executing \( \alpha \) then \( \langle \alpha > \phi \) is true in \( w \). We give some examples using Fig. 1.

- Exactly the worlds \( w_2, w_3, w_4 \) and \( w_5 \) make the atomic formula \( \phi \) true.
- Consequently, exactly the worlds \( w_2, w_4 \) and \( w_5 \) make \( \langle \alpha \rangle \phi \) true.
- If we execute atomic action \( \alpha \) in world \( w_1 \) then we end up either in \( w_2 \) or in \( w_5 \). Hence \( \langle \alpha > \phi \) is true in \( w_1 \) but \( \langle \alpha \rangle \phi \) is not.
- If \( \alpha \) cannot execute in \( w_1 \). Hence \( \langle \alpha > \phi \) is not true in \( w_1 \) for any formula \( \phi \). However, the formula \( \langle \alpha > \phi \) is trivially true.
- If we execute \( \alpha \circ \beta \) in \( w_2 \), we can only end up in \( w_5 \). Hence \( \langle \alpha > \phi \) and \( \langle \alpha > \beta \phi \) are both true in \( w_1 \).
- Executing \( \alpha \circ \beta \) in \( w_2 \) leads us either to \( w_2 \) or \( w_5 \).
- The star operator \( \langle \alpha > \phi \) can be seen as the reflexive, transitive closure of \( \langle \alpha \rangle \phi \). Thus executing \( \alpha \circ \beta \) in \( w_2 \) can lead to \( w_3, w_4, w_5 \) and \( w_6 \).
- Can you figure out which worlds in Fig. 1 make \( \langle \alpha > \phi \) true?
- Can you relate \( \langle \alpha > \phi \) to \( \aleph \) in Section 1? From now on, we call such formulae eventualities.

3 Deciding Satisfiability

The problem we are interested in is to decide satisfiability. That is, given a formula \( \varphi \), does there exist a Kripke Model \( M \) and a world \( w \) in \( M \) such that \( w \models \varphi \) holds? We now consider a fixed point logic, that is, a logic of eventualities. Two intuitions why deciding satisfiability can be useful.

- You have a specification given as a formula, say a specification of a web-service composed of several basic modules, and you want to obtain an actual configuration which does the job.
- If you want to show that a formula \( \varphi \) is true in all Kripke models and all worlds then you must show that \( \neg \varphi \) is not satisfiable.

Why do I need fixed points?

Fixed points are crucial for expressing important concepts like “will happen eventually” or “will happen infinitely often”. For example, if our system is an elevator, we might want to say that the elevator will eventually stop at my floor when I press the button.

Deciding satisfiability is possible for all formulae but it may be computationally hard. The problem is in EXPTIME which means (very roughly) that for “nasty” formulae the required time to decide satisfiability may double if the size of the formula increases by one. Compare this to finding an element in a list, where the required time doubles if the size of the list is doubled.

Fortunately, a lot of formulae occurring in practice are not “nasty”. Our goal is to find procedures which behave well in practice.

4 Tableau Procedures

Tableau procedures are one approach to decide satisfiability.

5 Experimental Results

As there are no other provers which can handle full PDL, we compared our tree-based method against our graph-based method on randomly generated formulae, see Fig. 5 and 6. As can be seen from Fig. 6, the graph-based method is more stable. That is, the number of formulae which it cannot handle is much lower (in these tests none). For satisfiable formulae, both methods are on par when ignoring timeouts. The tree-based method can even outperform the graph-based method on some examples. For unsatisfiable formulae, the graph method is superior even when ignoring timeouts.