Abstract. This document contains informations about the connections between the paper 'Algebraic Derivation of Until Rules and Application to Timer Verification'and the enclosed KIV project.

1 Structure

Due to historic reasons, some namings deviate from the common ones. We give first an overview of the project structure.

The project consists of the specifications 'semiring', 'idempotent semiring', 'dom_semiring', 'cod_semiring', 'area_semiring', 'modal_semiring', 'kleene_algebra' and 'pedestrian_lights'. 'semiring' and 'idempotent_semiring'contain the specifications and theorems of semirings and idempotent semirings, resp. 'dom_semiring' and 'cod_semiring'are dedicated to semirings with domain and codomain, resp., and the diamond and box operators derived therefrom. 'area_semiring' combines semirings with domain and semirings with codomain, in particular, it contains the Lemma of Schröder and other connections between forward and backward operators (which are not used in the paper nor in the proofs regarding the pedestrian lights). 'modal_semiring' enriches an area semiring with the laws of modality. 'kleene_algebra'does not correspond to a Kleene algebra but to a modal Kleene algebra, i.e., it enriches a modal semiring with the Kleene star. Finally, 'pedestrian_lights'contains the verification of pedestrian lights along the lines of the paper mentioned in the abstract.

2 Namings

Also for historic reasons, some variables and axioms have different names than in the paper. We give here a list from the specification 'pedestrian_lights:

- `req` from the paper corresponds to `counterStart` in KIV.
- `nootherstates` from the paper corresponds to `counter_sum` in KIV. However, in KIV we used instead of $\models \mathtt{ctn}_1 + \sum_{\mathtt{i}=0}^{\mathtt{n}-1} |(\mathtt{nit} \cdot \mathtt{cycle})^*\rangle \mathtt{cti}_1$ the equivalent formulation $\sum_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{cti}_1 + \sum_{\mathtt{i}=0}^{\mathtt{n}-1} |(\mathtt{nit} \cdot \mathtt{cycle})^*\rangle \mathtt{cti}_1 = \mathtt{1}$. Of course, we instantiated it here with concrete values for the `cti`'s, namely `c0`, `c3`, `c13` and `c22`.
- All other counter formalizations corresponds to the axioms containing `c0`, `c3`, `c13` or `c22`.
- `lig` from the paper corresponds to `lights` in KIV.
- `push` from the paper corresponds to `sw` in KIV.
- Equation (5) from the paper corresponds to `green_after_3s` in KIV.
- Equation (7) from the paper corresponds to `green_until_13` in KIV.
- Equation (9) from the paper corresponds to `green_after_redAndsw` in KIV.

3 Code Sample

Below we present a part of the KIV formalization. First, three abbrevations are introduced: `lights` corresponds to `lig` from the paper, while `counterStart` and `nit` correspond to $\texttt{req}_1$ and `nst` there. Then the elementary blocks and two exemplary further blocks are specified (`fd` and `fb` mean "forward diamond" and "forward box"). The excerpt finishes with the counter formalization from the paper. Note that the generic formulas from there are instantiated with concrete values; e.g. the formula `c0_c3` corresponds to **Starting**, $\texttt{c0}_1$ is part of **No simultaneity** and `c13_c22` belongs to **Order** (via the equivalence $p \leq q \Leftrightarrow p \rightarrow q = 1$ for tests p, q). For historic reasons, `anglistic_counter` is the KIV formalization of **Resting**.

```
;; shortcuts
lights\_def: lights = and1 * sr1 * and2 * sr2;
counter\_start: counterStart = c0\_1 * req\_1;
no\_important\_time: nit = c0\_0 * c3\_0 * c13\_0 * c22\_0;
;; modalities
fb\_eq\_fd\_and1: test(p) $\Rightarrow$ fb(and1,p) = fd(and1,p);
fb\_eq\_fd\_and2: test(p) $\Rightarrow$ fb(and2,p) = fd(and2,p);
fb\_eq\_fd\_sr1: test(p) $\Rightarrow$ fb(sr1,p) = fd(sr1,p);
fb\_eq\_fd\_sr2: test(p) $\Rightarrow$ fb(sr2,p) = fd(sr2,p);
;; behavior of and1
and1\_1: (push\_1 * gr\_0) $\leq$ fd(and1, and1\_1);
and1\_0: (push\_0 + gr\_1) $\leq$ fd(and1, and1\_0);
and1\_gr\_1: gr\_1 $\leq$ fd(and1, gr\_1);
and1\_gr\_0: gr\_0 $\leq$ fd(and1, gr\_0);
and1\_c3\_1: c3\_1 $\leq$ fd(and1, c3\_1);
and1\_c3\_0: c3\_0 $\leq$ fd(and1, c3\_0);
and1\_c13\_1: c13\_1 $\leq$ fd(and1, c13\_1);
and1\_c13\_0: c13\_0 $\leq$ fd(and1, c13\_0);
and1\_req\_1: req\_1 $\leq$ fd(and1, req\_1);
and1\_req\_0: req\_0 $\leq$ fd(and1, req\_0);
;; behavior of sr1
sr1\_req\_1: ((and1\_1 + req\_1) * c13\_0) $\leq$ fd(sr1, req\_1);
sr1\_req\_0: (c13\_1 + (req\_0 * and1\_0) ) $\leq$ fd(sr1, req\_0);
sr1\_gr\_1: gr\_1 $\leq$ fd(sr1, gr\_1);
sr1\_gr\_0: gr\_0 $\leq$ fd(sr1, gr\_0);
sr1\_c3\_1: c3\_1 $\leq$ fd(sr1, c3\_1);
sr1\_c3\_0: c3\_0 $\leq$ fd(sr1, c3\_0);
sr1\_c13\_1: c13\_1 $\leq$ fd(sr1, c13\_1);
sr1\_c13\_0: c13\_0 $\leq$ fd(sr1, c13\_0);
;; counter formalization
c0\_c3: counterStart impl fd(kstar(lights), c3\_1) = e1;
c3\_c13: c3\_1 impl fd(kstar(lights), c13\_1) = e1;
c13\_c22: c13\_1 impl fd(kstar(lights), c22\_1) = e1;
c22\_c0: c22\_1 impl fd(lights, c0\_1) = e1;
c31\_after\_c01: counterStart impl
  fd(lights, fd(kstar(c0\_0 * c3\_0 * c13\_0 * c22\_0 * lights), c3\_1)) = e1;
```

```
c131\_after\_c31: c3\_1 impl
  fd(lights, fd(kstar(c0\_0 * c3\_0 * c13\_0 * c22\_0 * lights), c13\_1)) = e1;
c221\_after\_c131: c13\_1
  impl fd(lights, fd(kstar(c0\_0 * c3\_0 * c13\_0 * c22\_0 * lights), c22\_1)) = e1;
counter\_runs: c0\_0 impl fd(kstar lights, c3\_1 + c13\_1 + c22\_1 + c0\_1) = e1;
counter\_sum: c0\_1 + fd(kstar(nit * lights), c3\_1) + c3\_1 + c13\_1 +
   fd(kstar(nit * lights), c13\_1) + fd(kstar(nit * lights), c22\_1) + c22\_1 = e1;
anglistic\_counter: c0\_1 * compl(counterStart) impl fd(lights, c0\_1) = e1;
c0\_1: c0\_1 $\leq$ (c3\_0 * c13\_0 * c22\_0);
c3\_1: c3\_1 $\leq$ (c0\_0 * c13\_0 * c22\_0);
c13\_1: c13\_1 $\leq$ (c0\_0 * c3\_0 * c22\_0);
c22\_1: c22\_1 $\leq$ (c0\_0 * c3\_0 * c13\_0);
c0\_1\_to\_c3\_1: fd(kstar(nit * lights), c3\_1) $\leq$
  compl fd(kstar(nit * lights), c13\_1) * compl fd(kstar(nit * lights), c22\_1) * c0\_0;
c3\_1\_to\_c13\_1: fd(kstar(nit * lights), c13\_1) $\leq$
  compl fd(kstar(nit * lights), c3\_1) * compl fd(kstar(nit * lights), c22\_1) * c0\_0;
c13\_1\_to\_c22\_1: fd(kstar(nit * lights), c22\_1) $\leq$
  compl fd(kstar(nit * lights), c3\_1) * compl fd(kstar(nit * lights), c13\_1) * c0\_0;
```