Algebraic Derivation of Until Rules and Application to Timer Verification

Jessica Ertel
Roland Glück
Bernhard Möller

RAMiCS 2018 Groningen

Introduction

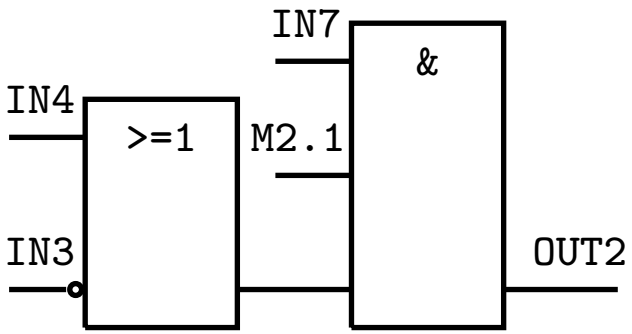
- **Aim:** formalization and interactive verification of programmable logic controllers (PLCs)
- substantial extension of an earlier approach to this
- in particular, modelling timers
- **Approach:** use modal Kleene algebra
- re-use an algebraic formulation of Linear Temporal Logic (LTL)
- **Why?** Algebraic reasoning much more compact, needs much fewer steps than pointwise reasoning in original LTL or Dynamic Logic
- also more general

Programmable Logic Controllers (PLCs)

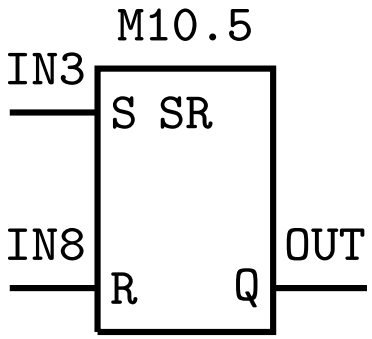
- widely used for the control of robots, plants and mechanical devices
- work in a cyclic way: in each cycle they
  - read values of inputs (from the environment, e.g., switch signals or sensor values) and of internal variables
  - compute new values of internal variables, to store values for the next execution cycle, and of output variables
  - which are forwarded to the environment and may, e.g., start or stop a machine or control the speed of a motor

- most common notation for PLCs: function block diagrams (FBD)
- rectangular *blocks* represent basic functions like Boolean gates
- negation of an input or output is denoted by a small circle
- the *evaluation order* is from left to right and from top to bottom

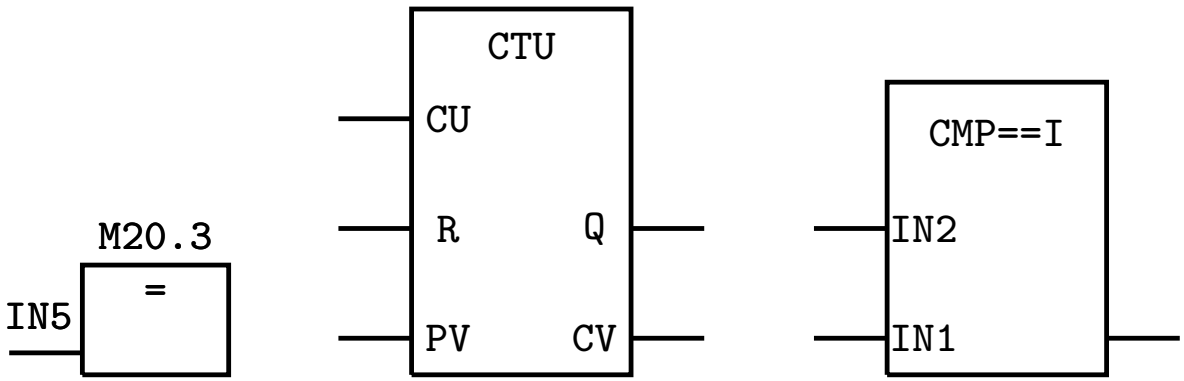
Example



- to store values use, e.g., *flip-flops*
- with a *set* input S and a *reset* input R
- and an internal variable
- whose value is also provided as the output Q



- further elementary blocks: simple assignment, counter and comparator



- finally, timed signals are obtained by configuring the single bits of a specified internal byte as pulse generators with various frequencies

Modeling Blocks Algebraically

- glassbox view of components, i.e., all connection names visible
- state: function from all names to values, and a component a relation between states
- by the evaluation convention a component can be modelled as a linear composition $\texttt{cycle}$ of elementary components
- its overall operation as $\texttt{cycle}^*$
- properties can a.o. be expressed as LTL formulas
- abstraction from relational view: model basic blocks as elements a_i of a modal Kleene algebra and components by linear products

$$\texttt{cycle} = \mathsf{a}_1 \cdot \cdots \cdot \mathsf{a}_n$$

Modal Kleene Algebra

- we assume the basic notion of *Kleene algebras* $(M, +, \cdot, 0, 1, ^*)$
- their elements are abstract representations of transition relations
- sets of states represented by *tests* p:
- need to have a complement $\neg p$ relative to 1, namely $p + \neg p = 1$ and $p \cdot \neg p = 0 = \neg p \cdot p$
- in particular, $p \leq 1$ (subidentity relations)
- $\neg p$ uniquely determined by these axioms if existent
- on tests $+$ and $\cdot$ coincide with disjunction and conjunction
- moreover, $p \leq q$ means p represents a subset of q

- modal operators diamond and box:
- for transition element a and test p, the test $|a\rangle p$ represents the set of all starting states from which a may lead into some state in p
- this is the inverse image of p under a
- box is the De Morgan dual of diamond: $|a]p =_{df} \neg|a\rangle\neg p$
- diamond and box satisfy many useful laws, see paper
- in MKA an abstract relation a is a *total function* if $|a\rangle p = |a]p$ for all tests p however, many proofs only need the weaker condition $|a\rangle p \leq |a]p$ for all tests p
- such an element is called *modally deterministic*

Modeling Blocks in MKA

- Boolean values represented by tests
- each simple Boolean gate is specified by inequations involving the diamond operator
- example: `or`-gate with inputs `in1`, `in2`, `in3` and output `out1`

$$\texttt{in1} + \texttt{in2} + \texttt{in3} \leq |\texttt{or}\rangle\texttt{out1}$$

- every gate `gat` is deterministic and total; so we require $|\texttt{gat}\rangle p = |\texttt{gat}]p$ for every test `p`

LTL

- LTL formulas characterise sets of traces (infinite sequences of "events")
- syntax

$$\Psi \ ::= \ \bot \mid \Phi \mid \Psi \to \Psi \mid \mathsf{X}\,\Psi \mid \Psi\,\mathsf{U}\,\Psi$$

- X and U are the next-time and until operators
- a trace satisfies $\mathsf{X}\varphi$ iff its tail satisfies φ
- hence the semantics of $\mathsf{X}\varphi$ is the inverse image of the semantics of φ under the tail operator
- a trace satisfies $\varphi\mathsf{U}\psi$ if one of its suffixes satisfies φ and all suffixes in between satisfy φ

- logical connectives $\neg$, $\wedge$, $\vee$ defined as usual
- $\top =_{df} \neg \bot$
- "finally" operator F and "globally" operator G:

$$\mathsf{F}\psi =_{df} \top \mathsf{U} \psi \qquad \text{and} \qquad \mathsf{G}\psi =_{df} \neg \mathsf{F} \neg \psi \ .$$

- a trace satisfies $\mathsf{F}\psi$ iff one of its suffixes satisfies ψ
- $\mathsf{G}\psi$ holds iff each of its suffixes satisfies ψ

Algebraic Semantics of LTL

- abstraction: move to Kleene algebras
- use tests instead of trace sets
- use general transition element a instead of the tail operator
- this yields for the semantics $[\![\ ]\!]$

$$\begin{aligned} [\![\bot]\!] &= 0 \\ [\![\varphi \to \psi]\!] &= \neg[\![\varphi]\!] + [\![\psi]\!] \\ [\![\mathsf{X}\,\varphi]\!] &= |a\rangle[\![\varphi]\!]\ , \\ [\![\varphi\,\mathsf{U}\,\psi]\!] &= |([\![\varphi]\!] \cdot a)^*\rangle[\![\psi]\!] \end{aligned}$$

- ψ is *valid*, in signs $\models \psi$, if $[\![\psi]\!] = 1$

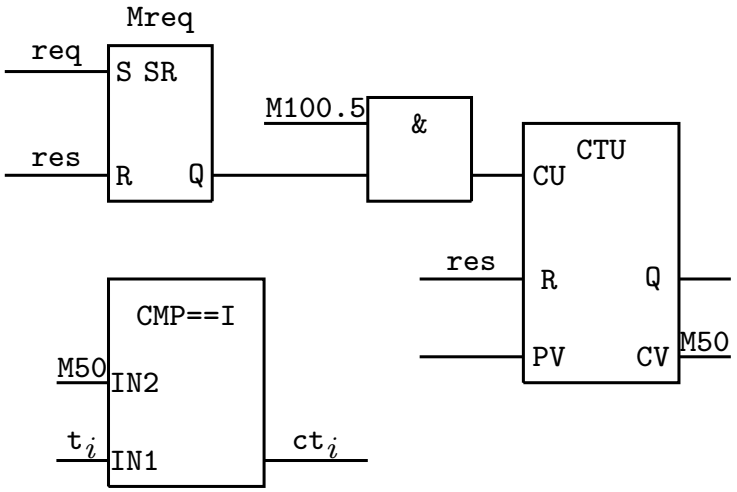
- motivated by these definitions we introduce, for fixed transition element a, temporal operators on tests by

$$\begin{aligned} \bigcirc p &=_{df} |a\rangle p \\ p \,\mathsf{U}\, q &=_{df} |(p \cdot a)^*\rangle q \\ \Diamond q &=_{df} |a^*\rangle q \\ \Box q &=_{df} |a^*] q \end{aligned}$$

- this allows LTL formulas for tests

Formalization of Timers

- a common mechanism for generating timed signals is shown below



- a TRUE on `req` sets the flip-flop, conjoined with a 1 Hz timer
- as long as the flip-flop output stays TRUE the counter value `M50` is increased every second by one

- we view the *timer* as a black box with inputs `req` and `res` as start and reset signals
- comparator results arranged as a sequence $\mathtt{t}_0, \mathtt{t}_1, \ldots, \mathtt{t}_n$
- final value $\mathtt{t}_n$ used as reset input
- Boolean comparator outputs denoted by $\mathtt{ct}_0, \mathtt{ct}_1, \ldots, \mathtt{ct}_n$
- in particular, $\mathtt{ct}_i$ corresponds to a state with counter value $\mathtt{t_i}$

requirements:

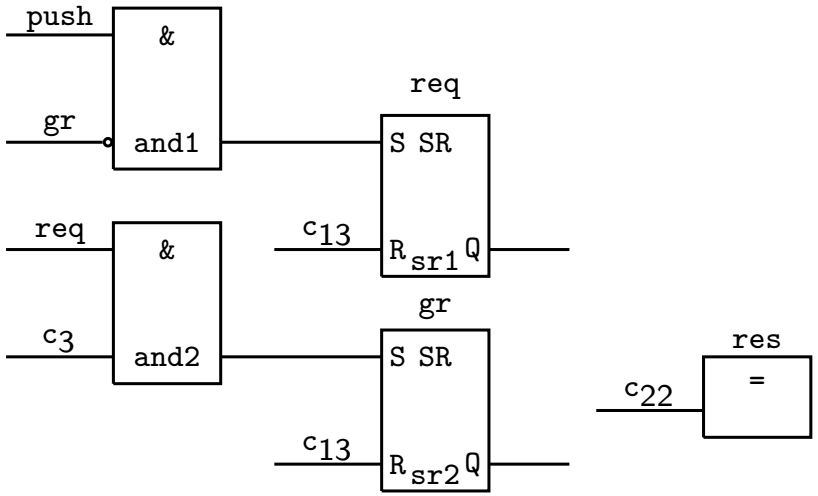
- **No simultaneity:** if the output of one timer comparator is TRUE then the others have to be FALSE: $\mathtt{ct}_i \leq \Pi_{j \neq i} \neg \mathtt{ct}_j$
- **Order:** the timer comparators output TRUE according to the above ordering: $\mathtt{ct}_i \leq \Diamond \mathtt{ct}_{i+1}$ for all $0 \leq i < n$
- **Resetting:** TRUE on $\mathtt{ct}_n$ resets the counter in the following cycle: $\mathtt{ct}_n \leq \bigcirc \mathtt{ct}_0$
- **Resting:** the counter does not start until it gets a request: $\mathtt{ct}_0 \cdot \neg \mathtt{req} \leq \bigcirc \mathtt{ct}_0$

- **Starting:** if a resting counter receives a start request it should eventually output $\mathtt{ct}_1$: $\mathtt{ct}_0 \cdot \mathtt{req}_1 \leq \Diamond\, \mathtt{ct}_1$
- **Intermediate states:** $\mathtt{ct}_i \leq \mathtt{nst}\, \mathsf{U}\, \mathtt{ct}_{i+1}$ for all $0 \leq i < n$, where $\mathtt{nst} =_{df} \prod_{i=0}^{n} \neg \mathtt{ct}_i$ ("no significant time")
  by the resetting rule, $\mathtt{ct}_n$ is followed immediately by $\mathtt{ct}_0$
- **No other states:** $\models \mathtt{ct}_n \ \vee \ \bigvee_{i=0}^{n-1} (\mathtt{nst}\, \mathsf{U}\, \mathtt{ct}_i)$
  - the counter comparator outputs cannot be altered by any block except the last one in the evaluation ordering
  - this ensures that time does not change during the execution of one cycle but also may progress between two consecutive cycles

Case Study: Traffic Lights

- example application: FBD controlling pedestrian lights of a traffic control signal
- when the button is pressed, the pedestrian lights should eventually become green for ten seconds
- after a green phase it should take at least nine seconds before the pedestrian lights can become green again (to respect car drivers)
- also, there should be a time of three seconds for the car traffic lights to become yellow after pushing the request button
- then, the lights should stay green for ten seconds, and a new such cycle can start only nine seconds later

- `push` is a Boolean input from the request button
- `gr` is an variable whose indicating whether the lights are green
- `req` and `res` are the start and reset inputs of a timer
- c_0, c_3, c_{13} and c_{22} are the outputs of a timer corresponding to time values after starting the timer
- the overall system behavior is $\texttt{cycle} = \texttt{and1} \cdot \texttt{sr1} \cdot \texttt{and2} \cdot \texttt{sr2}$



Some Properties of Until

for any transition system the relation transforming a set of states into the set of their successor states is a total function

Lemma assume a to be modally deterministic

1. if $\models p \to \Diamond q$ and $\models p \to \circ(p + q)$ then $\models p \to (p \,\mathsf{U}\, q)$
2. with $u =_{df} q \,\mathsf{U}\, p$ assume $\models p \to \circ u$
   i) $\models p \to \Box u$
   ii) If additionally $\models p \to q$ then $\models p \to \Box q$
3. if a is even total then
   i) $p \leq q \to \circ p \;\Rightarrow\; p \cdot \Diamond q \leq p \,\mathsf{U}\, q$
   ii) $(r \leq \circ(p \cdot \Diamond q) \wedge p \leq q \to \circ p) \Rightarrow r \leq \circ(p \,\mathsf{U}\, q)$

Formal Specification of Traffic lights

we now use LTL formulas to describe our requirements

1. if the timer value is three and there is a request then the lights should be green in the following cycle:

$$\models \Box(\mathtt{req} \wedge \mathtt{c}_3 \rightarrow \bigcirc \mathtt{gr})$$

2. if the lights just turned green (after three seconds), they should stay green for ten seconds:

$$\models \Box\,(\mathtt{req} \wedge \mathtt{c}_3 \rightarrow \bigcirc(\mathtt{gr}\,\mathsf{U}\,\mathtt{c}_{13}))$$

3. overall behaviour

- reasonable choice for initial state: lights red and timer zero
- then always, after pushing the button, we should reach a state such that next
  - eventually the lights turn green,
  - the counter value is thirteen, and
  - in the next state the lights are red until the counter reaches 22
- LTL-formulation:

$$\models \quad \neg \mathtt{gr} \wedge \mathtt{c}_0 \rightarrow \Box\,(\neg \mathtt{gr} \wedge \mathtt{push} \rightarrow \bigcirc(\Diamond\, \mathtt{gr} \wedge \mathtt{c}_{13} \wedge \bigcirc(\neg \mathtt{gr}\, \mathsf{U}\, \mathtt{c}_{22})))$$

Conclusion

- paper outlines the formal proofs of these three properties;
- they involve the properties of until shown in the beginning
- and have been carried out with the verification system KIV
- after some time of familiarization, proving in KIV became routine work without greater difficulties
- an increasing amount of calculation rules and lemmata in MKA made it a pleasant task

- next we will tackle the verification of larger, more lifelike systems
- evisaged help for this goal: the by now accumulated substantial body of reusable rules and lemmata
- other future topics: automated construction of input files, comparison with model checkers and extension of the approach to other PLC languages

- another task: formal proof of the timer properties (presently inserted as axioms without further verification)
- proof needs meta-knowledge about the natural numbers which has to be added in some way
- also need to model that the execution time of one cycle of the PLC does not exceed the period of the used frequency generator
- otherwise, effects similar to the Nyquist-Shannon sampling theorem would destroy the functionality of the timer