On the Characterization of Until as a Fixed Point under Clocked Semantics

Dana Fisman

Hebrew University,

IBM Haifa Research Lab

In Singly Clocked Designs

- The signal that causes a memory element (flip-flop or latch) to make a transition is termed the clock
- The temporal operators in logics such as LTL are interpreted with respect to the clock.
- The formula

  globally (p -> next q)

  is interpreted as

  globally, if p then at the next clock tick, q.

In Multiply Clocked Designs

- Some flip-flops may be clocked with clka and some with clkb
  - Thus, the mapping between temporal operators and clock cycles cannot be done automatically
  - The formula itself must provide the desired mapping
- For example

  globally (p -> next q)@clka

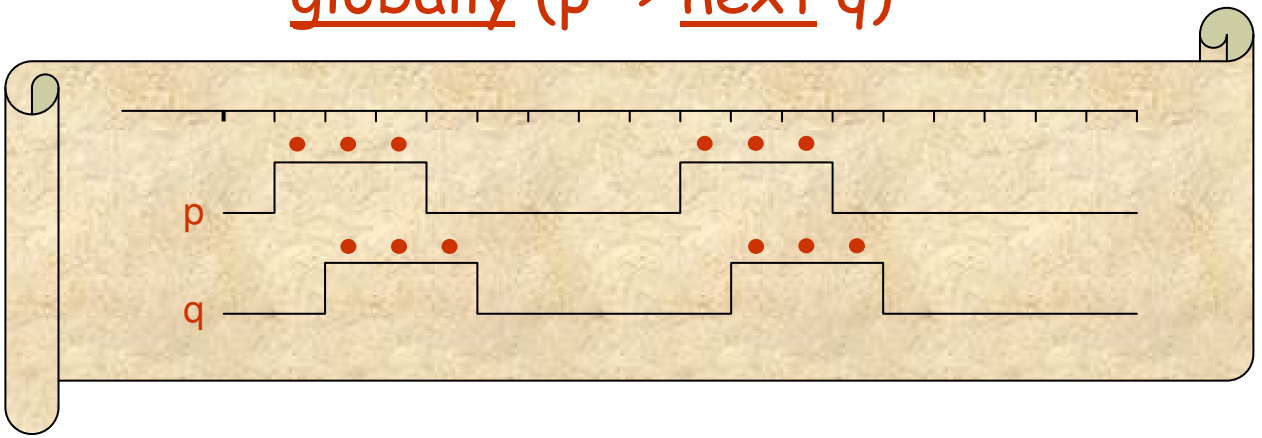
  is interpreted as

  globally, if p during a cycle of clka

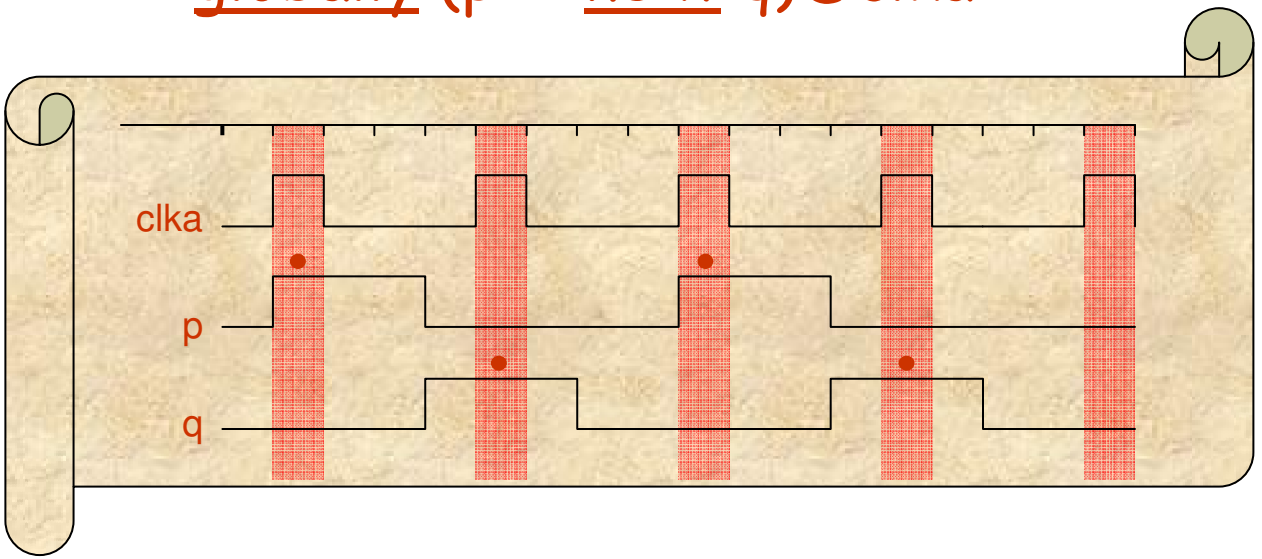
  then at the next clock tick of clka, q

Singly vs. Multiply Clocked Designs

globally (p -> next q)



globally (p -> next q)@clka



Previous Work

- In [EFHMV03] Eisner et al gave a simple semantics for LTL extended with a clock operator
- The suggested semantics has been adopted by the IEEE standards PSL and SVA
- The logic in [EFHMV03] was measured against a list of design goals
- It was shown that it meets all design goals, **except for**: preserving the least-fixed point characterization of the until! operator under multiple clocks

The characterization of until as fixed point is not merely a theoretical issue – it has practical applicability for tools

In this work

- We show that with a minor addition to the semantics of [EFHMV03] the until! operator preserves its least fixed-point characterization (as well as the other requirements)

Overview of the talk

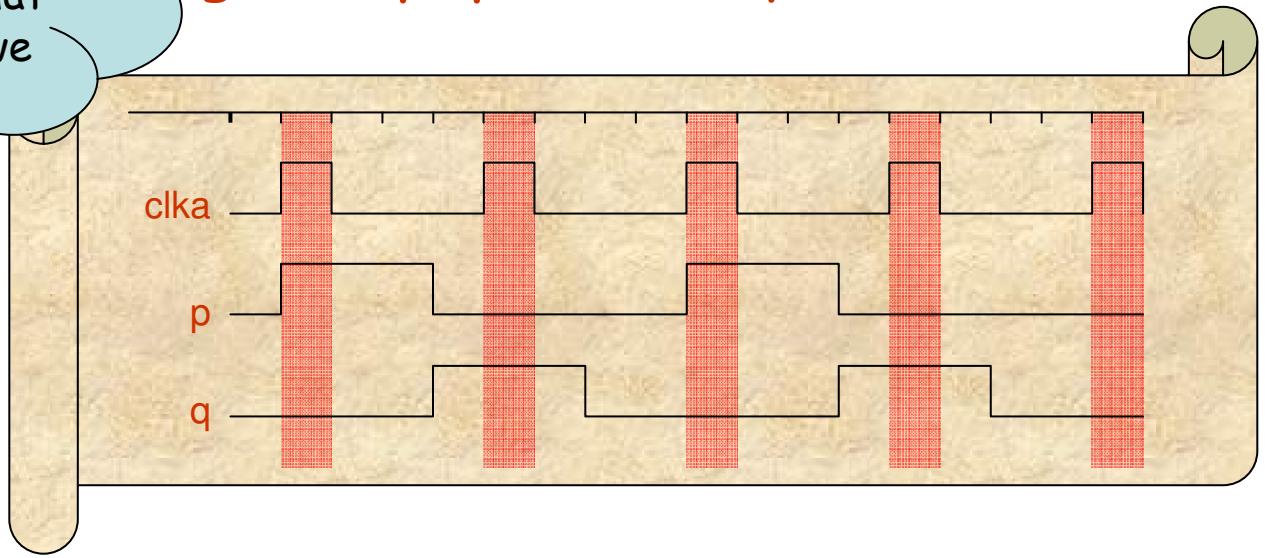
- Understanding the ideas and semantics of [EFHMV03]
- Understanding the problem with the least fixed-point characterization of until!
- The proposed solution

The idea in [EFHMV03]

- Projection: the role of the clock operator is to define the projection on the cycles in which the formula should be evaluated.

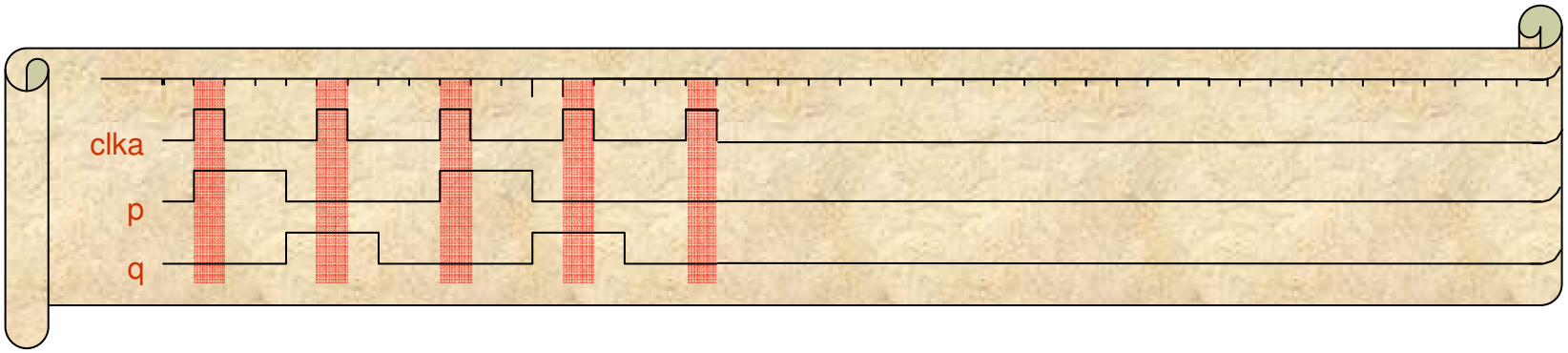
The projection is going to be one that we can undo (as we will see later)

globally (p -> next q)@clka



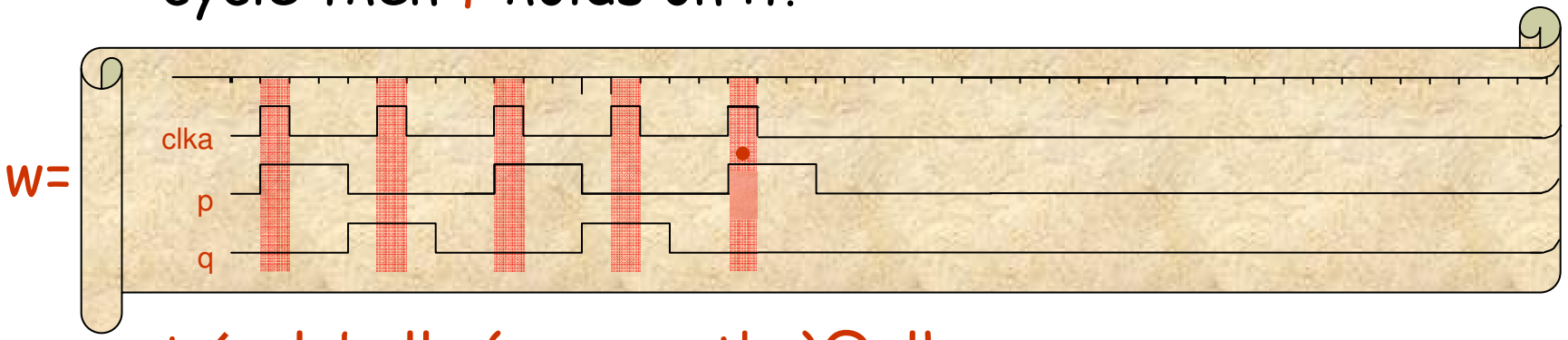
Coping with finite ticks [EFHMV03]

- But what if the clock stops ticking?
- The semantics of LTL is typically defined for infinite paths.
- A projection w.r.t clka of an infinite path where clka stops ticking yields a finite path



Coping with finite ticks [EFHMV03]

- Solution: use the weak and strong versions of the next operator (next! and next) that are introduced by Pnueli et al. for finite path
  - The formula next! f demands that there is a next cycle and f holds on it
  - Whereas next f demands that if there is a next cycle then f holds on it.



w |≠ globally (p -> next! q)@clka

w |= globally (p -> next q)@clka

Coping with zero ticks [EFHMV03]

- But what if the clock never ticks?
- The semantics of LTL (for finite/infinite words) assumes paths are non-empty.
- A projection w.r.t clka of a finite/infinite path where clka never ticks yields an empty path.

Coping with zero ticks [EFHMV03]

- Solution: define weak and strong versions of a boolean expression (b! and b)
- The formula b! demands that there is a current cycle and b holds on it
- Whereas b demands that **if** there is a current cycle then b holds on it.

Nested Clocks [EFHMV03]

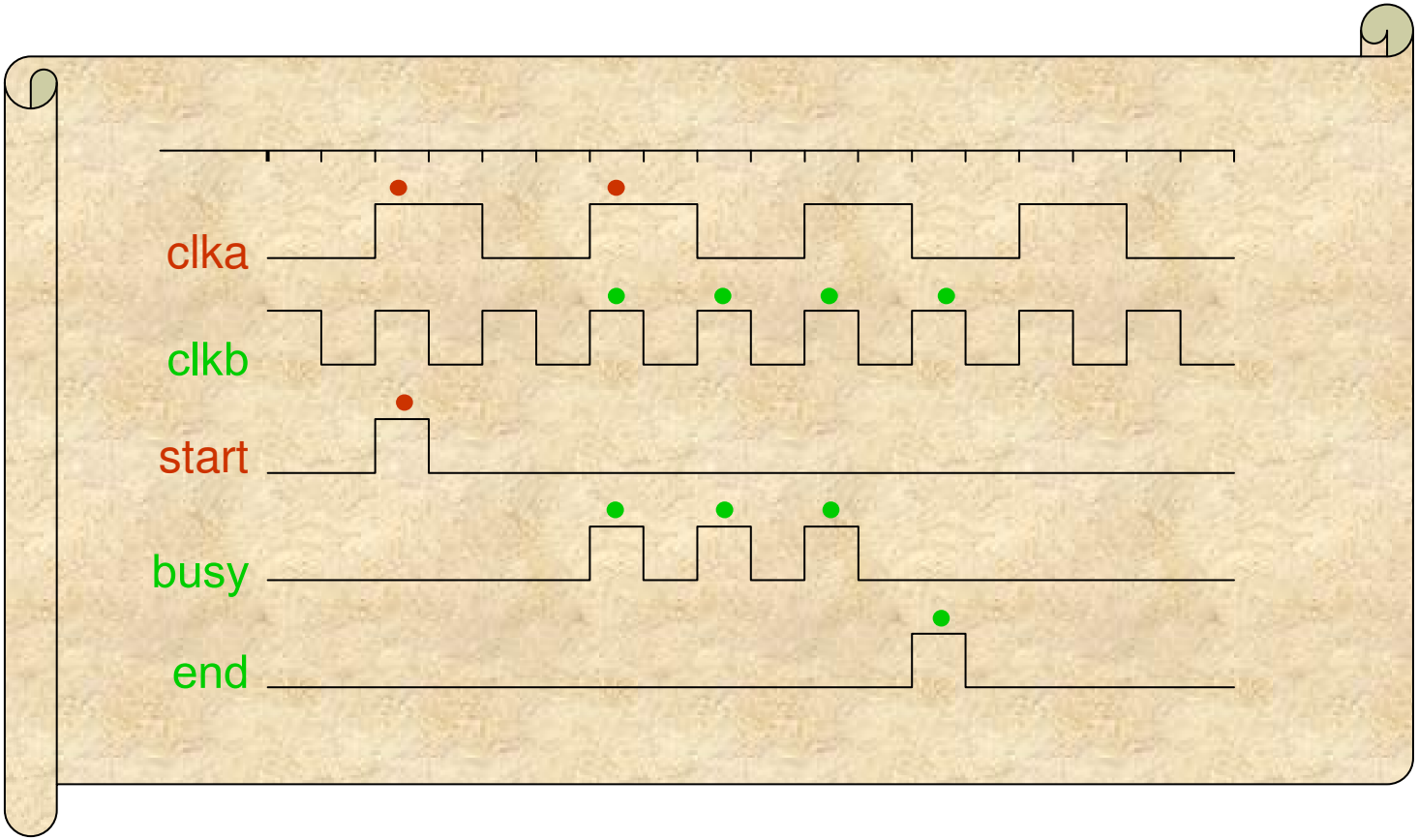
- What happens in the event of a clock switch?
- Consider

  (start -> next ((busy until end))@clkb)@clka

- We would like the inner formula to be evaluated on the cycle where clkb holds rather than the cycles where both clka and clkb hold
  - That is, we would like the nested clock to define a new projection rather than refine the projection further

Nested Clocks [EFHMV03]

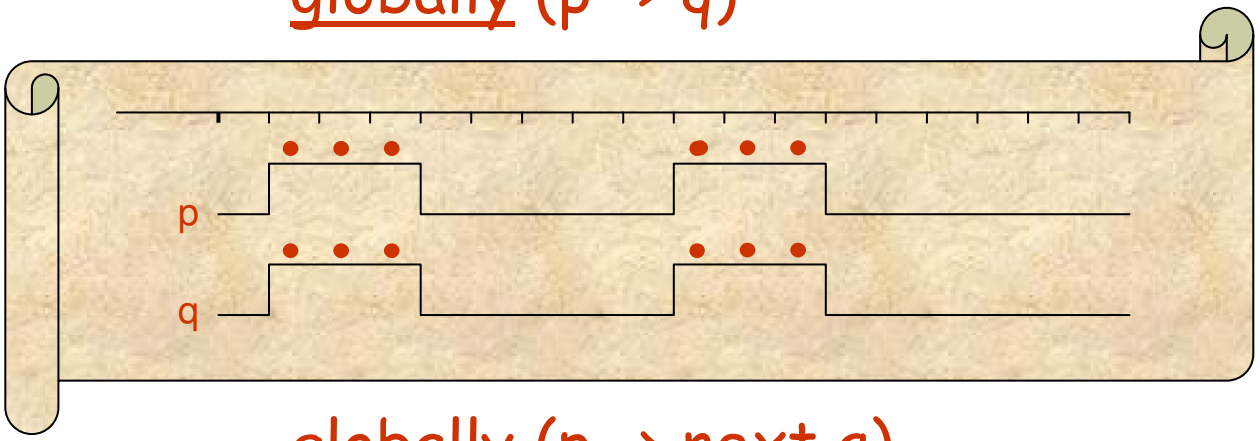
(start -> next ((busy until end))@clkb)@rose(clka)



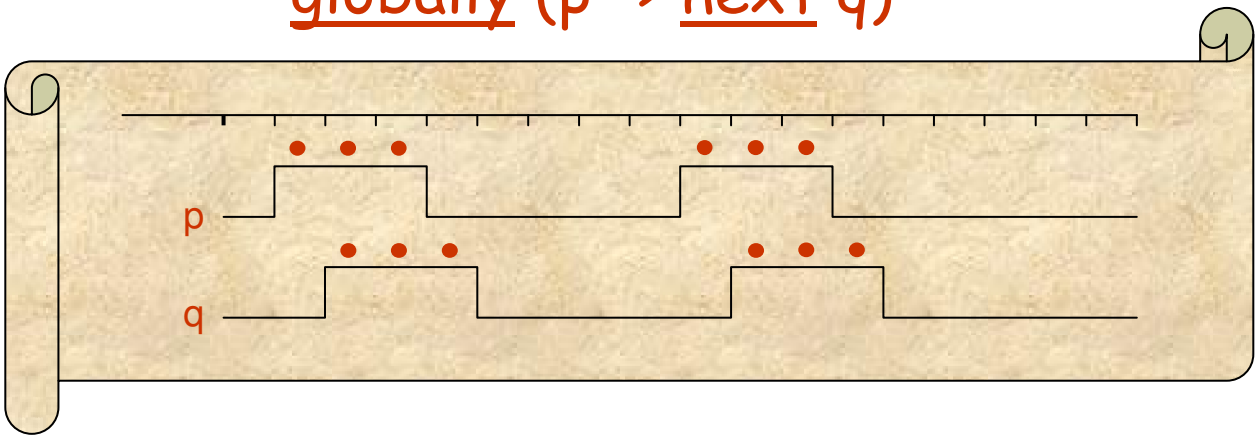
b and next b under multiple clocks

- In LTL
  - The formula b checks b at the current cycle
  - The formula next b checks b at the next cycle

globally (p -> q)



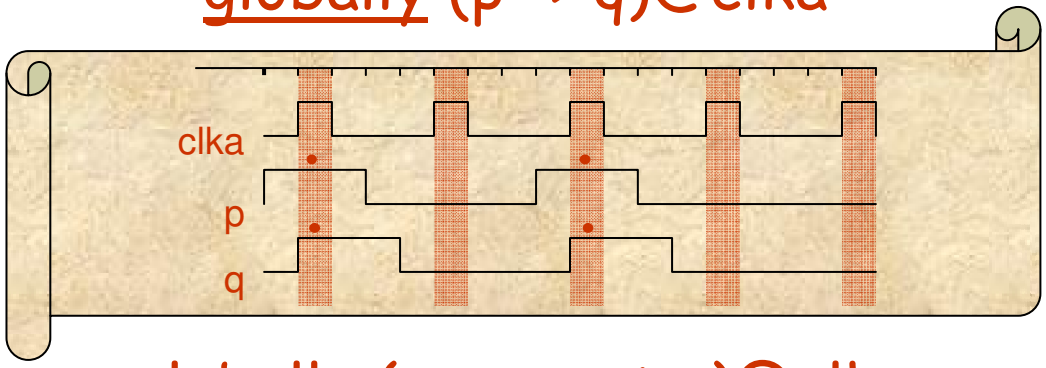
globally (p -> next q)



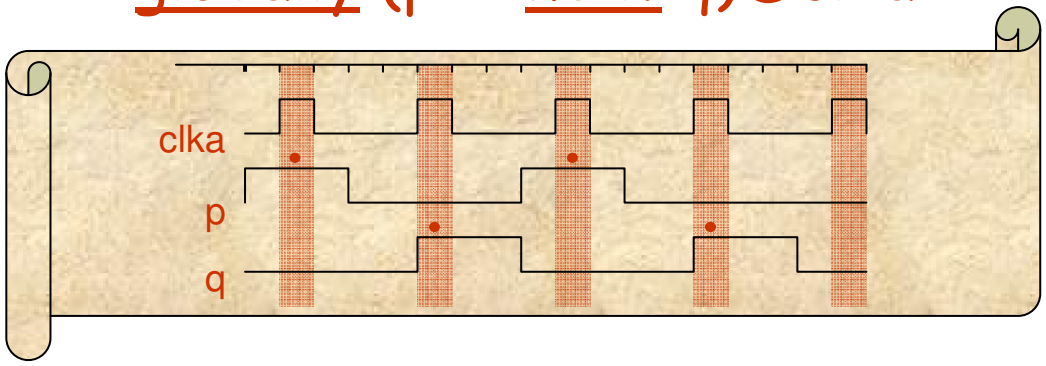
b and next b under multiple clocks

- In LTL@
  - The formula b@clk checks b at the closest clock tick of clk (if such a tick exists)
  - The formula next b@clk checks b at the second closest clock tick of clk (if such a tick exists)

globally (p -> q)@clka



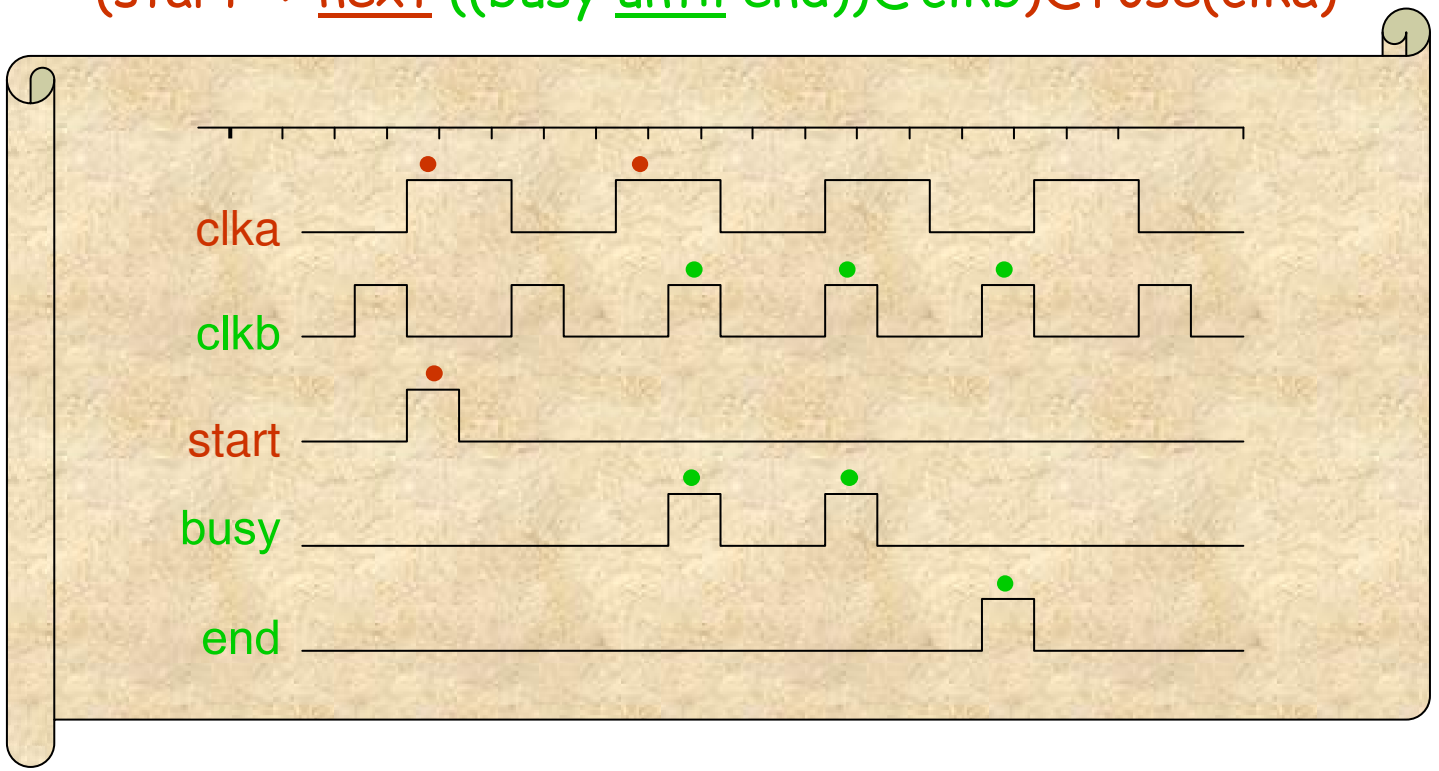
globally (p -> next q)@clka



Misaligned Nested Clocks

- In singly-clocked formulas all sub-formulas are evaluated on a clock tick, thus the closest clock tick is always the current tick
- In multiply-clocked formulas sometimes sub-formulas are evaluated on a cycle which is not a clock tick, thus the closest clock tick may be different than the current tick

(start -> next ((busy until end))@clkb)@rose(clka)



The Semantics of [EFHMV03]

A finite word w is a clock tick of clk if clk holds only on the last letter of w.

- w $\models_{clk}$ b! iff w contains at least 1 clock tick of clk and b holds on the first
- w $\models_{clk}$ b iff if w contains at least 1 clock tick of clk then b holds on the first

- w $\models_{clk}$ next! f iff w contains 2 clock ticks of clk and f holds on the second
- w $\models_{clk}$ next f iff if w contains 2 clock ticks of clk then b holds on the second

- w $\models_{clk}$ $f \wedge g$ iff w $\models_{clk}$ f and w $\models_{clk}$ g
- w $\models_{clk}$ $\neg f$ iff w $\not\models_{clk}$ f

- w $\models_{clk}$ f until! g iff there exists a clock tick of clk in w where g holds and f holds on every preceding tick of clk
- w $\models_{clk}$ f until g iff either w $\models_{clk}$ f until! g or f holds on every tick of clk in w

- w $\models_{clk}$ f@clkb iff w $\models_{clkb}$ f

The Problem with [EFHMV03]'s solution

- In LTL f $\underline{\text{until}}!$ g is a least fixed-point solution of the equation

$$E(S) = g \vee (f \wedge \underline{\text{next}}!\ S)$$

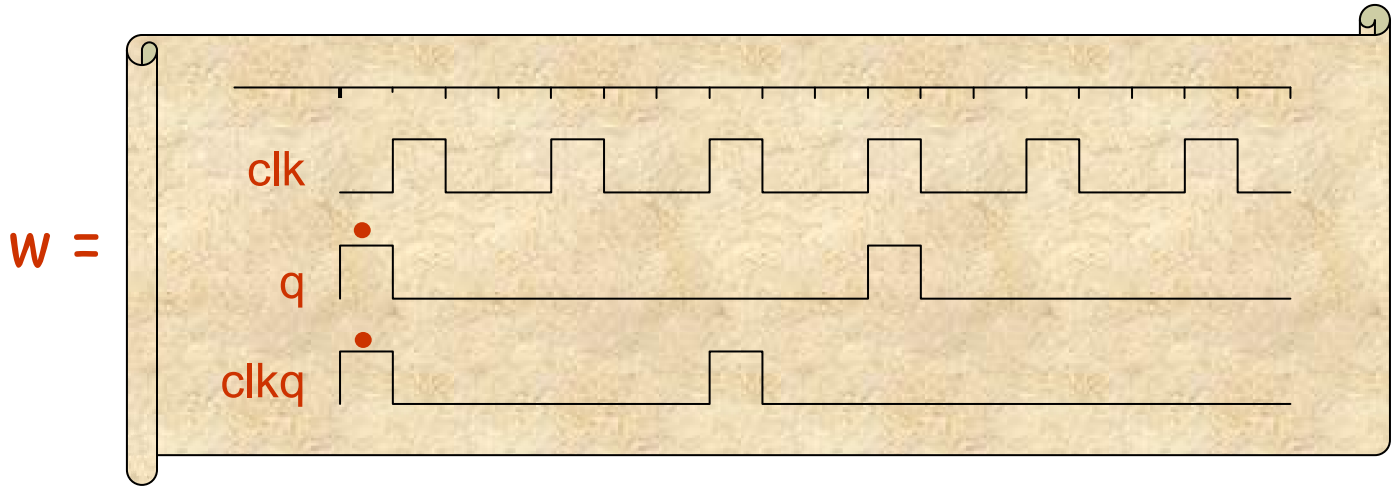
- In [EFHMV03] Eisner et al show that f $\underline{\text{until}}!$ g is a least fixed point solution of the equation

$$E'(S) = (\text{true}! \wedge g) \vee (f \wedge \underline{\text{next}}!\ S)$$

  - If only a single clock is involved.
  - When multiple clocks are involved the characterization no longer holds.

The counter example [EFHMV03]

(p until! (q@clkq))

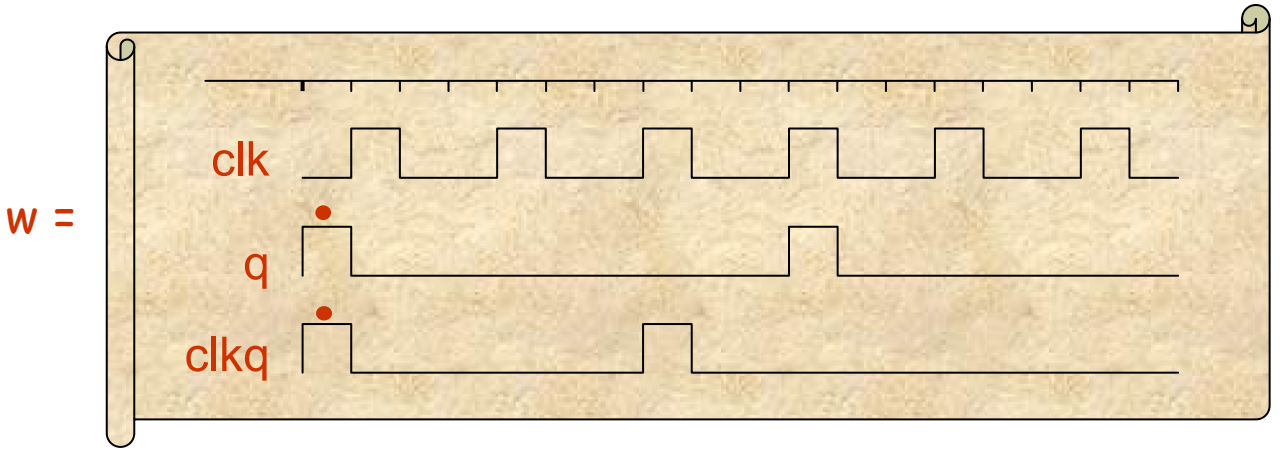


- w ⊭clk (p until! (q@clkq))
  Since there is no clock tick of clk where q@clkq holds

- w ⊨clk (true! ∧ (q@clkq)) ∨ (p ∧ Next! (p until! (q@clkq)))
  Since both q@clkq and true! hold on the first cycle

The Problem

- The problem in [EFHMV03] is that there is no way to express the property "evaluate f at the closest clock tick of clkb"
  - Unless the clock context (the outer clock) is clkb
  - In the previous example we got that q@clkq is evaluated now while we wanted it to be evaluated at the closest clock tick of clk



$w \models_{clk} (q@clkq)$

Solution

- Solution: introduce an alignment operator (such as in CBV) that takes you to the closest clock tick
  - Actually 2 alignment operators (weak & strong)

Formally

- We introduce next!^{m} and next^m
  - The formula (next!^{m} f)@clk demands there are m+1 clock ticks of clk and f holds on the last of them
  - The formula (next^m f)@clk demands that *if* there are m+1 clock ticks of clk *then* f holds on the last of them

- When the exponent
  - m=1 we get the usual next/next! operators
  - m>1 we get iteration of the usual next/next! operators
  - m=0 we get the alignment operators

The Semantics of [EFHMV03]

A finite word w is a clock tick of clk if clk holds only on the last letter of w.

- $w \models_{clk} b!$ iff w contains at least 1 clock tick of clk and b holds on the first
- $w \models_{clk} b$ iff if w contains at least 1 clock tick of clk and b holds on the first

- $w \models_{clk} \underline{next}!\ f$ iff w contains 2 clock ticks of clk and f holds on the second
- $w \models_{clk} \underline{next}\ f$ iff if w contains 2 clock ticks of clk then b holds on the second

- $w \models_{clk} f \wedge g$ iff $w \models_{clk} f$ and $w \models_{clk} g$
- $w \models_{clk} \neg f$ iff $w \not\models_{clk} f$

- $w \models_{clk} f\ \underline{until}!\ g$ iff there exists a clock tick of clk in w where g holds and f holds on every preceding tick of clk
- $w \models_{clk} f\ \underline{until}\ g$ iff either $w \models_{clk} f\ \underline{until}!\ g$ or f holds on every tick of clk in w

- $w \models_{clk} f@clkb$ iff $w \models_{clkb} f$

The Resulting Semantics

A finite word w is a clock tick of clk if clk holds only on the last letter of w.

- $w \models_{clk} b!$ iff w contains at least 1 clock tick of clk and b holds on the first
- $w \models_{clk} b$ iff if w contains at least 1 clock tick of clk and b holds on the first

- $w \models_{clk} next!^m f$ iff w contains m+1 clock ticks of clk and f holds on the second
- $w \models_{clk} next^m f$ iff if w contains m+1 clock ticks of clk then b holds on the second

- $w \models_{clk} f \wedge g$ iff $w \models_{clk} f$ and $w \models_{clk} g$
- $w \models_{clk} \neg f$ iff $w \not\models_{clk} f$

- $w \models_{clk} f$ until! g iff there exists a clock tick of clk in w where g holds and f holds on every preceding tick of clk
- $w \models_{clk} f$ until g iff either $w \models_{clk} f$ until! g or f holds on every tick of clk in w

- $w \models_{clk} f@clkb$ iff $w \models_{clkb} f$

Fixed-Point Characterization of until!

- In LTL f until! g is a least fixed-point solution of the equation

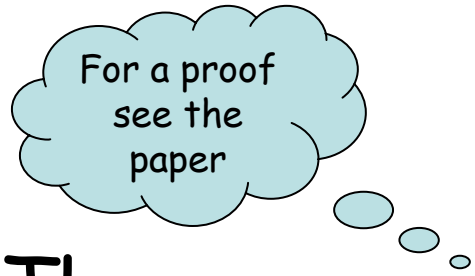
$$E(S) = g \vee (f \wedge \text{next! } S)$$

- In the suggested logic f until! g is a least fixed point solution of the equation

$$E'(S) = \text{next!}^0 (g \vee (f \wedge \text{next! } S))$$

- That is

$$E'(S) = \text{next!}^0 E(S)$$

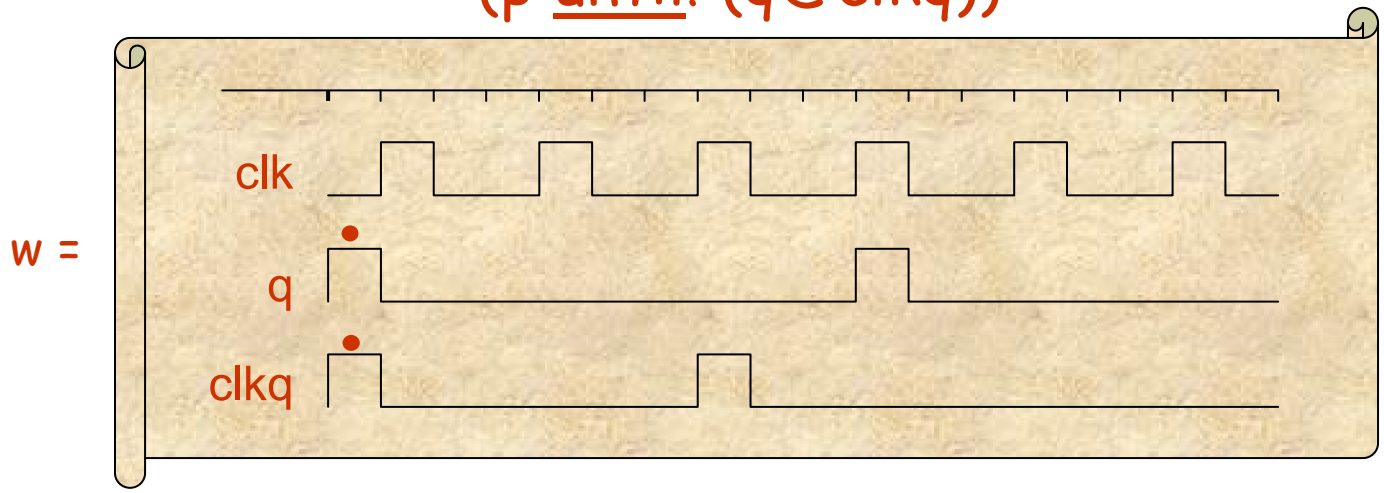


- A simple generalization of the LTL characterization!!!

Examining that counter example

- The problem with the counter example was that the until formula did not hold while the "bad" fixed-point characterization did.
- Now both do not hold:

(p until! (q@clkq))



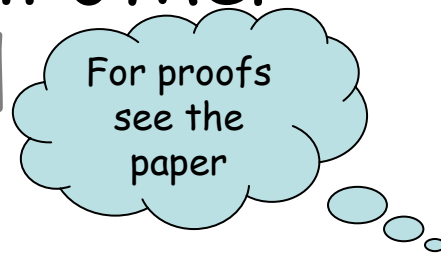
$w \not\models_{clk}$ next!^{0} ((q@clkq) ∨ (p ∧ next! (p until! (q@clkq))))

since both $w \not\models_{clk}$ next!^{0} (q@clkq)

and $w \not\models_{clk}$ next!^{0} next! (p until! (q@clkq))

Conclusions

- We have shown that by adding weak/strong alignment operators to LTL@ the least fixed-point characterization of until! is preserved
- The resulting logic is obtained by augmenting the next/next! operators of LTL@ by an exponent
  - the alignments operators are obtained by taking the exponent to be zero
- The resulting semantics meets all other requirements set by [EFHMV03]



The End

Thank you!