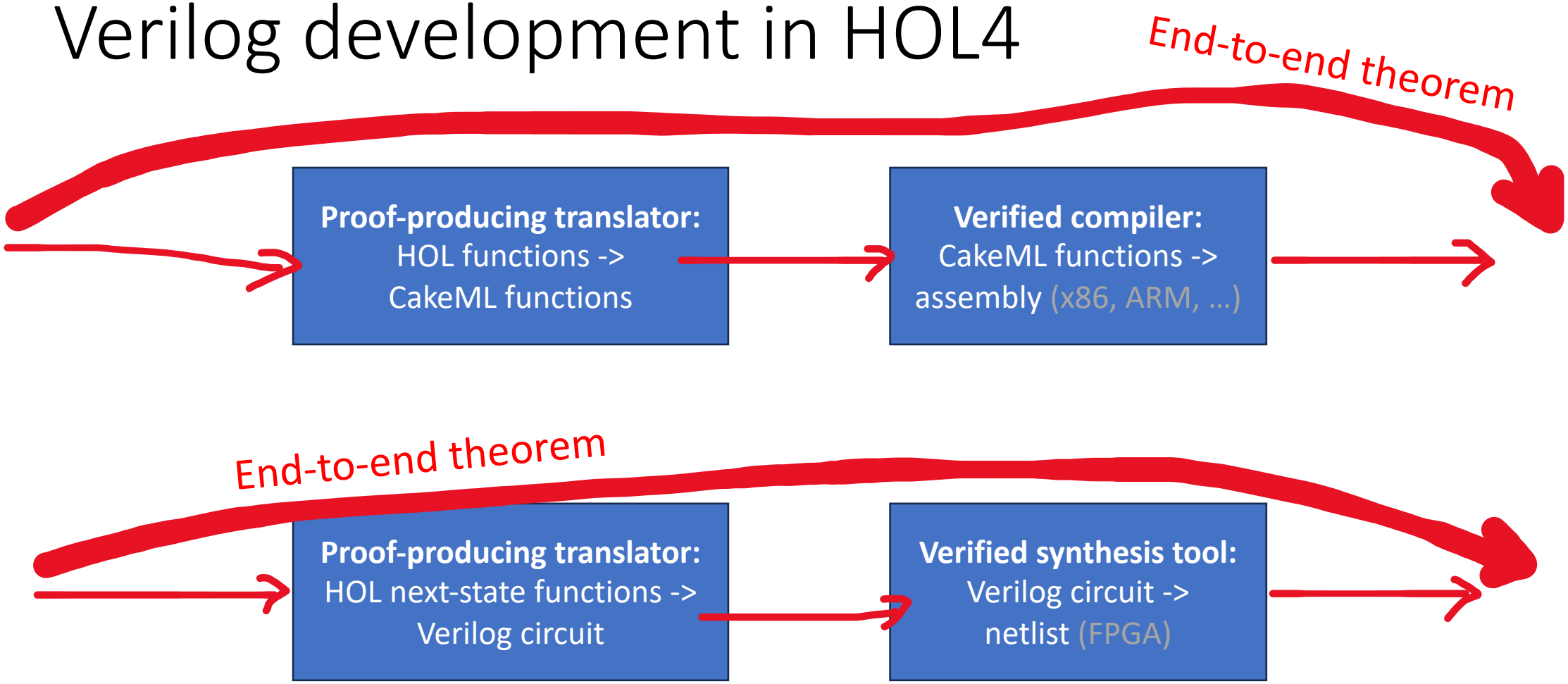
The current state of Verilog semantics modelling in HOL4

Andreas Lööw

Imperial

London

Verilog development in HOL4



Tools/semantics used in four projects

- CakeML (PLDI'19): verified processor that can run CakeML programs
- Ning Dong (FMCAD'23): extension of above processor to pipelining
- Yann Herklotz (OOPSLA'21, PLDI'24): verified C-to-Verilog compiler ("HLS")
- In progress: Verified Verilog formal equivalence tool

The relevant subset of Verilog
(for synchronous hardware)

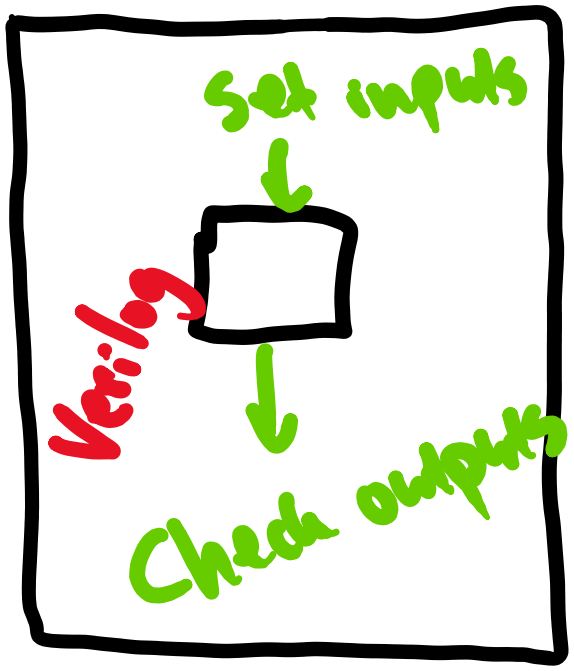
Verilog = large

- Standard is 1315 pages

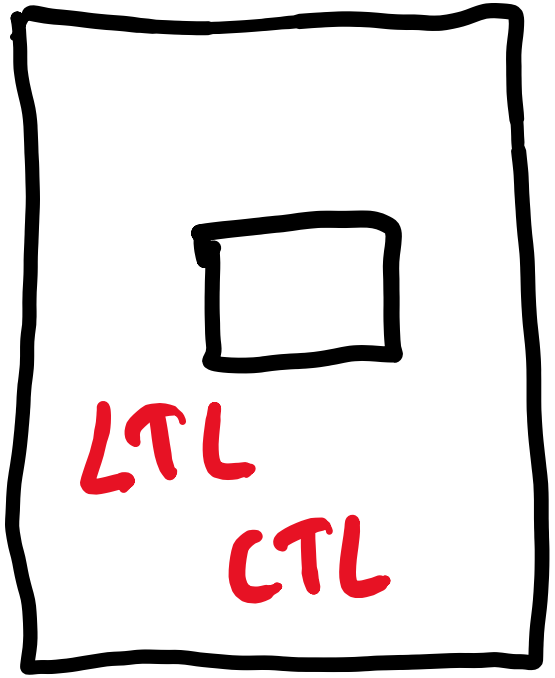
- Essentially two languages in one:
  - ”HDL” – hardware-description language
  - Test infrastructure / “test benches”

Some hardware “verification” setups

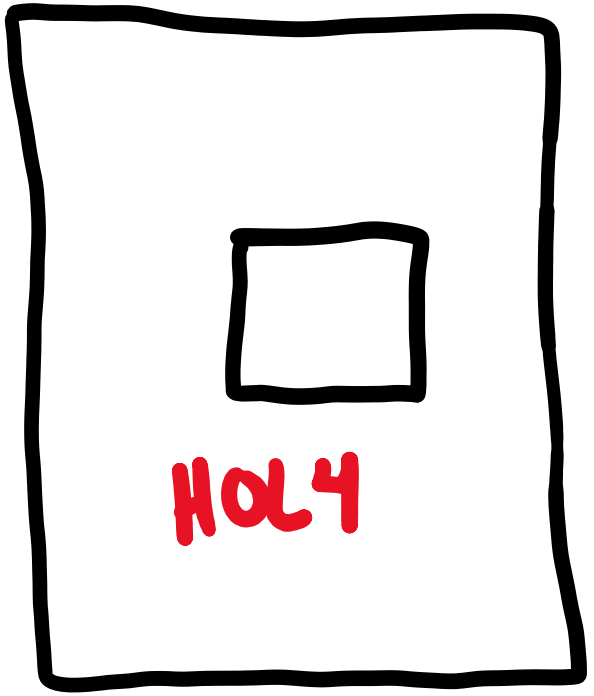
The traditional setup



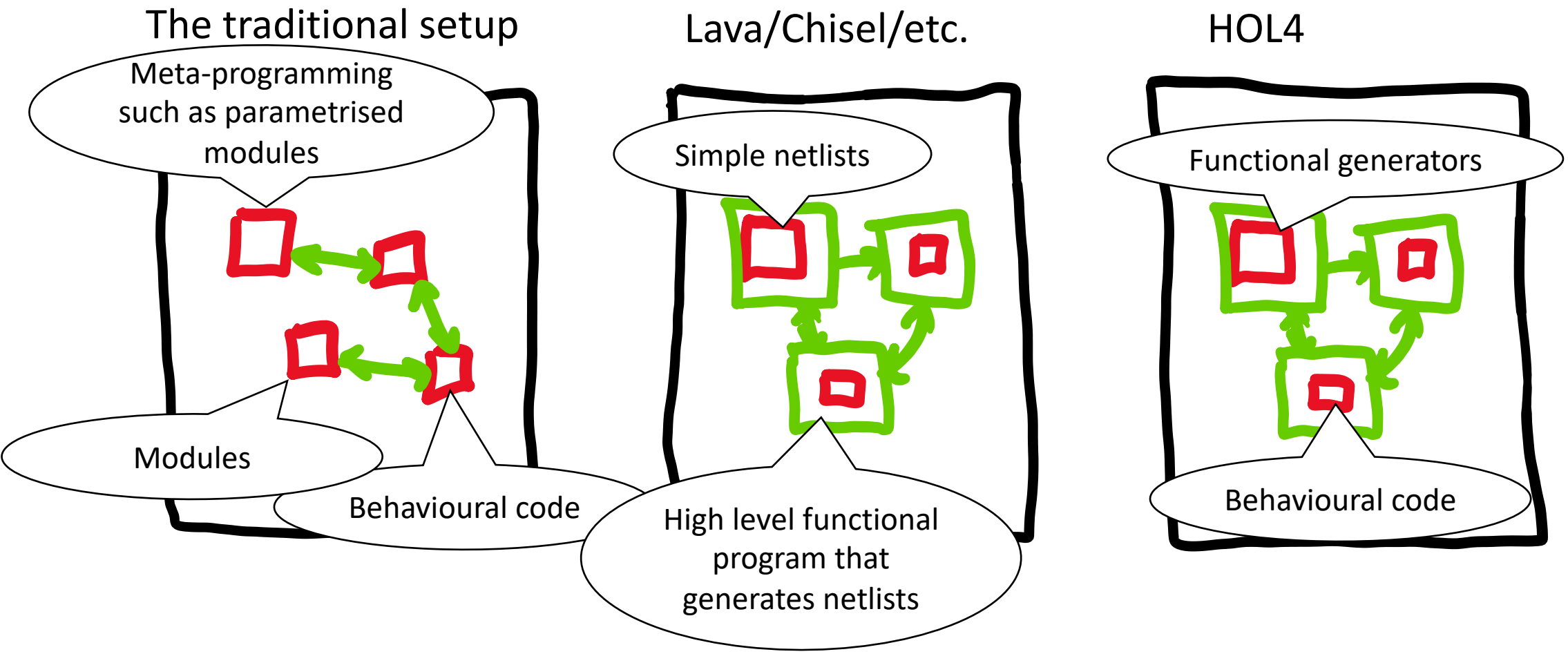
Model checking



HOL4



Some hardware development setups



What’s left (i)

- Hardware is not so complicated
- It’s about gluing together memory elements using combinational logic
- Memory elements such as registers, RAM memories, etc.
- Combinational logic = pure function implemented in hardware

What's left (ii)

- Values: 4 bit values (1, 0, X, Z) and various arrays of these
- "Variables": nets and variables
- Expressions: a lot of them but pretty straightforward
- Statements: various C-like constructs (if-statements, switch/case-statements, assignments) and some hardware-specific constructs (e.g. "nonblocking" assignments and event control)
- Processes: initial, always, always_comb, always_ff, assign

The semantics of this relevant subset of Verilog

The semantic landscape

- Simulation semantics/scheduling semantics
  - Defined by the Verilog standard
  - Event-driven – signals propagate through circuits

- “Synthesis semantics”
  - Specified by legacy standard, no current standard
  - Kind of built on top of the simulation semantics
  - “Simulation-synthesis mismatches”

- “Cycle semantics”
  - A semantics for synchronous hardware
  - Not really specified anywhere but used everywhere
  - Used in formal methods: model checking, SAT/SMT reasoning, etc.
  - Simple!
  - Not event-driven!

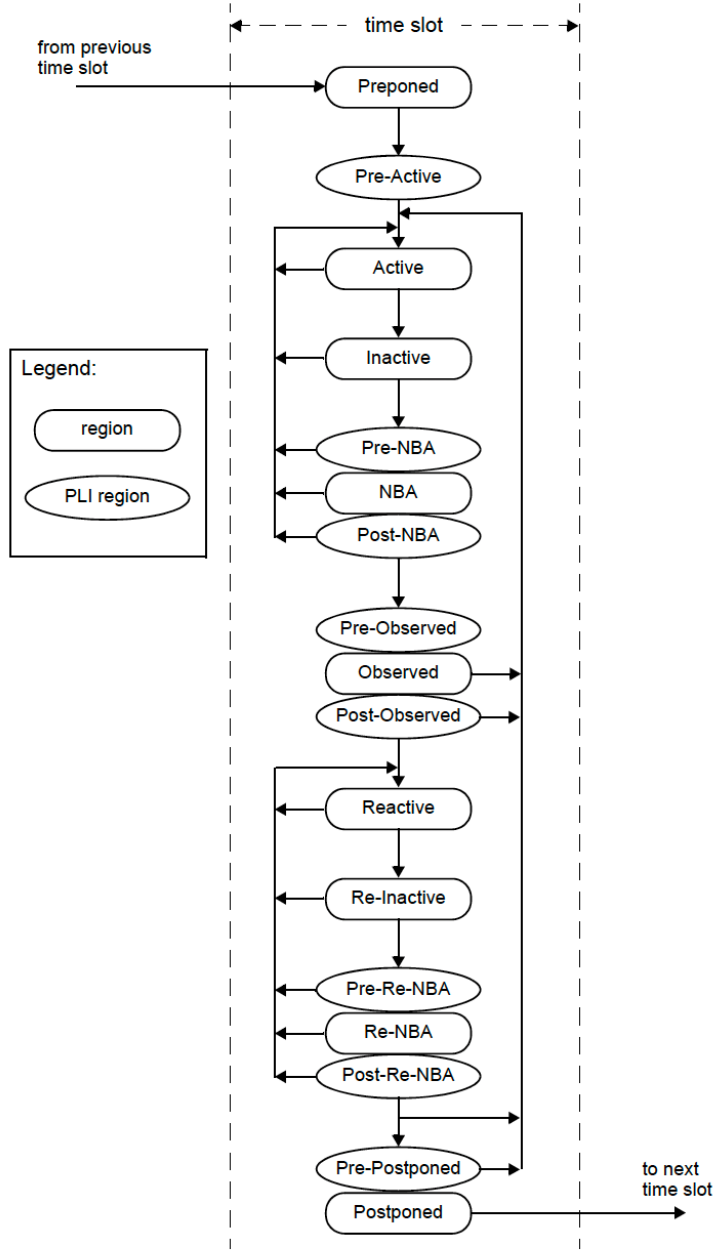
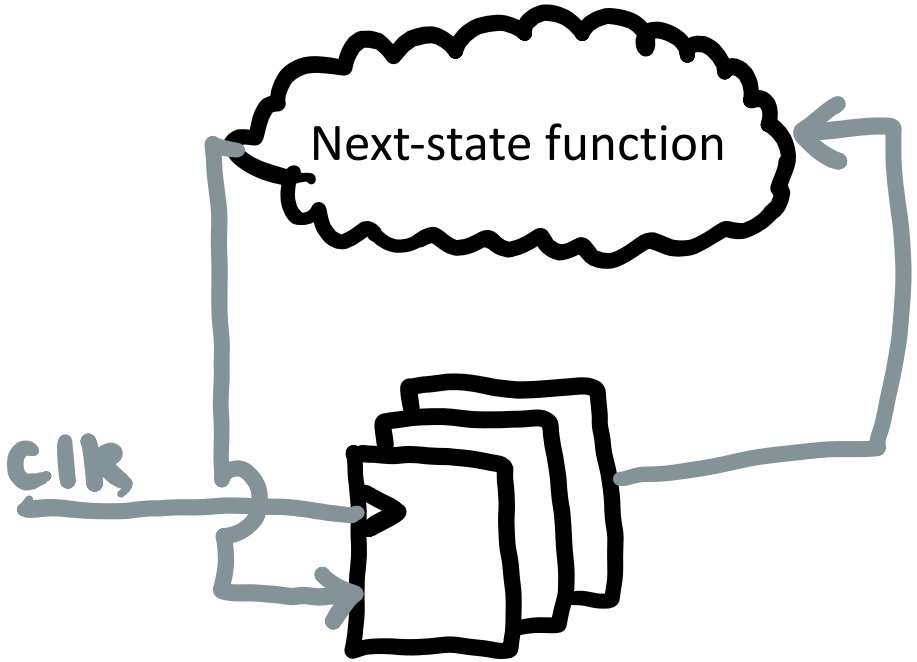


Figure 4-1—Event scheduling regions

Cycle semantics (i)

- (Synchronous) hardware = state machines
- Cycle semantics = cycle-by-cycle state
- I.e., just need to describe:
  - Initial state
  - Next-state function



Cycle semantics (ii)

```
logic foo = E1;
logic bar = E2;
// …
logic baz = En;
```

Memory elements and wires

```
always_comb C1
always_comb C2
// …
always_comb Cn
```

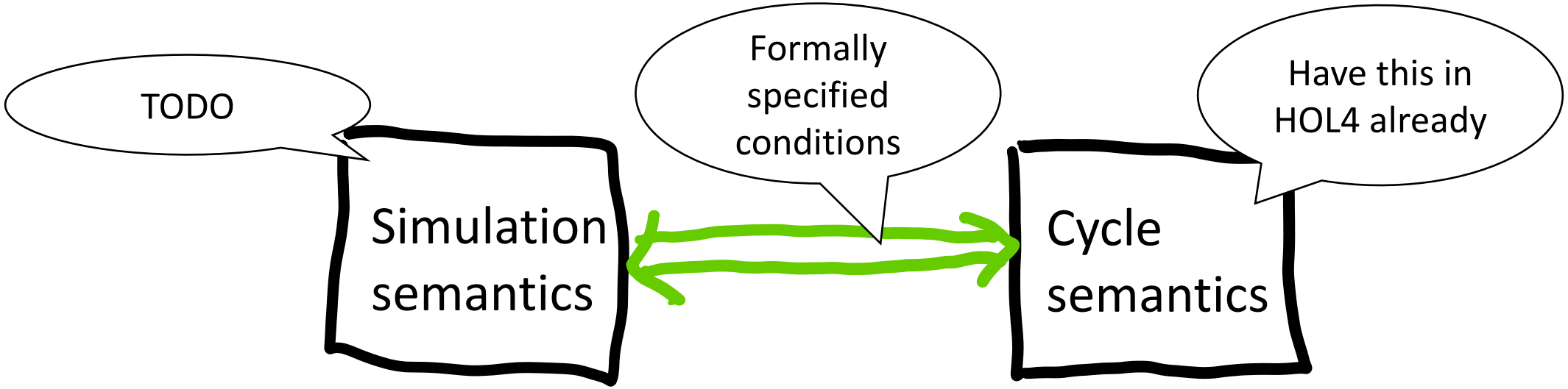
Combinational logic

```
always_ff @(posedge clk) C1
always_ff @(posedge clk) C2
// …
always_ff @(posedge clk) Cn
```

Sequential logic

- Initial state:
  - Set variables to initial values
  - Run **always_comb**
- Compute next state:
  - Run **always_ff**
  - Run **always_comb**
- Event queue: only need active and NBA region
- Preprocessing: sort all always blocks by dependencies, so we know in which order to run them in

Relating simulation semantics and cycle semantics



Some problems with the simulation semantics

- The usual problems associated with any prose standard
- Verilog is a language with various pitfalls and counterintuitive cases
- Verilog standard inconsistent with Verilog practice
- Verilog standard inconsistent with itself

Some **solutions for** the simulation semantics

- Highlight how the standard differs from
  - simulators,
  - assumptions made in real-world Verilog code,
  - and other sources that indirectly describes the semantics of Verilog.
- Icarus the main open-source simulator
- For the various closed-source simulators we can only do best guesses

Example 1: interleaving semantics

- Verilog standard[illegible] allowed to p[illegible]
- If followed, would break a lot of code
- Simulators do not preempt:
  - Icarus – does not preempt
  - Closed source simulators *seem* to not preempt

Informal semantics = run every time a or b is updated

```
logic a, b, c;

always @(a, b) a = b + c;

initial b = 0;

initial c = 0;
```

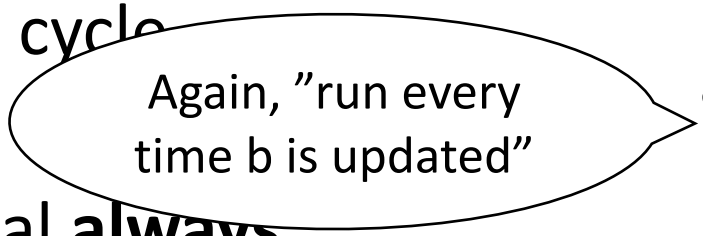
2. Wake up, do addition, and then preempt

1. Run!

3. Run!

Example 2: first-cycle always semantics

- Verilog standards after SystemVerilog state **always_comb** must run in the first cycle
- Leaves combinational **always** broken
- Actual simulators, e.g., Icarus, have special first-cycle semantics for **always**




```
logic a, b;

always @(b) a = b;

initial a = 5;
```

Example 3: nonblocking assignments

- Semantics described by both
  - Pseudocode
  - Prose text
- Pseudocode and prose text inconsistent
  - Maintain order of NBA events
  - Mixing NBA events events with other events
- Prose text inconsistent with itself
  - Different order guarantees for NBA events
- Icarus maintains all order and does not mix

Pseudocode: when all regions before NBA region, move NBA events to active region

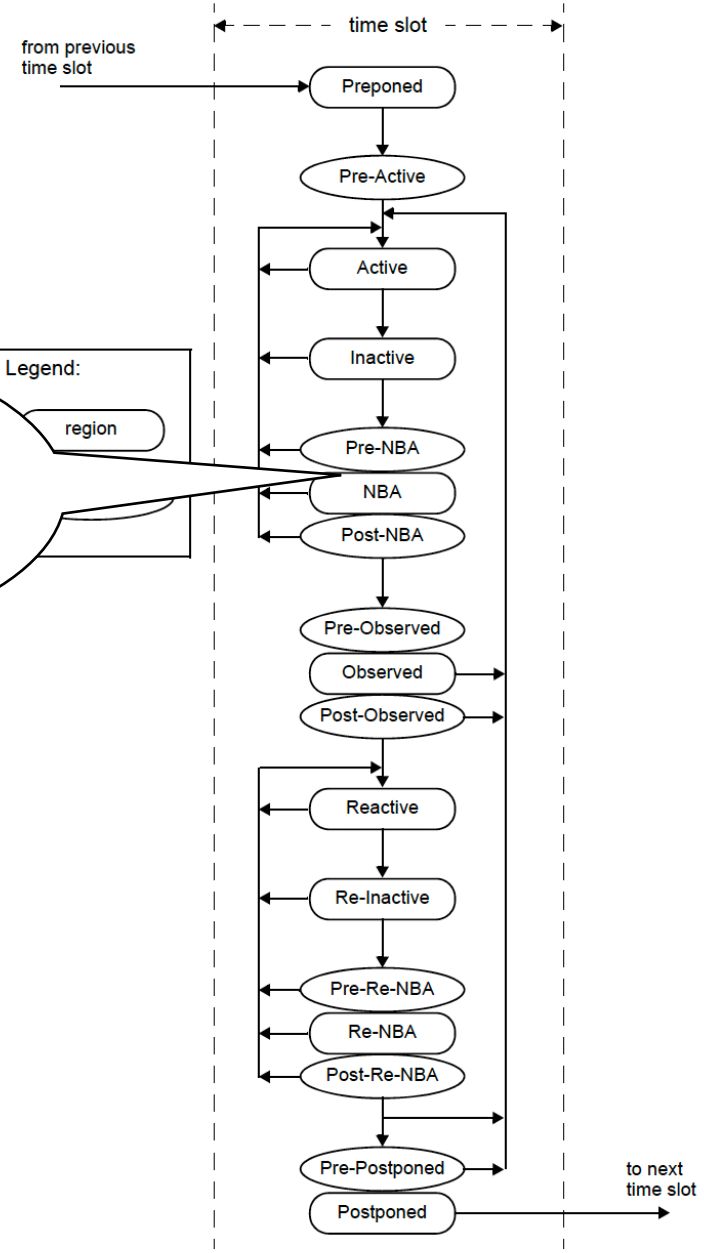


Figure 4-1—Event scheduling regions

Example 4: language pitfalls

- Does not matter much when doing formal methods – we only need a clearly specified semantics, does not matter if it is “sensible” or not

- Example of potential expression-level pitfall:
  - `a = 2’b11 + 2’b01`
  - Question: does this overflow?
  - Answer: depends on size of a – lhs also used to determine size of computation

Example 5: X values

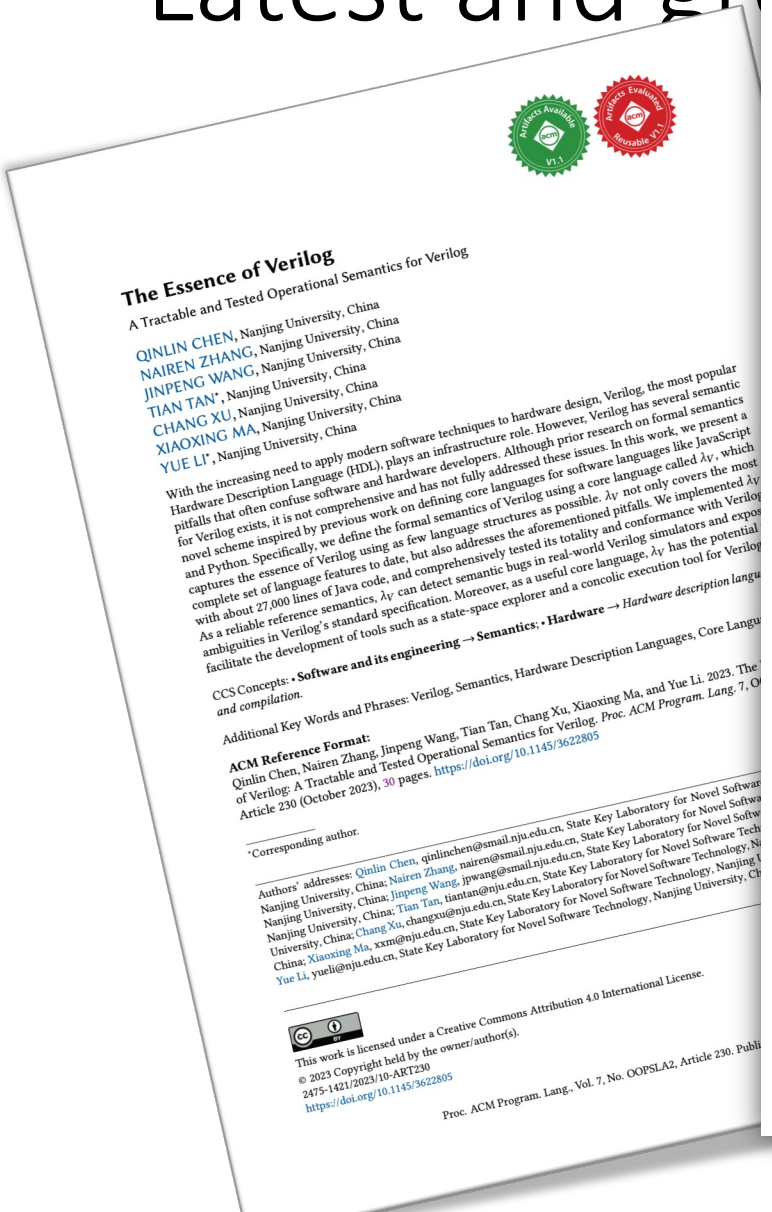
- X = unknown value
  (and a few other things)
- X values handled:
  - Underapproximate
  - Exact
  - Overapproximate

Code	Output
`if ('x) a = 1; else a = 0;`	
`a = 'x ? 1 : 0;`	
`a = 2'b00 + 2'b0x;`	

- Imho: language feature too broken to be used

- Easy solution in SAT/SMT/HOL4: free variables

Latest and gr



The Essence of Verilog

A Tractable and Tested Operational Semantics for Verilog

QINLIN CHEN, Nanjing University, China
NAIREN ZHANG, Nanjing University, China
JINPENG WANG, Nanjing University, China
TIAN TAN*, Nanjing University, China
CHANG XU, Nanjing University, China
XIAOXING MA, Nanjing University, China
YUE LI*, Nanjing University, China

With the increasing need to apply modern software techniques to hardware design, Verilog, the most popular Hardware Description Language (HDL), plays an infrastructure role. However, Verilog has several semantic pitfalls that often confuse software and hardware developers. Although prior research on formal semantic for Verilog exists, it is not comprehensive and has not fully addressed these issues. In this work, we present a novel scheme inspired by previous work on defining core languages for software languages like JavaScript and Python. Specifically, we define the formal semantics of Verilog using a core language called λ_V, which captures the essence of Verilog using as few language structures as possible. λ_V not only covers the most complete set of language features to date, but also addresses the aforementioned pitfalls. We implemented λ_V with about 27,000 lines of Java code, and comprehensively tested its totality and conformance with Verilog simulators and exposes ambiguities in Verilog's standard specification. Moreover, as a useful core language, λ_V has the potential to facilitate the development of tools such as a state-space explorer and a concolic execution tool for Verilog.

CCS Concepts: • **Software and its engineering** → **Semantics**; • **Hardware** → *Hardware description languages and compilation.*

Additional Key Words and Phrases: Verilog, Semantics, Hardware Description Languages, Core Languages

ACM Reference Format:
Qinlin Chen, Nairen Zhang, Jinpeng Wang, Tian Tan, Chang Xu, Xiaoxing Ma, and Yue Li. 2023. The Essence of Verilog: A Tractable and Tested Operational Semantics for Verilog. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 230 (October 2023), 30 pages. https://doi.org/10.1145/3622805

*Corresponding author.

Authors' addresses: Qinlin Chen, qinlinchen@smail.nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Nairen Zhang, nairen@smail.nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Jinpeng Wang, jpwang@smail.nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Tian Tan, tiantan@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Chang Xu, changxu@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Xiaoxing Ma, xxm@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China; Yue Li, yueli@nju.edu.cn, State Key Laboratory for Novel Software Technology, Nanjing University, China.

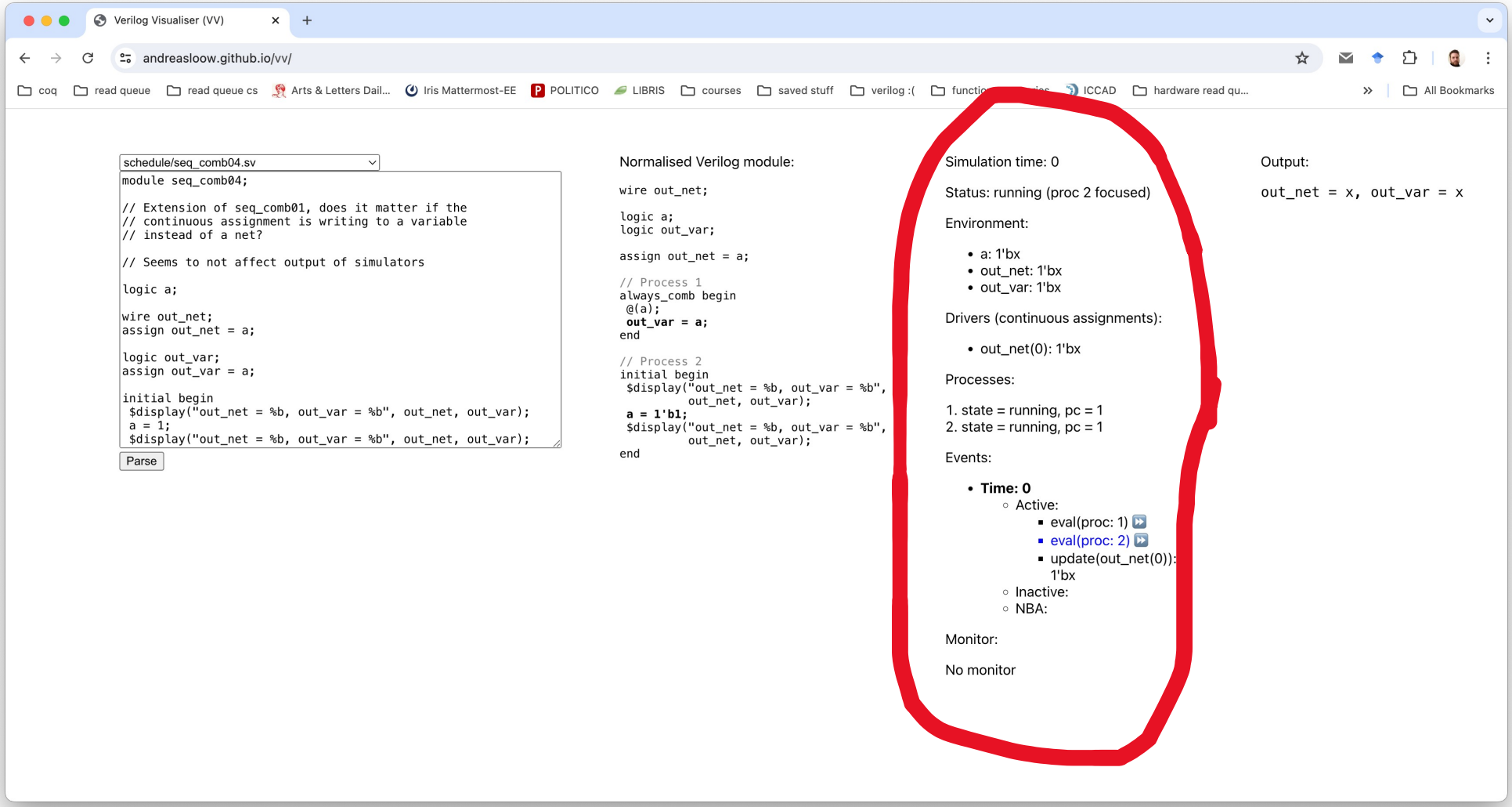
2475-1421/2023/10-ART230
https://doi.org/10.1145/3622805

Root Cause	Short Description	Cases	Number	Running Result of LV	Related Sections
iv bug	hidden data race	see scripts/data-race-cases.list	99	most passed, several depend on scheduling	Section 5.1.2 of the paper (Line 915-971) and Section 3.1.2 of this document
	loss of newest results	pr1662508.v	1	depend on scheduling	Section 3.1.3 of this document
	undefined initial output of delayed driver	ldelay1.v		failed	Section 5.2 of the paper (Line 983-997) and Section 3.2.1 of this document
				on scheduling	Section 3.2.2 of this document
					Section 3.2.3 of this document
					Section 3.2.4 of this document
erroneous case	using == rather than ===	casex3.9E.v	5	failed	Section 4.2 of the paper (Line 849-857) and Section 3.3 of this document
		casez3.10E.v			
		casez3.10D.v			
		casez3.10C.v			
	shared loop variable	br1000.v	1	depend on scheduling	Section 3.3 of this document
	arithmetic overflow	sqrt32.v	1	depend on generated random numbers	
stricter type check	implicit convertion of port direction	bufif.v	1	failed	Section 4.2 of the paper (Line 858-863) and Section 3.3 of this document
	inout with different netKinds	br1001.v	1	failed	

If you address the things I have highlighted here, almost all of these go away

Figure 1: Test cases within the Language Features Suite that λ_V fails to pass due to: (1) bugs in *Icarus Verilog* and *Verilator*; (2) ambiguities in Verilog; (3) erroneous cases; (4) our stricter type checker.

New visualisation of simulation semantics



That’s all

- There are some (rudimentary) HOL4 tool for Verilog development
- Initial work on mechanising the simulation semantics of Verilog
  - Goal: relating the cycle semantics and the simulation semantics
  - Work has been lying around for a while – need to write it down and submit…
- (Main current work:
  - Exact and underapproximate separation logic
  - Mechanisation of the multi-language symbolic-execution platform Gillian)