From Semantics to Software: Building a Verification Ecosystem for P4 using HOL4P4

Didrik Lundberg[1,2]

P4 Developer Days, September 25 2025

[1]KTH Royal Institute of Technology [2]Saab AB

Overview

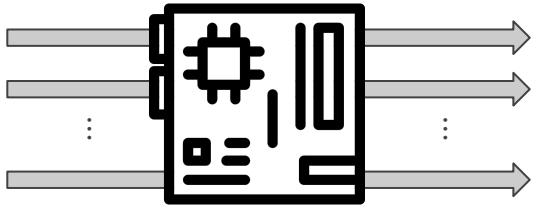
– – –

- Formal Verification
- HOL4P4
- Related Work
- Future Work and Conclusions

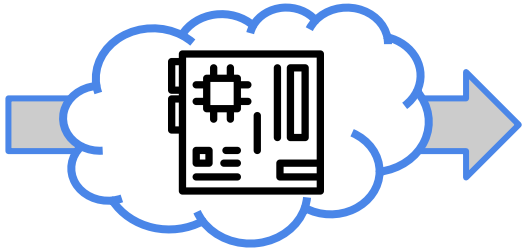
Formal Verification

Formal Verification

- Replaces testing with logical reasoning
- Requires a formal model



Testing vs. Formal Verification



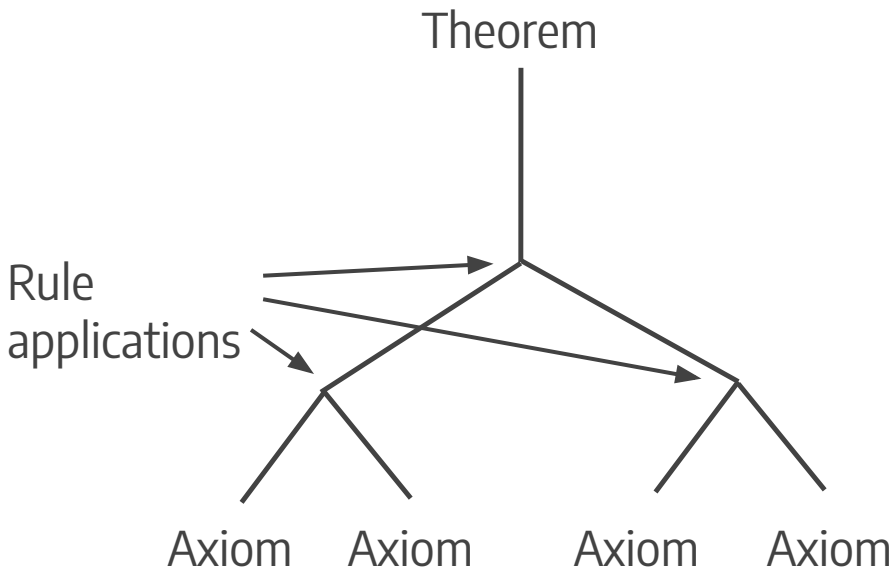
Interactive Theorem Proving

– – –

- What is a theorem?

Interactive Theorem Proving

- What is a theorem?



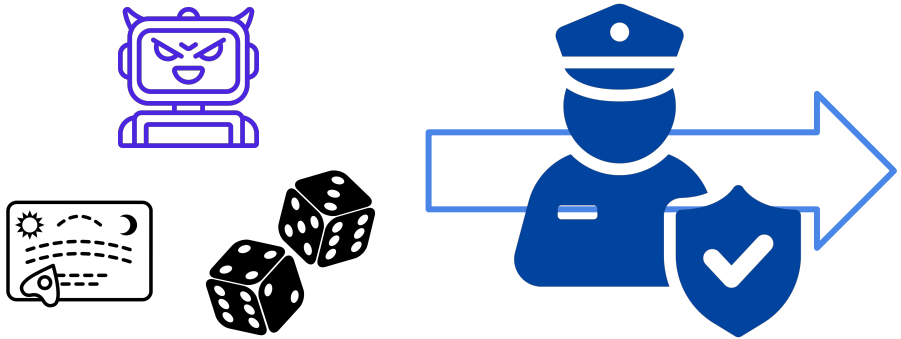
Interactive Theorem Proving

- What is a theorem?
- TCB consists of
  - Formal model
  - Proof-checking mechanism

Only valid proofs allowed

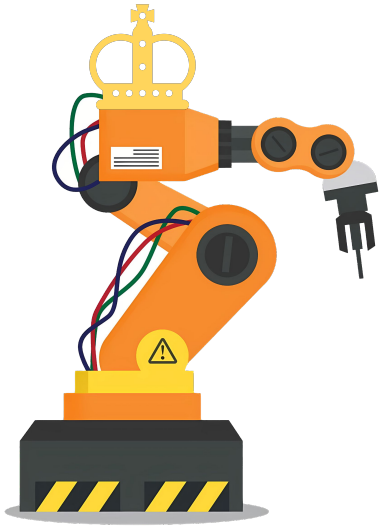
Interactive Theorem Proving

- What is a theorem?
- TCB consists of
  - Formal model
  - Proof-checking mechanism



Working with an ITP

- Automation is key



Working with an ITP

- Automation is key
- High assurance over bug-finding

HOL4P4

HOL4P4 Overview

A formalisation of P4 using the ITP HOL4.

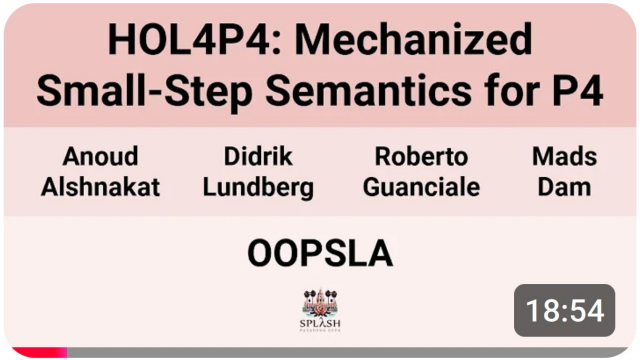
- Syntax
- Import tool
- Semantics
- Metatheory

HOL4P4 Overview

A formalisation of P4 using the ITP HOL4.

- Syntax
- Import tool
- Semantics
- Metatheory

For more details:



HOL4P4 Overview

A model of P4 formalised in the ITP HOL4.

- Syntax
- Import tool
- Semantics
- Metatheory

Verification toolbox

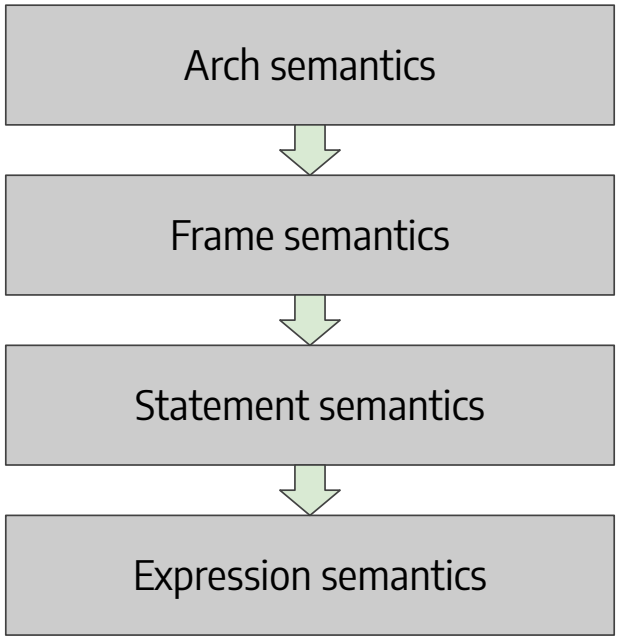
HOL4P4 Syntax: Expressions

e	$:= v$	value
	$\mid var$	variable
	$\mid \ominus\ e$	unary operation
	$\mid e_1\ \oplus\ e_2$	binary operation
	$\mid e[\overline{b} : \overline{b}']$	slicing
	$\mid e.f$	field access
	$\mid f(e_1,\ldots,e_n)$	function call
	$\mid$ **select** $e\ \{v_1 : st_1;\ldots;\ v_n : st_n\}\ st$	select
	$\mid \langle f_1 = e_1;\ldots;\ f_n = e_n \rangle$	struct
	$\mid (cast)e$	cast

HOL4P4 Syntax: Statements

$$
\begin{array}{llllll}
s & := \emptyset & \mid lv := e & \mid \textbf{if } e \textbf{ then } s \textbf{ else } s' & \mid \textbf{return } e & \mid s; s' \\
 & \mid \textbf{transition } e & \mid \textbf{apply } tbl\ e_1,\ldots,e_n & \mid \blacksquare & \mid \{(x_1,\tau_1),\ldots,(x_n,\tau_n)\ s\} &
\end{array}
$$

HOL4P4 Semantics

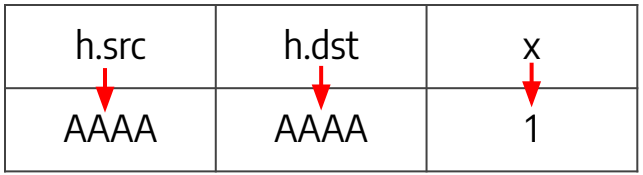


HOL4P4 State

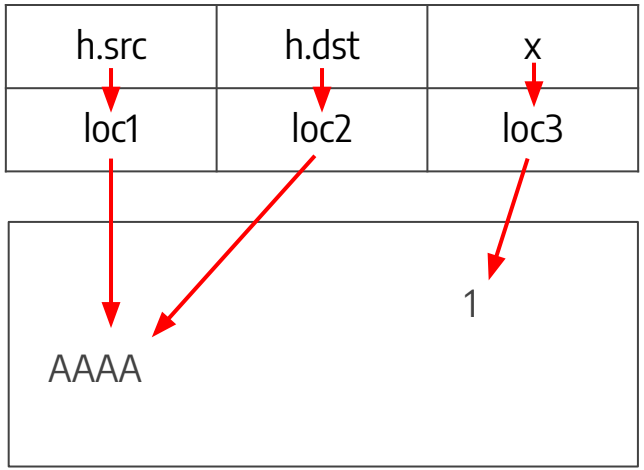
Top level: $(\overline{io}, \alpha, i, \overline{\gamma_G}, \overline{\Phi}, t)$

Heapless State

Heapless memory



Heap-based memory

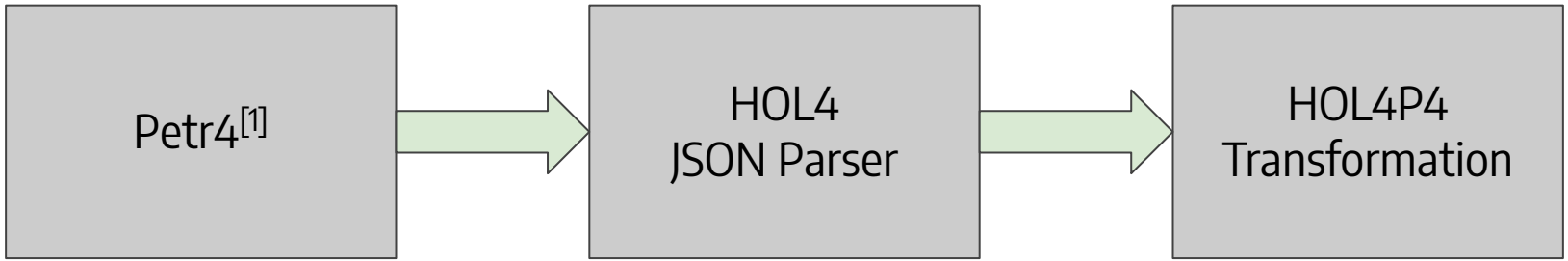


HOL4P4 State

Statement level: $(\overline{\chi}, \alpha, \overline{\gamma_G}, \overline{\Phi}, t)$

Expression level: Only reduces *e*, may push new Φ

HOL4P4 Import Tool



[1] Doenges, Ryan, et al. "Petr4: formal foundations for p4 data planes." POPL (2021)

HOL4P4 Metatheory: Type System Guarantees

Type preservation

$$st : \Gamma \wedge (E \vdash st \rightsquigarrow st') \implies st' : \Gamma$$

Progress

$$st : \Gamma \implies (E \vdash st \rightsquigarrow st') \vee \mathrm{final}(st)$$

Symbolic Execution

- Scope: any functional properties
  - “If P, then code successfully executes and Q holds afterward”
  - “If ingress port is n, and source MAC address is a, then egress port is not m”
  - Can refer to tables, extern data, ...
- Can overapproximate externs and tables
- Fully proof-producing
- Supports V1Model

Symbolic Executor: Usage

1. Import P4 program
2. Modify initial state and arguments to symbolic executor
3. Provide pre- and postcondition
4. Run!
5. (If proof fails: tweak the internal reasoning)
6. (If proof takes too long: break up into multiple proofs)

Symbolic Executor: Examples

- Test suite for simple properties
- Basic IPSec program
- Larger industry applications

Symbolic Executor: Benchmarks

- LoC<100, branches<5: seconds
- LoC<1000, branches<20: ~15 minutes
- LoC<10000, branches<20 per block: must be split up, 1-2 hours

Work in progress

Verified Compilation to SW Switch

- Uses CakeML
- Compiles the HOL4P4 semantics
- CakeML wrapper for system calls
- Supports V1Model and eBPF

HOL4P4 SW Switch: Usage

1. Import P4 program
2. Modify initial state to contain desired tables et.c.
3. Compile!

HOL4P4 SW Switch: Examples

– – –

- fabric_border_router.p4 (~3000 LoC)
- vss-example.p4
- Some smaller examples

HOL4P4 SW Switch: Benchmarks

– – –

- Slower than BMv2, faster than petr4 [CAV ‘21]
- vss-example.p4: ~20 Mbps throughput, 1.5 ms latency
- fabric_border_router.p4: ~2 Mbps throughput, 16.5 ms latency
- Experiments show direct compilation could beat BMv2

Symbolic Execution + Verified Compilation

How can these tools be used together?

- Verified compiler preserves SE guarantees
- SE optimizes program before compilation

Related Work

Related Work

– – –

- Petr4 [POPL ‘21]
  - Later formalized in Rocq
  - Heap-based semantics
  - No verification tool

Related Work

– – –

- Petr4 [POPL ‘21]
  - Later formalized in Rocq
  - Heap-based semantics
  - No verification tool
- Verifiable P4 [ITP ‘23]
  - More similar to HOL4P4
  - More details in Qinshi Wang’s and Mengying Pan’s theses
  - 2024 preprint presents complete verification toolbox

Future Work and Conclusions

Future Work: P4ncake

———

Pancake:

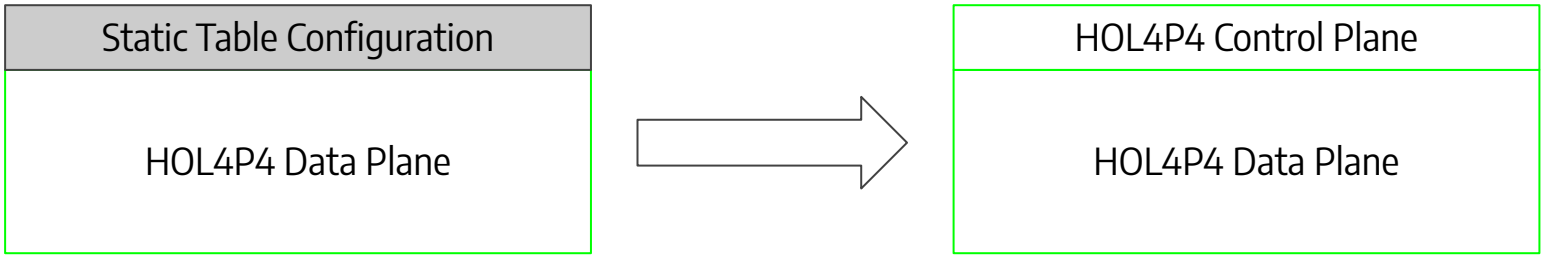
- C-like systems language
- Has verified compiler
- Verified device drivers for LionsOS

P4+Pancake=P4ncake

Direct compilation faster than interpreting

Future Work: Verified Control Plane

- HOL4P4 semantics designed for interleaving
- Enables proofs about DP+CP interplay
- Enables software switch with dynamic control plane



Current solution

Future solution

Conclusions

You've learned:

- How to minimize TCB with theorem proving
- How the HOL4P4 symbolic executor can be used
- How the HOL4P4 software switch can be used
- How to compare HOL4P4 to related work

Questions?