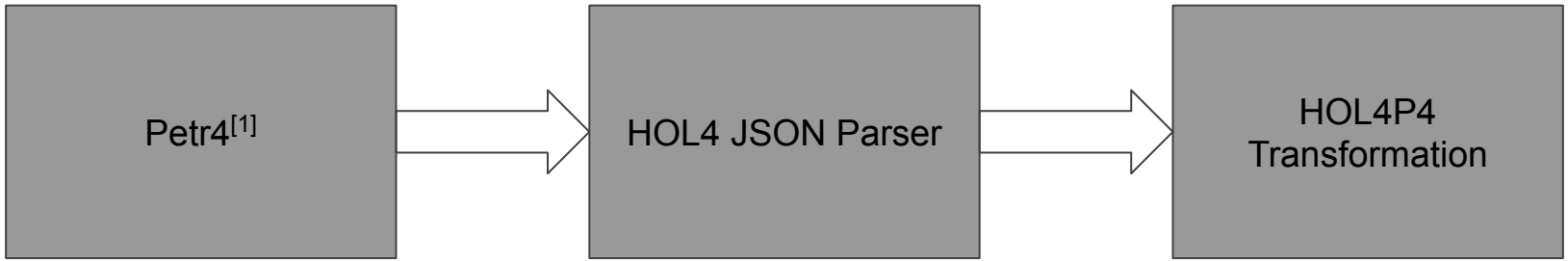
P4 Executable Semantics and Symbolic Execution

Didrik Lundberg

June 25 2024

Import Tool

Import Tool



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

HOL4 JSON Parser

– – –

- Serialiser and deserialiser
- ~500 LoC in HOL4
- Can be compiled with CakeML
- Validated to obey IETF standard using SotA test suite[2]

```
Datatype:
  json =
     Object ((string # json) list)
   | Array (json list)
   | String string
   | Number (sign # num) (num option)
       ((sign # num) option)
   | Bool bool
   | Null
End
```

[2] N. Seriot, “JSON Parsing Test Suite.” https://github.com/nst/JSONTestSuite (2023)

HOL4P4 Transformation

– – –

- From HOL4 JSON to HOL4P4 representation
- ~5000 LoC in HOL4
- Type inference of constants
- Program transformations
  - In-lining of nested blocks
  - Desugaring
- Custom option/exception datatypes

Executable Semantics

Executable Semantics

- Used to evaluate *n* steps in the semantics
- Written to be CakeML-adjacent
- Soundness proved
  - ~1500 LoC
- Validated on standard test program suite

```
premises ⇒
rel_sem env st st’

vs.

exec_sem env st = st’
```

Symbolic Execution

Symbolic Execution

- Note: Overapproximating!
- Remember: No loops* in P4
- HOL4 free variables as bits
  - … doesn't evaluate expressions with symbolic bits
- Path condition in assumption
- List of n-step thms as paths
- Naïve version quick to implement

```
path_cond ⇒
exec_sem env st(b1, b2, …) =
st'(b1, b2, …)
```

Symbolic Branches: Conditional

- If-statements with symbolic expressions
  - `stmt_cond ((e_var “a”) binop_and (e_var “b”)) stmt1 stmt2`
    reduced to
    `stmt_cond (a /\ b) stmt1 stmt2`
    results in
    `a /\ b ⇒ stmt_cond T stmt1 stmt2`
    `~(a /\ b) ⇒ stmt_cond F stmt1 stmt2`

Symbolic Branches: Table Application

- Table application
  - … with known table configuration:

```
stmt_app "table1" [a]

[(“table1”,
   [1 |-> set_out_port(1),
    2 |-> set_out_port(2),
    3 |-> set_out_port(3)])]
```

adds to path conditions

a=1, **a**=2, **a**=3,{1,2,3}∌**a**

Symbolic Branches: Table Application

- Table application
  - … with unknown table configuration:

```
stmt_app "table1" [a]

tables
```

  - Must introduce fresh free variables:

```
?b1 b2 b3 b4. set_out_port(bitv [b1; b2; b3; b4])
```

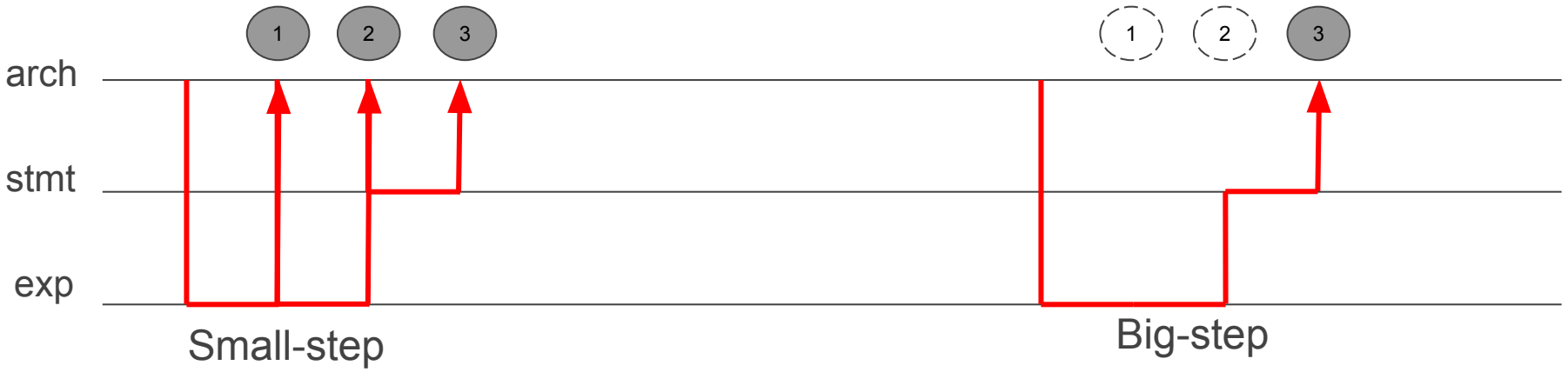
Overapproximation

- Extern functions
  - e.g. implementations of CRC-16 and IPv4 header checksum use copious word operations

```
?b1 b2 b3 b4. compute_checksum (bitv [a1; T; T; F]) =
  bitv [b1; b2; b3; b4]
```

Big-Step Semantics

- Expressions: No function calls
- Statements: Only Seq, Assign
- Avoids layer transitions of the regular semantics



Big-Step Semantics

– – –

- Implementation size
  - ~500 LoC
  - Soundness ~5000 LoC
- On real-world programs, from ~3s to ~10ms per reduction
- Bespoke semantics versions, esp. incipits.
  - Minimizing the environment and state

Scalability: Numbers

	Small	Medium[3]	Large
.p4 size	~100 LoC	~500 LoC	~10000 LoC
HOL4P4 size	~200 LoC	~1200 LoC	~20000 LoC
# Paths	12	20	*
Time	30s	12m30s	Hours

[3] F. Hauser et al., "P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN," in IEEE Access (2020)

Scalability: Bottlenecks

- Bottlenecks
  - **`CBV_CONV`** performing complex reductions (function call, …)

`EVAL' ``exec env st``

where `env` and `st` *huge*

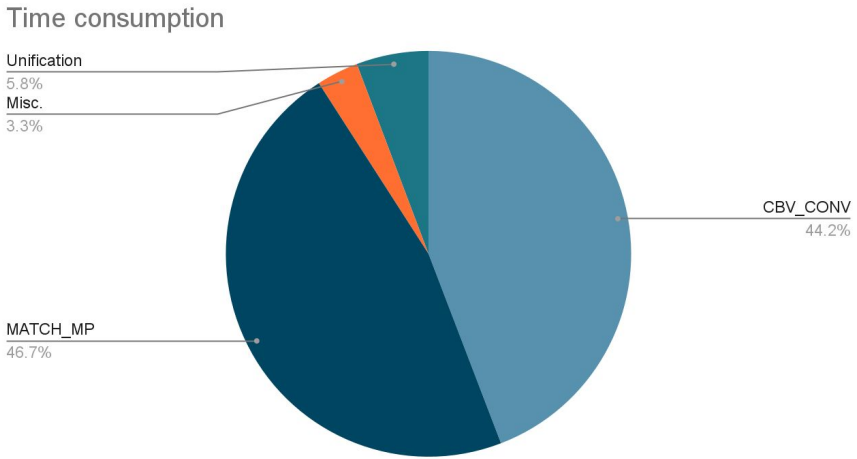
Scalability: Bottlenecks

- Bottlenecks
  - **`CBV_CONV`** performing complex reductions (function call, …)
  - **`MATCH_MP`** in composition of execution

```
(path_cond ⇒
  exec env st n = st') ⇒
exec env st' = st'' ⇒
(path_cond ⇒
  exec env st n = st'')
```

Scalability: Bottlenecks

- Bottlenecks
  - `CBV_CONV` performing complex reductions (function call, …)
  - `MATCH_MP` in composition of execution



Concurrency?

– – –

- Stable enough for benchmarking
- Doesn’t scale well with core count
  - ~40% speedup of IPsec example on 4-core laptop
  - ~30% speedup on 16-core stationary

Future Work

- Verilog from P4
- Control Plane in CakeML

Questions?