
# P4-SpecTec

## Mechanized Language Definition for P4

Jaehyun Lee*, Yechan Woo*, Nate Foster†, Sukyoung Ryu*

KAIST*, Cornell University†

# P4 Language from a Programming Language POV

There are mainly **three representations** of the P4 programming language.

## 1. Official Spec

THE definition in natural language and code

## 2. Implementations

Executable representation (p4c, Bmv2, ...)

## 3. Formal Spec

Precise mathematical description
(Petr4, P4Light, HOL4P4, ...)

# Despite in Different Forms,

Representing the same underlying concept: **the syntax and semantics of P4**.

1. Official Spec

in natural language

2. Implementations

in programming language (C++, ...)

3. Formal Spec

in mathematical notations

# QUESTION: Are they consistent with each other?

They should be, in principle.

| Definition | | Execution | | Guarantees |
|---|---|---|---|---|
| 1. Official Spec | = | 2. Implementations | = | 3. Formal Spec |

But in practice,

- manual maintenance
- feature updates
- ambiguities in the official spec

cause discrepancy among the representations.

# Official Spec versus Implementations

## Name duplication and name shadowing #974

Closed **MollyDream** opened this issue on Oct 22, 2021 · 3 comments

**Do we accept this?**

```
void foo() {
  bit<32> x;
  bit<32> x;
}
```

1. Official Spec

no restriction on duplicate local declaration

2. Implementations

p4c rejects duplicate local declaration

# Official Spec versus Formal Specs

## 1. Official Spec

defines the complete set of language features

## 3. Formal Spec

often omit some feature in their core calculi

- parser block semantics
- type inference algorithm
- cast insertion

# Worse, the language evolves

## 1. Official Spec

**A.3. Summary of changes made in version 1.2.2, released May 17, 2021**

- Added support for accessing tuple fields (Section 8.12).
- Added support for generic structures (Section 7.2.11).

introduced generic structs and headers (v1.2.2)

## 2. Implementations

Compiler Bug: `Could not find type of <Type_Header>` `...` on specialized generic struct type #4835

Open jaehyun1ee opened this issue on Jul 26 · 7 comments

p4c bug related to generic structs used with type inference

## 3. Formal Spec

unsupported (undefined) in all existing formalizations

i.e., the **gap widens as the language evolves**

# Other Language with a Similar Issue

WebAssembly (Wasm) strives to maintain consistency among representations.

For a new feature to be standardized, the committee requires four artifacts:


1. Formal Spec
2. Prose Spec

semantics in both mathematics and prose pseudocode

3. Reference Interpreter in OCaml

4. Test Suite for the new feature

# The Wasm Standard: Formal and Prose Spec

## 2. Prose Spec step-by-step pseudocode-style explanation

$t.binop$

1. Assert: due to validation, two values of value type $t$ are on the top of the stack.
2. Pop the value $t.\mathsf{const}\ c_2$ from the stack.
3. Pop the value $t.\mathsf{const}\ c_1$ from the stack.
4. If $binop_t(c_1, c_2)$ is defined, then:
    a. Let $c$ be a possible result of computing $binop_t(c_1, c_2)$.
    b. Push the value $t.\mathsf{const}\ c$ to the stack.
5. Else:
    a. Trap.

$$
\begin{array}{lcll}
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & (t.\mathsf{const}\ c) & (\text{if } c \in binop_t(c_1, c_2)) \\
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & \mathsf{trap} & (\text{if } binop_t(c_1, c_2) = \{\})
\end{array}
$$

## 1. Formal Spec execution rules as formal reduction rules

# A Day in the Life of a Language Designer

```
.. _exec-binop:

:math:`t\K{.}\binop`
....................

1. Assert: due to :ref:`validation <valid-binop>`, two values of :ref:`\

2. Pop the value :math:`t.\CONST~c_2` from the stack.

3. Pop the value :math:`t.\CONST~c_1` from the stack.

4. If :math:`\binopF_t(c_1, c_2)` is defined, then:

   a. Let :math:`c` be a possible result of computing :math:`\binopF_t(c

   b. Push the value :math:`t.\CONST~c` to the stack.

5. Else:

   a. Trap.

.. math::
   \begin{array}{lcl@{\qquad}l}
   (t\K{.}\CONST~c_1)~(t\K{.}\CONST~c_2)~t\K{.}\binop &\stepto& (t\K{.}\
     & (\iff c \in \binopF_t(c_1,c_2)) \\
   (t\K{.}\CONST~c_1)~(t\K{.}\CONST~c_2)~t\K{.}\binop &\stepto& \TRAP
     & (\iff \binopF_{t}(c_1,c_2) = \{\})
   \end{array}
```

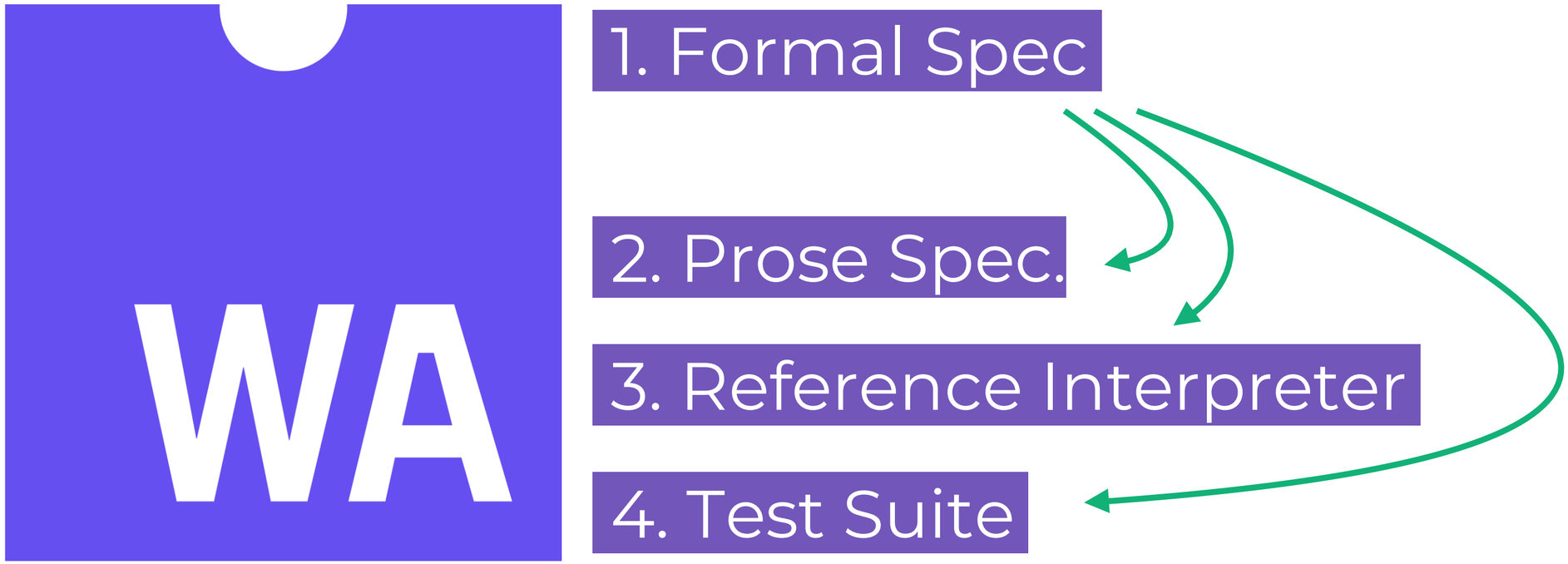
```
| Binary binop, Num n2 :: Num n1 :: vs' ->
  (try Num (Eval_num.eval_binop binop n1 n2) :: vs', []
  with exn -> vs', [Trapping (numeric_error e.at exn) @@ e.at])
```

```
(assert_return (invoke "shr_s" (i32.const 1) (i32.const 1)) (i32.const 0))
(assert_return (invoke "shr_s" (i32.const 1) (i32.const 0)) (i32.const 1))
(assert_return (invoke "shr_s" (i32.const -1) (i32.const 1)) (i32.const -1))
(assert_return (invoke "shr_s" (i32.const 0x7fffffff) (i32.const 1)) (i32.const 0x3fffffff))
(assert_return (invoke "shr_s" (i32.const 0x80000000) (i32.const 1)) (i32.const 0xc0000000))
(assert_return (invoke "shr_s" (i32.const 0x40000000) (i32.const 1)) (i32.const 0x20000000))
(assert_return (invoke "shr_s" (i32.const 1) (i32.const 32)) (i32.const 1))
(assert_return (invoke "shr_s" (i32.const 1) (i32.const 33)) (i32.const 0))
```

(left) formal and prose spec in raw text
(right) interpreter and test suite

a laborious and error-prone task: **consistency is also an issue in Wasm**

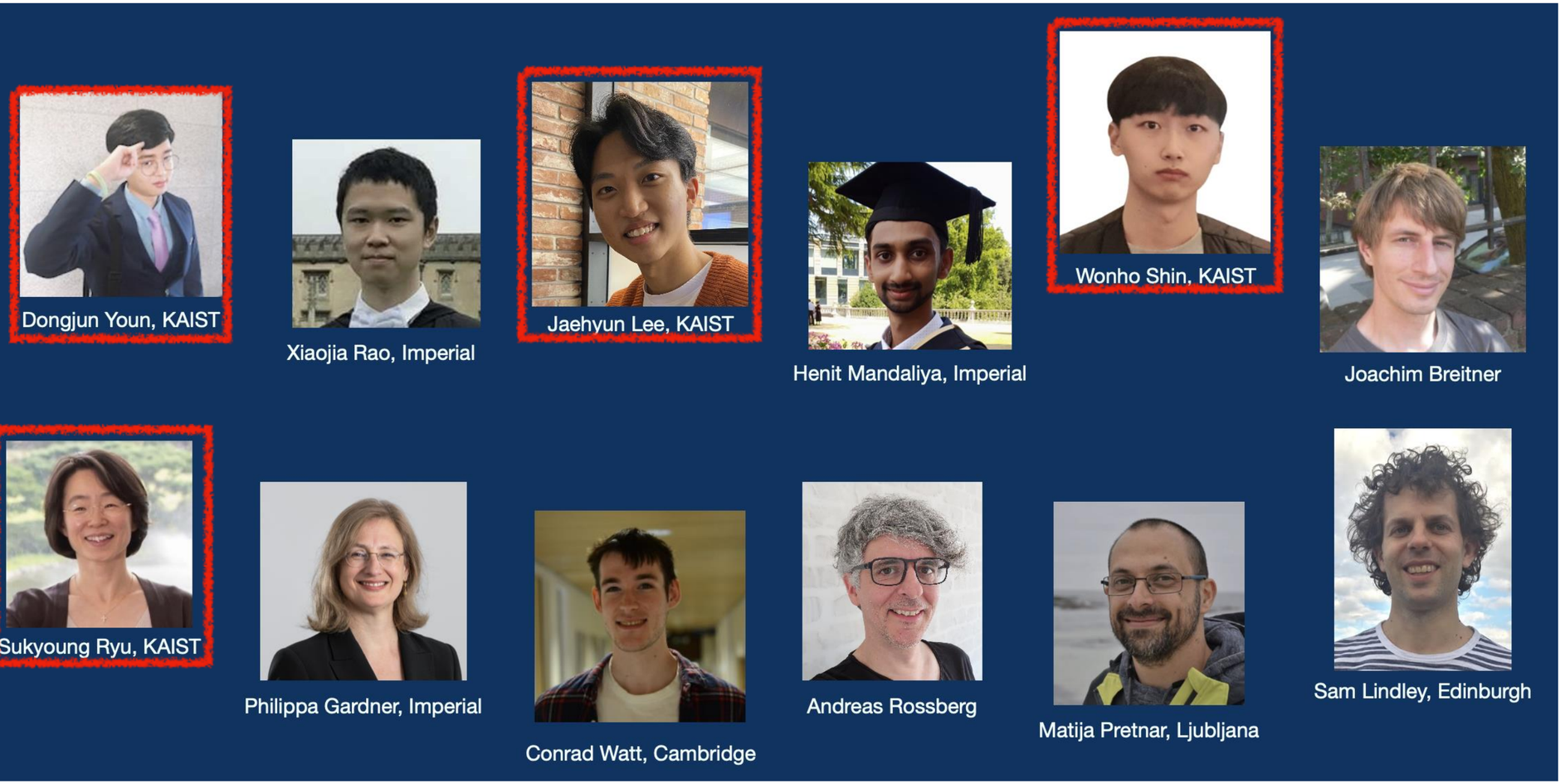
# Observation: What is the source of truth?




From a Programming Language POV, all originates from … the **formal spec**.

**Unambiguous** and **Precise** (than prose), yet **Abstract** (than OCaml).

# A Toolchain Using Formalism as the Source of Truth




Suhyeon Ryu

Hoseong Lee

Hyunhee Kang

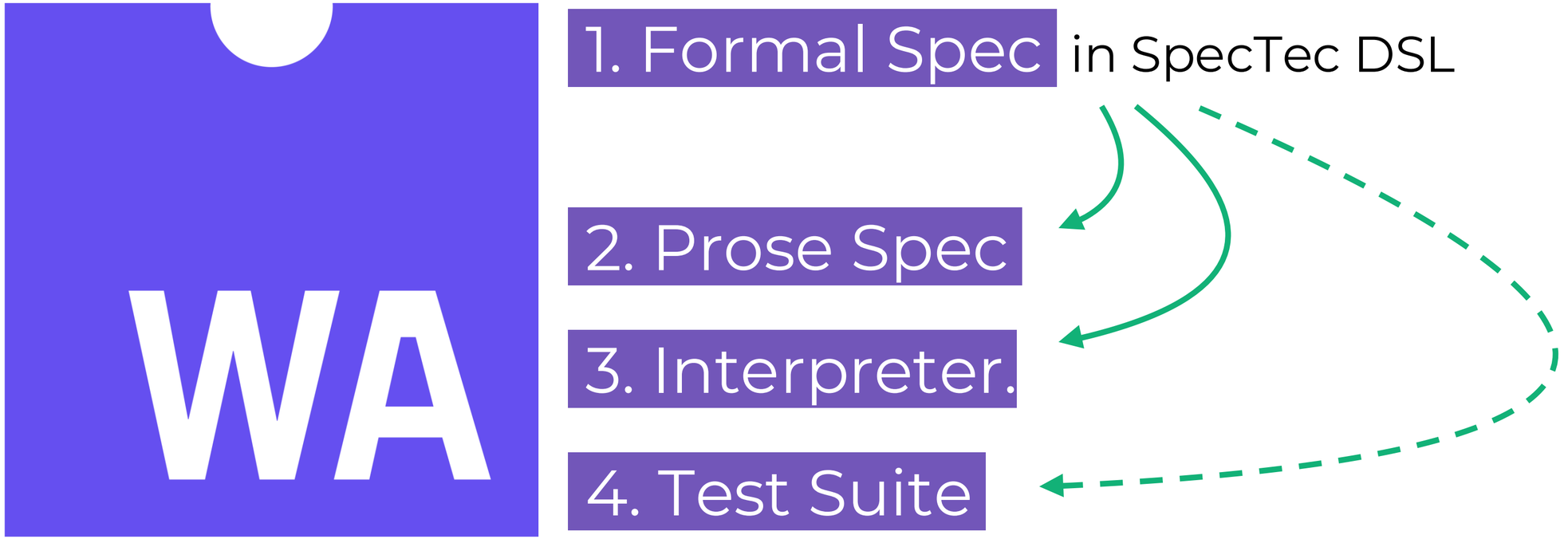
Can we automate standardization with the formal spec?

# SpecTec for WebAssembly Standard

SpecTec is a framework for mechanizing the Wasm spec.

(1) Specify the formal Wasm syntax and semantics in SpecTec DSL.

(2) SpecTec auto-generates various backends from that **single source of truth**.



# Input: Formal Spec, the Single Source of Truth

(1) **Specify all** Wasm 2.0 formal syntax and semantics in SpecTec DSL.

$$(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop \quad \hookrightarrow \quad (t.\mathsf{const}\ c) \quad (\text{if } c \in binop_t(c_1, c_2))$$

```
rule Step_pure/binop-val:
    (CONST nt c_1) (CONST nt c_2) (BINOP nt binop) ~> (CONST nt c)
    — if c <- $binop_(nt, binop, c_1, c_2)

rule Step_pure/binop-trap:
    (CONST nt c_1) (CONST nt c_2) (BINOP nt binop) ~> TRAP
    — if $binop_(nt, binop, c_1, c_2) = eps
```

ASCII representation of the formal notations.

More, the definitions are type-checked to prevent human errors.

# Output: Auto-Generated Representations

(2) SpecTec **auto-generates** various backends from that **single source of truth**.

**1. Formal Spec** in LaTeX **2. Prose Spec** in reStructuredText

*nt.binop*

1. Assert: Due to validation, a value of value type $numtype_0$ is on the top of the stack.
2. Pop the value $(numtype_0.\text{const}\ c_2)$ from the stack.
3. Assert: Due to validation, $numtype_0$ is $nt$.
4. Assert: Due to validation, a value of value type $numtype_0$ is on the top of the stack.
5. Pop the value $(numtype_0.\text{const}\ c_1)$ from the stack.
6. If $|binop_{nt}(c_1, c_2)|$ is less than or equal to 0, then:
    a. Trap.
7. Let $c$ be an element of $binop_{nt}(c_1, c_2)$.
8. Push the value $(nt.\text{const}\ c)$ to the stack.

$$[\text{E-BINOP-VAL}]\ (nt.\text{const}\ c_1)\ (nt.\text{const}\ c_2)\ (nt.binop) \hookrightarrow (nt.\text{const}\ c) \quad \text{if } c \in binop_{nt}(c_1, c_2)$$

$$[\text{E-BINOP-TRAP}]\ (nt.\text{const}\ c_1)\ (nt.\text{const}\ c_2)\ (nt.binop) \hookrightarrow \text{trap} \quad \text{if } binop_{nt}(c_1, c_2) = \epsilon$$

**3. Interpreter** passing against **all** (49833 tests) of the official test suite

**4. Test Suite** a Work-In-Progress

# SpecTec Helps the Language Ecosystem

## Specification Bug Prevention

- Injected 13 retrospective spec bugs into SpecTec, all were detected.
- Detected 10 bugs in feature proposals.


## Forward Compatibility

- Applied SpecTec to 5 proposals.
- SpecTec can support fast prototyping for language extensions.

# Current Status of SpecTec

Meeting note from Wasm Community Group Meeting on June 2024,

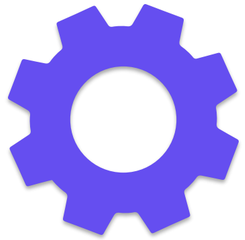
> Poll: Adopt SpecTec (once it’s ready) as the toolchain for authoring the spec.
>
> SF: 19 in room, 5 on chat F: 13 in room, 4 online N: 4 in room, 4 online A: 1 in room SA: 0
>
> DS: Consensus.

Polishing the tool, working on test generation & theorem prover backends.

# Recap: Three Representations of P4 Language

1. Official Spec
2. Implementations
3. Formal Spec


**SpecTec**



1. Formal Spec
2. Prose Spec
3. Reference Interpreter
4. Test Suite

# Idea: P4-SpecTec for the P4 Language Infrastructure

(1) Specify the formal P4 syntax and semantics in SpecTec DSL.

(2) Auto-generate various backends from that **single source of truth**.

3. Formal Spec with complete set of features written in SpecTec DSL

1. Official Spec comprehensible document

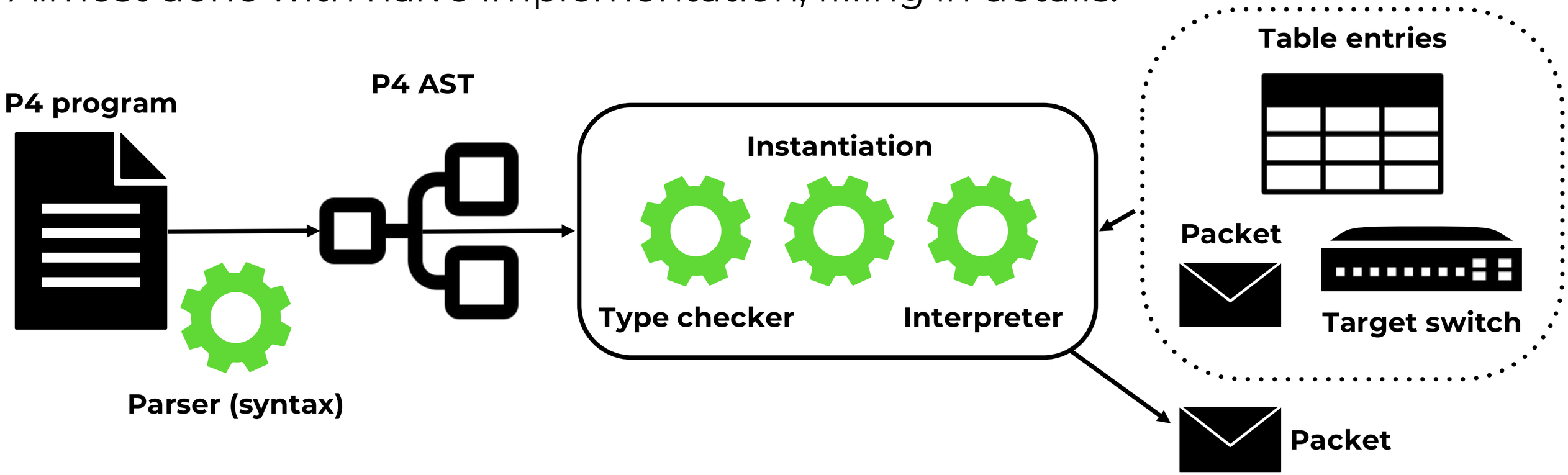
2. Implementations lightweight interpreter

# Actually, We Need an Initial Step

**(1) Make a formal P4 syntax and semantics definition.**

Building a complete OCaml model of P4 based on Petr4, P4Light, and HOL4P4.

Almost done with naive implementation, filling in details.



# “Complete” Formalization Invites Questions & Clarity

## Identifiers declared as constants are local compile-time known #1307

Merged **jonathan-diloren...** merged 1 commit into `p4lang:main` from `jaehyun1ee:main` 13 hours ago

## Do we allow `string` type to be aliased by `typedef`? #1293

Closed **jaehyun1ee** opened this issue on Jul 29 · 1 comment · Fixed by #1296

## Add explanation on passing directionless, extern object argument #1278

Merged **jnfoster** merged 1 commit into `p4lang:main` from `jaehyun1ee:main` on May 14

**Jae Hyun Lee** EDITED

What is the range of HM type inference in P4?

Hi @Ryan Doenges (guest) , I have a question about the type inference algorithm used in Petr4.

As of my knowledge, Petr4 uses local type inference rather than the general Hindley-Milner style global type inference by unification. Today I’ve thought of an example that may be an evidence that P4 actually needs global type inference. Could you take a look at it and see if it makes sense?

# A Proof of Concept Prototype

## (2) Specify the formal P4 syntax and semantics in SpecTec DSL.

A prototype based on the OCaml model.

Semantics of if statement in the official spec

### 12.6. Conditional statement

The conditional statement uses standard syntax and semantics familiar from many programming languages.

However, the condition expression in P4 is required to be a Boolean (and not an integer).

# A Proof of Concept Prototype

## (2) Specify the formal P4 syntax and semantics in SpecTec DSL.

A prototype based on the OCaml model.

Syntax of statements

```
syntax stmt =
  | I_EMPTY                    hint(show EMPTY)
  | I_ASSIGN expr expr         hint(show % := %)
  | I_SWITCH expr swcase*      hint(show SWITCH (%) %)
  | I_IF expr stmt stmt        hint(show IF
```

```
rule Interp_stmt/i_if-cont-tru:
  S CONT C |- I_IF expr_c stmt_t stmt_f : sig C''
  -- Interp_expr: S C |- expr_c : C' val_c
  -- if V_BOOL true = val_c
  -- Interp_stmt: S CONT C' |- stmt_t : sig C''
```

Dynamic semantics of if statement

# Generating a Formal Spec in PDF

**(3) Auto-generate various backends from that single source of truth.**

LaTeX backend already works on the fly.

SpecTec DSL

```
rule Interp_stmt/i_if-cont-tru:
  S CONT C |- I_IF expr_c stmt_t stmt_f : sig C''
  -- Interp_expr: S C |- expr_c : C' val_c
  -- if V_BOOL true = val_c
  -- Interp_stmt: S CONT C' |- stmt_t : sig C''
```

Generated LaTeX document

$$\frac{S\ C \vdash expr_c : C'\ val_c \quad \text{vbool true} = val_c \quad S\ \text{cont}\ C' \vdash stmt_t : sig\ C''}{S\ \text{cont}\ C \vdash \text{if}\ (expr_c)\ stmt_t\ stmt_f : sig\ C''}$$

# Generating a Formal Spec in PDF

## (3) Auto-generate various backends from that single source of truth.



### 2.7 Declarations

$$
\begin{array}{rcl}
\mathit{decl} & ::= & \mathsf{const}\ \mathit{type}\ \mathit{id} = \mathit{expr} \\
& | & \mathit{type}\ \mathit{id} = \mathit{expr}^{?} \\
& | & \mathit{type}\ \mathit{id}\ (\mathit{arg}^{*})\ \mathit{block}^{?} \\
& | & \mathsf{error}\ \mathit{field}^{*} \\
& | & \mathsf{matchkind}\ \mathit{field}^{*} \\
& | & \mathsf{struct}\ \mathit{id}\ (\mathit{field}, \mathit{type})^{*} \\
& | & \mathsf{header}\ \mathit{id}\ (\mathit{field}, \mathit{type})^{*} \\
& | & \mathsf{header\_union}\ \mathit{id}\ (\mathit{field}, \mathit{type})^{*} \\
& | & \mathsf{enum}\ \mathit{id}\ \mathit{field}^{*} \\
& | & \mathsf{enum}\ \mathit{id}\ \mathit{type}\ (\mathit{field}, \mathit{expr})^{*} \\
& | & \mathsf{type}\ \mathit{type}^{?}\ \mathit{decl}^{?}\ \mathit{id} \\
& | & \mathsf{typedef}\ \mathit{type}^{?}\ \mathit{decl}^{?}\ \mathit{id} \\
& | & \mathsf{value\_set}\ \{\mathit{type}\}\ (\mathit{expr})\ \mathit{id} \\
& | & \mathsf{parser}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*}) \\
& | & \mathsf{parser}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*})\ (\mathit{param}^{*})\ \mathit{decl}^{*}\ \mathit{state}^{*} \\
& | & \mathsf{action}\ \mathit{id}\ (\mathit{param}^{*})\ \mathit{block} \\
& | & \mathsf{table}\ \mathit{id}\ \mathit{key}^{*}\ \mathit{action}^{*}\ \mathit{entry}^{*}\ \mathit{default}^{?}\ \mathit{custom}^{*} \\
& | & \mathsf{control}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*}) \\
& | & \mathsf{control}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*})\ (\mathit{param}^{*})\ \mathit{decl}^{*}\ \mathit{block} \\
& | & \mathit{type}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*})\ \mathit{block} \\
& | & \mathsf{extern}\ \mathit{type}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*}) \\
& | & \mathit{id}\ (\mathit{param}^{*}) \\
& | & \mathsf{abstract}\ \mathit{type}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*}) \\
& | & \mathit{type}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*}) \\
& | & \mathsf{extern} < \mathit{id} > \mathit{tparam}^{*} \\
& | & \mathsf{package}\ \mathit{id} < \mathit{tparam}^{*} > (\mathit{param}^{*})
\end{array}
$$

$$
\frac{
\begin{array}{c}
C_{callee'} = C_{callee}[.\mathsf{gvis} = vis] \qquad C_{callee''} = \mathsf{enter}\ C_{callee'} \\
(param'^{*}, expr^{*}) = \mathrm{align}_{args}(param^{*}, arg^{*}) \\
S\ C_{caller} \vdash expr^{*} : C_{caller'}\ val^{*} \\
C_{callee'''} = \mathrm{copyin}_{loc}(C_{callee''}, param^{*}, val^{*}) \\
S\ \mathsf{cont}\ C_{callee'''} \vdash block : sig\ C_{callee''''} \\
S\ C_{caller'}\ C_{callee''''}\ param^{*}\ expr^{*} \hookrightarrow C_{caller''}
\end{array}
}{
S\ C_{caller}\ C_{callee} \vdash (\mathsf{faction}\ vis\ param^{*}\ block)\ type^{*}\ arg^{*} : sig\ C_{caler''}
}
$$

$$
\frac{
\begin{array}{c}
S\ C \vdash expr_b : C'\ (\mathsf{vstack}\ val^{*}\ num_i\ num_s) \\
S\ C' \vdash expr_i : C''\ val_i \\
i = \mathrm{unpack}(val_i) \qquad val = val^{*}[i]
\end{array}
}{
S\ C \vdash expr_b\ [expr_i] : C''\ val
}
$$

# Future Directions

**(3) Auto-generate various backends from that single source of truth.**

Others (official prose spec, interpreter, ...), to be designed & discussed...!

3. Formal Spec with complete set of features written in SpecTec DSL

1. Official Spec comprehensible document

2. Implementations lightweight interpreter

# Caveats: What SpecTec is, What SpecTec is NOT

SpecTec is a toolchain for **mechanizing** programming **language definitions**, and auto-generating language representations.

Currently SpecTec does **NOT**,

- generate a **parser**
- generate a **type checker**
- generate an **interpreter** written in **industrial** programming languages
  - instead operates on our IR(Intermediate Representation)
- provide a **plug-and-play** experience
  - need adaptations to make it work on P4-specific invariants

# Welcoming Discussions

We hope P4-SpecTec can:

- Support **fast prototyping** of new features.
- **Clarify** unintentionally ambiguous terms in the official spec.

And **discuss** with the community:

- In what shape do we imagine the auto-generated spec?
- What feature can be prototyped in P4-SpecTec?
- What other backends may be useful for P4?


# Thank You


WebAssembly Community Group, **WebAssembly specification**, version 2.0, August 2024.

Dongjun Youn, Wonho Shin, Jaehyun Lee, Sukyoung Ryu, Joachim Breitner, Philippa Gardner, Sam Lindley, Matija Pretnar, Xiaojia Rao, Conrad Watt, and Andreas Rossberg. 2024. **Bringing the WebAssembly Standard up to Speed with SpecTec**. Proc. ACM Program. Lang. 8, PLDI, Article 210 (June 2024), 26 pages. https://doi.org/10.1145/3656440

The P4 Language Consortium. **P416 language specification**, version 1.2.4, May 2023.

Ryan Doenges, Mina Tahmasbi Arashloo, Santiago Bautista, Alexander Chang, Newton Ni, Samwise Parkinson, Rudy Peterson, Alaia Solko-Breslin, Amanda Xu, and Nate Foster. 2021. **Petr4: formal foundations for p4 data planes**. Proc. ACM Program. Lang. 5, POPL, Article 41 (January 2021), 32 pages. https://doi.org/10.1145/3434322

Qinshi Wang, Mengying Pan, Shengyi Wang, Ryan Doenges, Lennart Beringer, and Andrew W. Appel. **Foundational Verification of Stateful P4 Packet Processing**. In 14th International Conference on Interactive Theorem Proving (ITP 2023). Leibniz International Proceedings in Informatics (LIPIcs), Volume 268, pp. 32:1-32:20, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ITP.2023.32

Anoud Alshnakat, Didrik Lundberg, Roberto Guanciale, and Mads Dam. 2024. **HOL4P4: Mechanized Small-Step Semantics for P4**. Proc. ACM Program. Lang. 8, OOPSLA1, Article 102 (April 2024), 27 pages. https://doi.org/10.1145/3649819