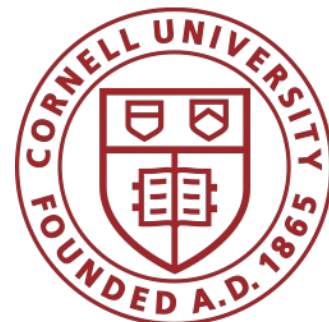


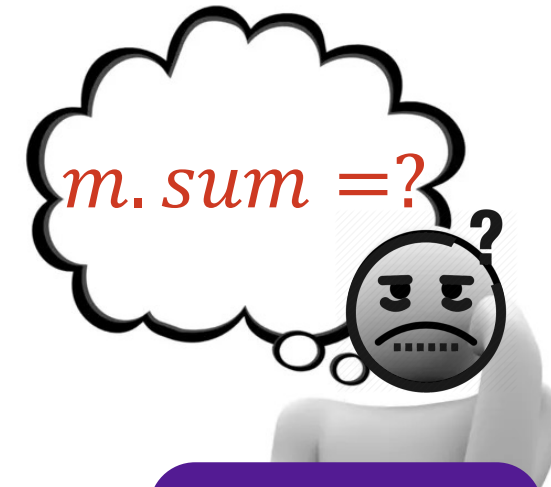


# Formalizing P4's Type System

Parisa Ataei  
Harim Hahn  
Ryan Doenges  
Nate Foster



# Types matter in P4



P4<sub>14</sub>

P4<sub>14</sub> spec

```
header_type ht {
  fields {
    a: 5;
    sum: 5;
  }
}

actions sum() {
  add(m.sum, m.a, m.a);
}

a = 0x10
```

## Description

The dest field's value is updated with the result of adding the two value parameters. Each value parameter may be from a table parameter, an immediate value, a field reference or a register reference; see `modify_field` above. If either value is a field reference and its parent header is not valid, then no change is made to `dest`. A description of the logical behavior follows in the Section 9.1.1 below. If a value is an immediate value, it may be negative.

```
BMv2 > m.sum()
0x20
```

# Migrating from P4<sub>14</sub> to P4<sub>16</sub>

P4 Git

## [design + prototyping] Strong typing #8

✓ Closed

chkim4142 opened this issue on Oct 20, 2015 · 1 comment



chkim4142 commented on Oct 20, 2015

Contributor



Address all the undefined behaviors regarding P4 types.

Problem statement and solution proposal:

[p4-strong-typing-mihai.pdf](#)

P4<sub>16</sub> spec

- Addition, denoted by +. This operation is associative. The result is computed by truncating the result of the addition to the width of the output (similar to C).

# P4<sub>16</sub> has a type system

## P4<sub>16</sub> Language Specification

version 1.2.3

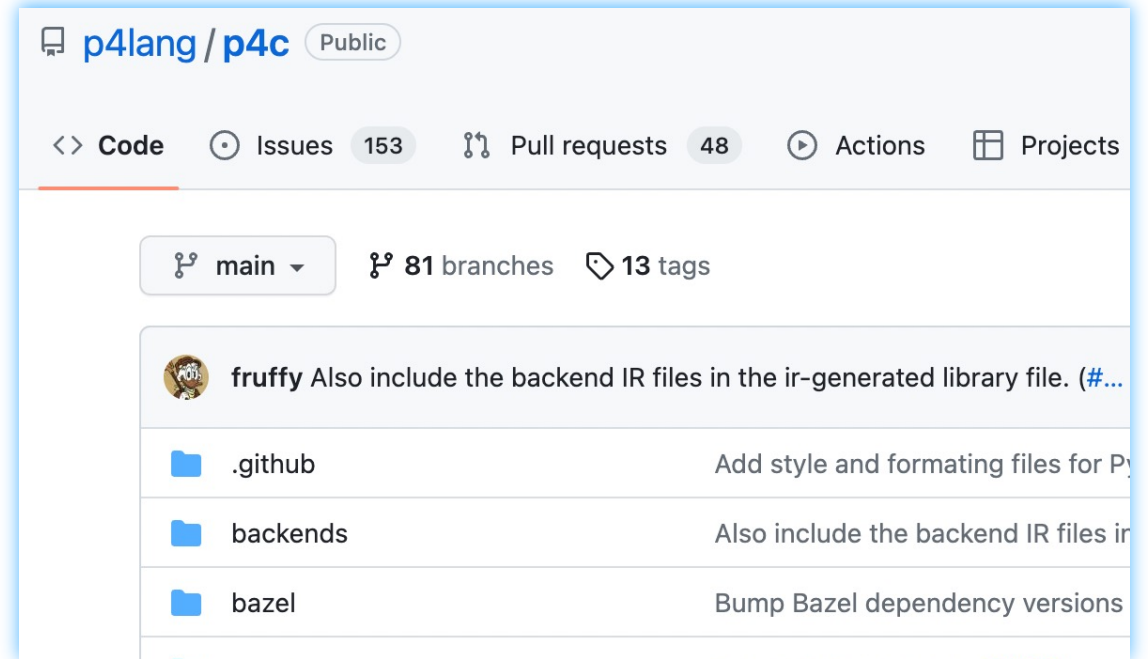
The P4 Language Consortium

2022-07-11




**Abstract.** P4 is a language for programming the data plane of network devices. This document provides a precise definition of the P4<sub>16</sub> language, which is the 2016 revision of the P4 language (<http://p4.org>). The target audience for this document includes developers who want to write compilers, simulators, IDEs, and debuggers for P4 programs. This document may also be of interest to P4 programmers who are interested in understanding the syntax and semantics of the language at a deeper level.

### Contents

1. Scope
2. Terms, definitions, and symbols
3. Overview
  - 3.1. Benefits of P4



The screenshot shows the GitHub repository page for `p4lang/p4c`, which is a public repository. The page includes navigation tabs for Code, Issues (153), Pull requests (48), Actions, and Projects. Below the navigation, it shows the current branch as `main` with 81 branches and 13 tags. A commit by user `fruffy` is highlighted, with the message: "Also include the backend IR files in the ir-generated library file. (#...". Below the commit, a list of files is shown:

 <code>.github</code>	Add style and formatting files for P4
 <code>backends</code>	Also include the backend IR files in
 <code>bazel</code>	Bump Bazel dependency versions

# P4's type system in spec

Consider clarifying what is meant by "same type" in the language spec #875

Open 2 of 9 tasks jafingerhut opened this issue on Jun 22, 2020 · 6 comments

When are two types equal?

Include no-op casts ( $t \rightarrow t$ ) in list of explicit casts #1241

Open hackedy opened this issue on Jun 22, 2020 · 1 comment

- Missing.

When does an implicit cast occur?

Allow easier casts for bit width #1242

Open qobilidop opened this issue on Jun 22, 2020 · 1 comment

- Ambiguous.

Allow using  $\dots$  in function calls #884

Open 2 of 2 tasks

- Spread out.

What is P4's inference algorithm?

Lists are not actually tuple types #1062

Open apinski-cavium opened this issue on Apr 20, 2022 · 3 comments

When does a program type check?

is bool a valid type for select expression #1101

Open apinski-cavium opened this issue on May 17, 2022 · 4 comments

# P4's type system in p4c

p4c/frontends/p4/  
frontend.cpp

```
PassManager passes({
    new P4V1::getV1ModelVersion,
    // Parse annotations
    new ParseAnnotationBodies(&parseAnnotations, &typeMap),
    new PrettyPrint(options),
    // Simple checks on parsed program
    new ValidateParsedProgram(),
    // Synthesize some built-in constructs
    new CreateBuiltins(),
    new ResolveReferences(&refMap, true), // check shadowing
    // First pass of constant folding, before types are known --
    // may be needed to compute types.
    new ConstantFolding(&refMap, nullptr),
    // Desugars direct parser and control applications
    // into instantiations followed by application
    new InstantiateDirectCalls(&refMap),
    new ResolveReferences(&refMap), // check shadowing
    new Deprecated(&refMap),
    new CheckNamedArgs(),
    // Type checking and type inference. Also inserts
    // explicit casts where implicit casts exist.
    new SetStrictStruct(&typeMap, true), // Next pass uses strict struct checking
    new TypeInference(&refMap, &typeMap, false, false), // insert casts, dont' check arrays
    new SetStrictStruct(&typeMap, false),
    new ValidateMatchAnnotations(&typeMap),
    new ValidateValueSets(),
    new DefaultValues(&refMap, &typeMap),
    new BindTypeVariables(&refMap, &typeMap),
    new PassRepeated(
        {new SpecializeGenericTypes(&refMap, &typeMap),
        new DefaultArguments(&refMap, &typeMap), // add default argument values to parameters
        new ResolveReferences(&refMap),
        new SetStrictStruct(&typeMap, true), // Next pass uses strict struct checking
        new TypeInference(&refMap, &typeMap, false), // more casts may be needed
        new SetStrictStruct(&typeMap, false),
        new SpecializeGenericFunctions(&refMap, &typeMap)}}),
    new CheckCoreMethods(&refMap, &typeMap),
    new StaticAssert(&refMap, &typeMap),
```

p4c's type system is distributed among multiple passes interspersed with non-typing passes.

# Diagnosis & consequences

P4's type system is not defined precisely in neither the spec nor the reference implementation.

Open to interpretation.

Error prone.

Hard to extend.



# Extending P4's type system is hard

PSA (and PNA): what types are valid for the hash extern object type params #1154

 Open apinski-cavium opened this issue on Sep 23, 2022 · 4 comments



Type families

Add support for generic arrays and applyonall/forall #1206

 Open apinski-cavium opened this issue on Dec 13, 2022 · 8 comments

Generics

P4\_16: Convenient initialization of header/structs #341

 Open  8 of 9 tasks jafingerhut opened this issue on Jul 2, 2017 · 16 comments


Initializers

Namespaces and imports #718

 Open  5 of 9 tasks jafingerhut opened this issue on Jan 7, 2019 ·

Name space

 Closed

 9 tasks done

Tuples: no elimination form #864

jnfoster opened this issue on May 24, 2020 · 10 comments



mbudiu-vmw closed this as completed on Mar 1, 2021



# Prescription: formalization

## Why?

Gives precise definition  
to the type system.

No holes. No  
ambiguity.

Unifies different  
implementations.

# Prescription: formalization

## How?

Precise mathematical descriptions.

HEADERSTACKINDEX-E

$$\frac{\begin{array}{l} \Gamma \vdash exp_1 : \tau[n] \\ \Gamma \vdash exp_2 : \tau' \\ is\_numeric(exp_2, \tau') \end{array}}{\Gamma \vdash exp_1[exp_2] : \tau}$$

$$\frac{\begin{array}{l} \Gamma \vdash exp_1 : \tau[n] \\ \Gamma \vdash exp_2 : \tau' \\ is\_int(\tau') \\ compile\_time\_known(exp_2)_\Gamma \end{array}}{\Gamma \vdash exp_1[exp_2] : \tau}$$

$$\frac{\begin{array}{l} \Gamma \vdash exp_1 : \tau[n] \\ \Gamma \vdash exp_2 : \tau' \\ is\_fixed\_sign\_int(\tau') \end{array}}{\Gamma \vdash exp_1[exp_2] : \tau}$$

$$\frac{\begin{array}{l} \Gamma \vdash exp_1 : \tau[n] \\ \Gamma \vdash exp_2 : \tau' \\ is\_fixed\_unsign\_int(\tau') \end{array}}{\Gamma \vdash exp_1[exp_2] : \tau}$$

# P4's formalized spec

Judgment	Form
Expression inference	$\Gamma \vdash exp \rightsquigarrow exp', \tau, dir$
Type equality	$\tau_1 =_{\Gamma} \tau_2$
Declaration type checking	$\Gamma \vdash dcl: \tau \dashv \Gamma'$
Type well-formedness	$\Gamma \vdash \tau$
...	...

**Who?**

Language designers.

Practitioners.

Feedback  
appreciated.

# Formalized spec vs. current spec

Add missing definitions.

Remove ambiguity.

Simplify detail-oriented complexities.

# Add missing definitions — example

Consider clarifying what is meant by "same type" in the language spec #875

Open

2 of 9 tasks

jafingerhut opened this issue on Jun 22, 2020 · 6 comments

## Formal spec

$$\frac{\tau_1 ==_{\Gamma} \tau_2}{\text{set}\langle\tau_1\rangle ==_{\Gamma} \text{set}\langle\tau_2\rangle} \quad \text{SETS-T}$$

$$\frac{\tau_1 ==_{\Gamma} \tau_2}{\{\tau_1\} ==_{\Gamma} \{\tau_2\}} \quad \text{LISTS-T}$$

$$\frac{\overline{\tau_1} ==_{\Gamma} \overline{\tau_2}}{\text{tuple}\langle\overline{\tau_1}\rangle ==_{\Gamma} \text{tuple}\langle\overline{\tau_2}\rangle} \quad \text{TUPLES-T}$$

$$\frac{\tau_1 ==_{\Gamma} \tau_2 \quad \overline{f_1} = \overline{f_2} \quad \overline{\tau_1} ==_{\Gamma} \overline{\tau_2}}{\text{enum } \tau_1 \ t_1 \ \{\overline{f_1}\} ==_{\Gamma} \text{enum } \tau_2 \ t_2 \ \{\overline{f_2}\}} \quad \text{SERIALIZABLEENUMS-T}$$

$$\frac{\overline{f_1} = \overline{f_2} \quad \overline{\tau_1} ==_{\Gamma} \overline{\tau_2}}{\text{enum } \tau_1 \ \{t_1\}\overline{f_1} ==_{\Gamma} \text{enum } \tau_2 \ \{t_2\}\overline{f_2}} \quad \text{ENUMS-T}$$

...

# Remove ambiguity — example

Are structure-valued expressions subject to implicit casts? #953



hackedy opened this issue on Aug 2, 2021 · 2 comments · May be fixed by #984

## Formal spec

$$\frac{\text{tuple}\langle\bar{\tau}\rangle ==_{\Gamma, []} \text{tuple}\langle\bar{\tau}'\rangle}{\Delta, \Theta \vdash \text{tuple}\langle\bar{\tau}\rangle \rightarrow \text{struct}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{TUPLETOSTRUCT-AE-2})$$
$$\frac{\Delta, \Theta \vdash \bar{\tau} \rightarrow \bar{\tau}'}{\Delta, \Theta \vdash \text{record}\{f : \tau\} \rightarrow \text{header}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{RECORDTOHEADER-AE-1})$$
$$\frac{\text{record}\{f : \tau\} ==_{\Gamma, []} \text{record}\{f : \tau'\}}{\Delta, \Theta \vdash \text{record}\{f : \tau\} \rightarrow \text{header}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{RECORDTOHEADER-AE-2})$$
$$\frac{\Delta, \Theta \vdash \bar{\tau} \rightarrow \bar{\tau}'}{\Delta, \Theta \vdash \text{record}\{f : \tau\} \rightarrow \text{struct}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{RECORDTOSTRUCT-AE-1})$$
$$\frac{\text{record}\{f : \tau\} ==_{\Gamma, []} \text{record}\{f : \tau'\}}{\Delta, \Theta \vdash \text{record}\{f : \tau\} \rightarrow \text{struct}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{RECORDTOSTRUCT-AE-2})$$

$$\frac{}{\Delta, \Theta \vdash \text{enum } \rho t \{f\} \rightarrow_e \rho} \quad (\text{ENUMTOUNDERLYINGTYPE-AE})$$
$$\frac{}{\Delta, \Theta \vdash \text{enum } \rho t \{f\} \rightarrow_i \rho'} \quad (\text{ENUMTOUNDERLYINGTYPE-AE})$$
$$\frac{\rho ==_{\Gamma, []} \rho'}{\Delta, \Theta \vdash \text{enum } \rho t \{f\} \rightarrow_e \text{enum } \rho' t' \{f'\}} \quad (\text{ENUMS-AE})$$
$$\frac{}{\Delta, \Theta \vdash \rho \rightarrow_e \text{enum } \rho t \{f\}} \quad (\text{TYPETOENUMWITHUNDERLYINGTYPE-AE})$$
$$\frac{\Delta, \Theta \vdash \bar{\tau} \rightarrow \bar{\tau}'}{\Delta, \Theta \vdash \text{tuple}\langle\bar{\tau}\rangle \rightarrow \text{header}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{TUPLETOHEADER-AE-1})$$
$$\frac{\text{tuple}\langle\bar{\tau}\rangle ==_{\Gamma, []} \text{tuple}\langle\bar{\tau}'\rangle}{\Delta, \Theta \vdash \text{tuple}\langle\bar{\tau}\rangle \rightarrow \text{header}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{TUPLETOHEADER-AE-2})$$
$$\frac{\Delta, \Theta \vdash \bar{\tau} \rightarrow \bar{\tau}'}{\Delta, \Theta \vdash \text{tuple}\langle\bar{\tau}\rangle \rightarrow \text{struct}\langle\bar{X}\rangle\{f : \tau'\}} \quad (\text{TUPLETOSTRUCT-AE-1})$$



# Simplify details — example

## 8. Expressions

- 8.1. Expression evaluation order
- 8.2. Operations on **error** types
- 8.3. Operations on **enum** types
- 8.4. Expressions on Booleans
  - 8.4.1. Conditional operator
- 8.5. Operations on fixed-width bit types (unsigned)
- 8.6. Operations on fixed-width signed integers
- 8.7. Operations on arbitrary-precision integers
- 8.8. Concatenation and shifts
  - 8.8.1. Concatenation
  - 8.8.2. A note about shifts
- 8.9. Operations on variable-size bit types
- 8.10. Casts
  - 8.10.1. Explicit casts
  - 8.10.2. Implicit casts
  - 8.10.3. Illegal arithmetic expressions
- 8.11. Operations on tuples expressions
- 8.12. Operations on lists
- 8.13. Operations on structure-valued expressions
- 8.14. Operations on sets
  - 8.14.1. Singleton sets
  - 8.14.2. The universal set
  - 8.14.3. Masks
  - 8.14.4. Ranges
  - 8.14.5. Products
- 8.15. Operations on struct types
- 8.16. Operations on headers
- 8.17. Operations on header stacks
- 8.18. Operations on header unions

## Formal spec

LOGICALOPS-AE( $\odot = \&\&, ||$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \text{bool}, d$ 

NUMERICOPS-E( $\odot = +, -, *$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $is\_numeric(exp_1, \tau_1) \quad is\_numeric(exp_2, \tau_2)$   
 $\tau_1 ==_{\Gamma, \{ \}} \tau_2$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \tau_1, d$ 

EQUALITYCHECKS-E( $\odot = ==, !=$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $if\_int\_is\_compile\_time\_known(\tau_1, exp_1) \quad if\_int\_is\_compile\_time\_known(\tau_2, exp_2)$   
 $\tau_1 ==_{\Gamma, \{ \}} \tau_2 \quad has\_equality(\tau_1) \Gamma$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \text{bool}, d$ 

OPSAT-E( $\odot = |+, |-|$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $width\_int(\tau_1) = - \quad \tau_1 ==_{\Gamma, \{ \}} \tau_2$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \tau_1, d$ 

BITWISEOPS-E( $\odot = \&, |, \wedge, \sim$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $width\_int(\tau_1) = - \quad \tau_1 ==_{\Gamma, \{ \}} \tau_2$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \tau_1, d$ 

BITSTRINGCONCATENATION-E  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $concat\_type(\tau_1, \tau_2) = \tau$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \# exp_2 \rightsquigarrow exp'_1 \# exp'_2, \tau, d$ 

COMPARISONOPS-E( $\odot = <, <=, >, >=$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $is\_numeric(exp_1, \tau_1) \quad is\_numeric(exp_2, \tau_2)$   
 $\tau_1 ==_{\Gamma, \{ \}} \tau_2$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \text{bool}, d$ 

DIVOPS-E( $\odot = /, \%$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \text{int}, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \text{int}, d_2$   
 $is\_nonneg\_numeric(exp_1) \Gamma \quad is\_nonneg\_numeric(exp_2) \Gamma$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \text{int}, d$ 

SHIFTOPS-E( $\odot = \ll, \gg$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $is\_numeric(exp_1, \tau_1) \quad shift\_condition(\tau_2, exp_2)_{\Sigma, \Delta}$   
 $in\_or\_directionless(d_1, d_2) = d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp'_1 \odot exp'_2, \tau_1, d$ 

BINOPS-INSERTIMPLICITCAST-E( $\odot = +, -, *, ==, !=, \#, <, <=, >, >=$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $insert\_implicit\_cast(exp_1, \tau_1) = exp''_1 \quad insert\_implicit\_cast(exp_2, \tau_2) = exp''_2$   
 $\Sigma, \Gamma, \Delta, c \vdash exp''_1 \odot exp''_2 \rightsquigarrow exp, \tau, d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp, \tau, d$ 

BINOPS-INSERT-ENUM-IMPLICITCAST-E( $\odot = \#, \ll, \gg$ )  
 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \rightsquigarrow exp'_1, \tau_1, d_1 \quad \Sigma, \Gamma, \Delta, c \vdash exp_2 \rightsquigarrow exp'_2, \tau_2, d_2$   
 $insert\_enum\_implicit\_cast(exp_1, \tau_1) = exp''_1$   
 $insert\_enum\_implicit\_cast(exp_2, \tau_2) = exp''_2$   
 $\Sigma, \Gamma, \Delta, c \vdash exp''_1 \odot exp''_2 \rightsquigarrow exp, \tau, d$

---

 $\Sigma, \Gamma, \Delta, c \vdash exp_1 \odot exp_2 \rightsquigarrow exp, \tau, d$

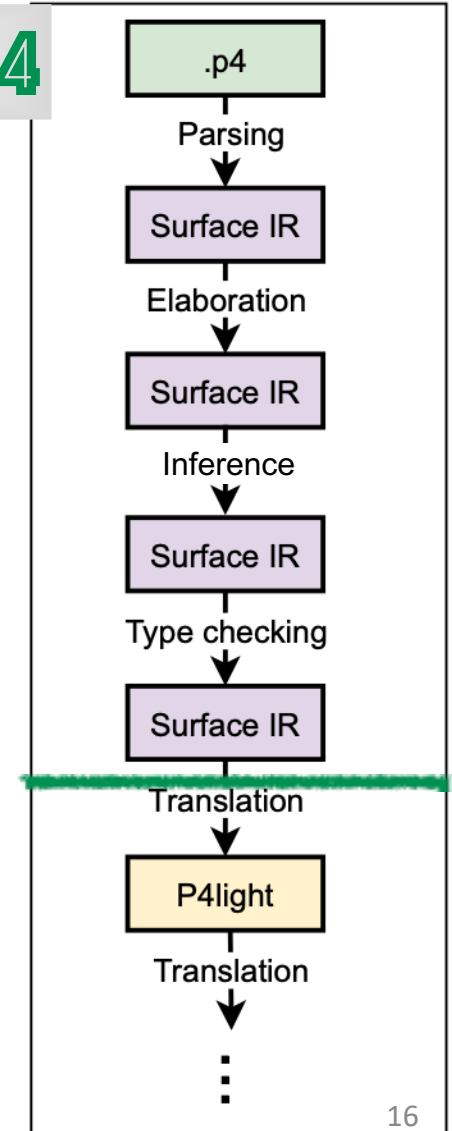
# Formalization to implementation

A simple surface IR.

Type system is carried out on surface IR.

Type system is divided into smaller passes for separation of concern.

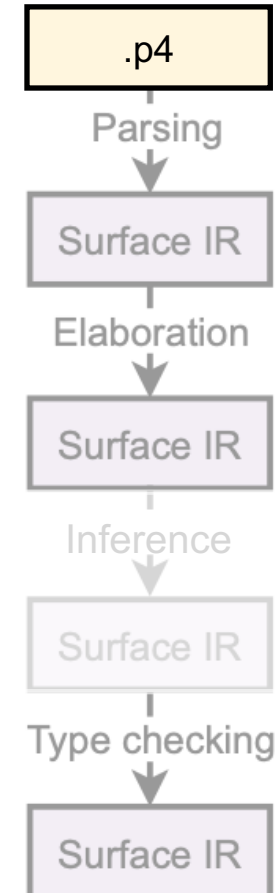
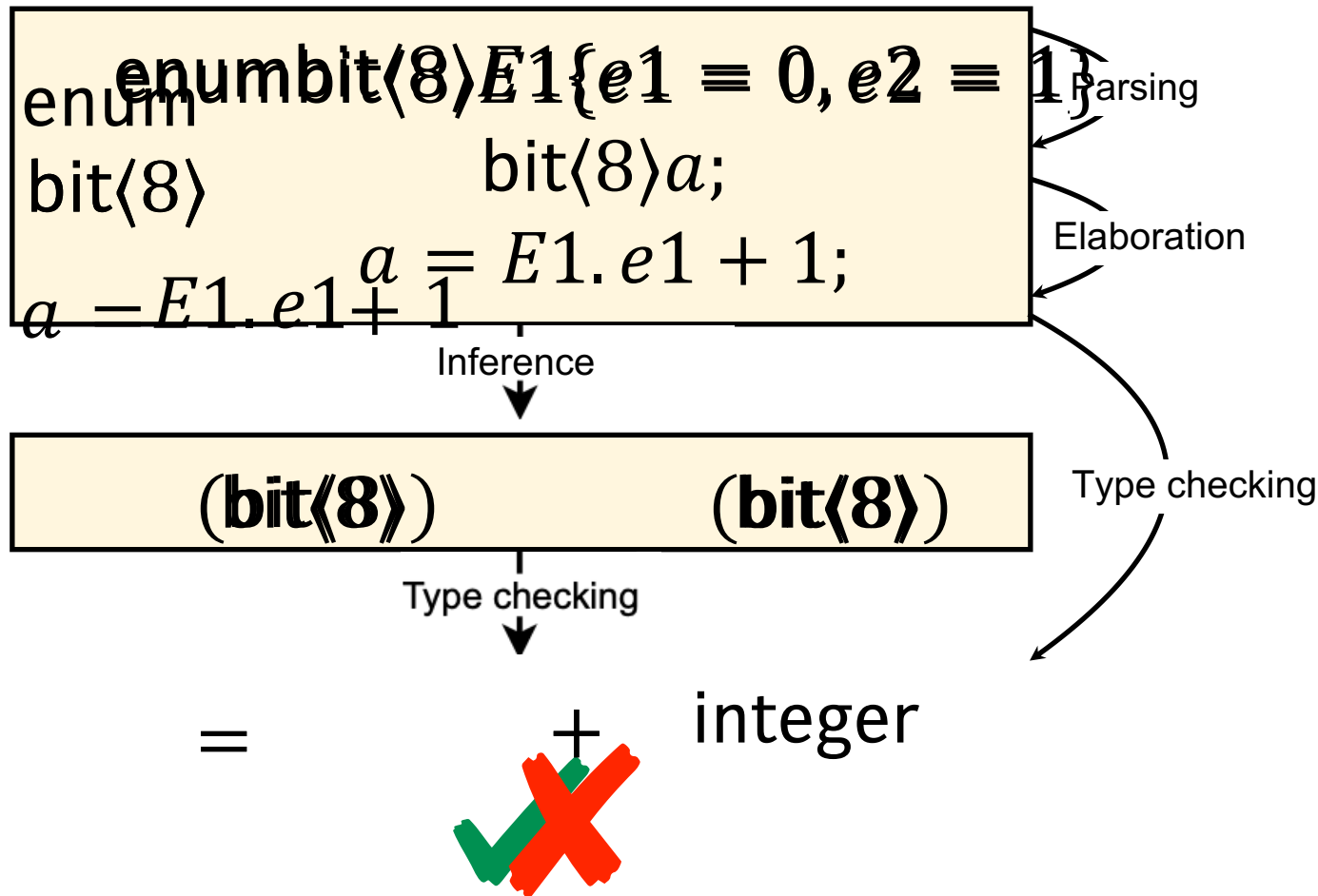
POULET4







# Inference & checking — example



# Artifacts and progress

Surface IR



Frontend  
passes



**In progress.**

Formalized  
spec



**In progress.**



# Conclusion

Types matter in P4.

Having precise definition (formalization) of type systems matters.

Benefits:

- unifies different implementations
- develop frontend-specific tools
- easier to understand and extend.



# Thank You!

Parisa Ataei

Harim Hahn

Ryan Doenges

Nate Foster

**On the job market!  
Let's chat!**

Active development of formalized spec: <https://github.com/verified-network-toolchain/petr4/tree/p4>

Active development of Poulet4: <https://github.com/verified-network-toolchain/petr4/tree/main>