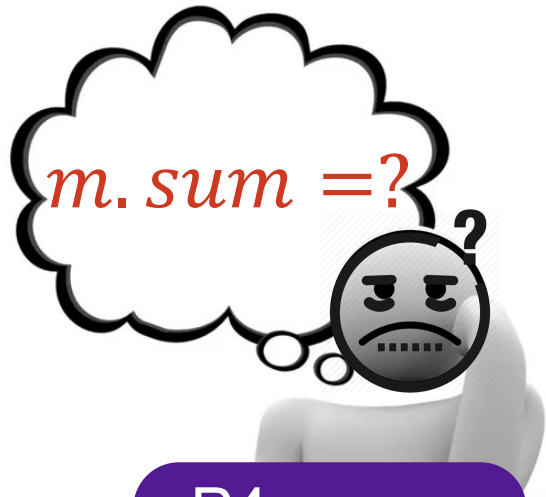
Formalizing P4's Type System

Parisa Ataei
Harim Hahn
Ryan Doenges
Nate Foster

Types matter in P4



P4$_{14}$

header_typeht{
fields{
a: 5;
sum: 5;
}
}
actionsum(){
add($m.sum, m.a, m.a$);
}
$a = 0x10$

P4$_{14}$ spec

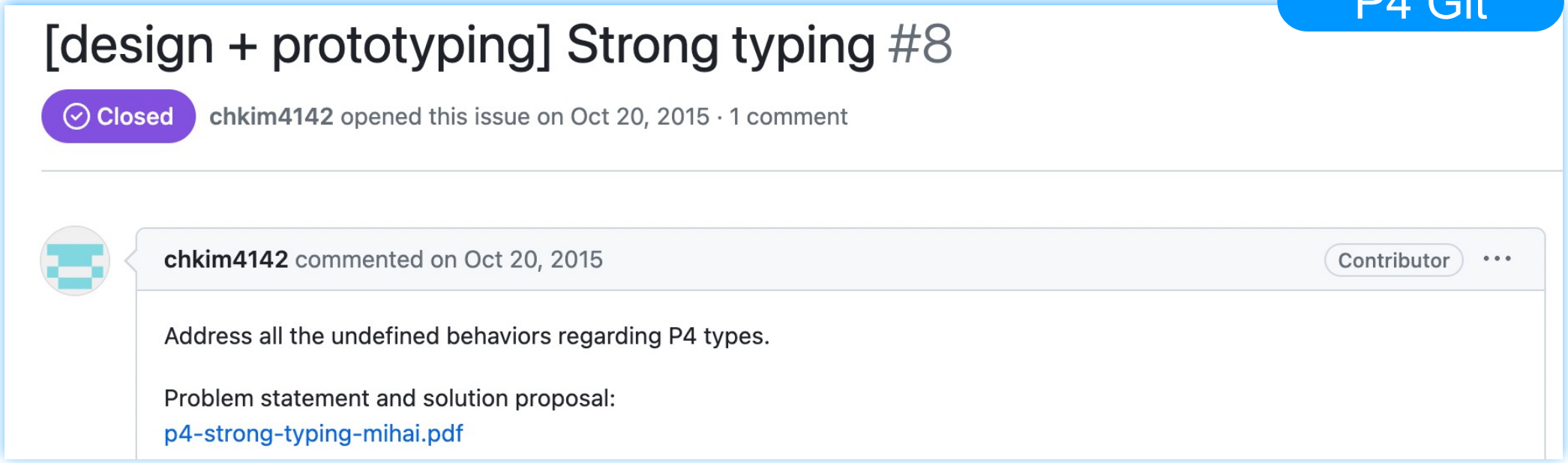
Description

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 a 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 $P4_{14}$ to $P4_{16}$

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

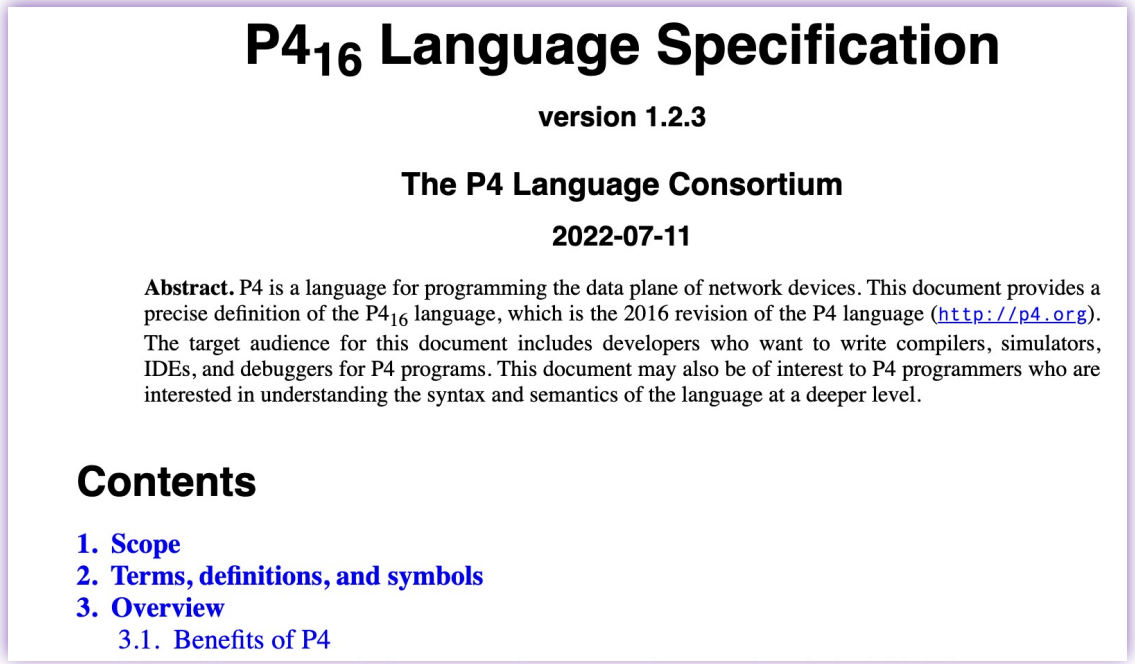
Address all the undefined behaviors regarding P4 types.

Problem statement and solution proposal:
p4-strong-typing-mihai.pdf

$P4_{16}$ 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).

$P4_{16}$ has a type system



$P4_{16}$ Language Specification

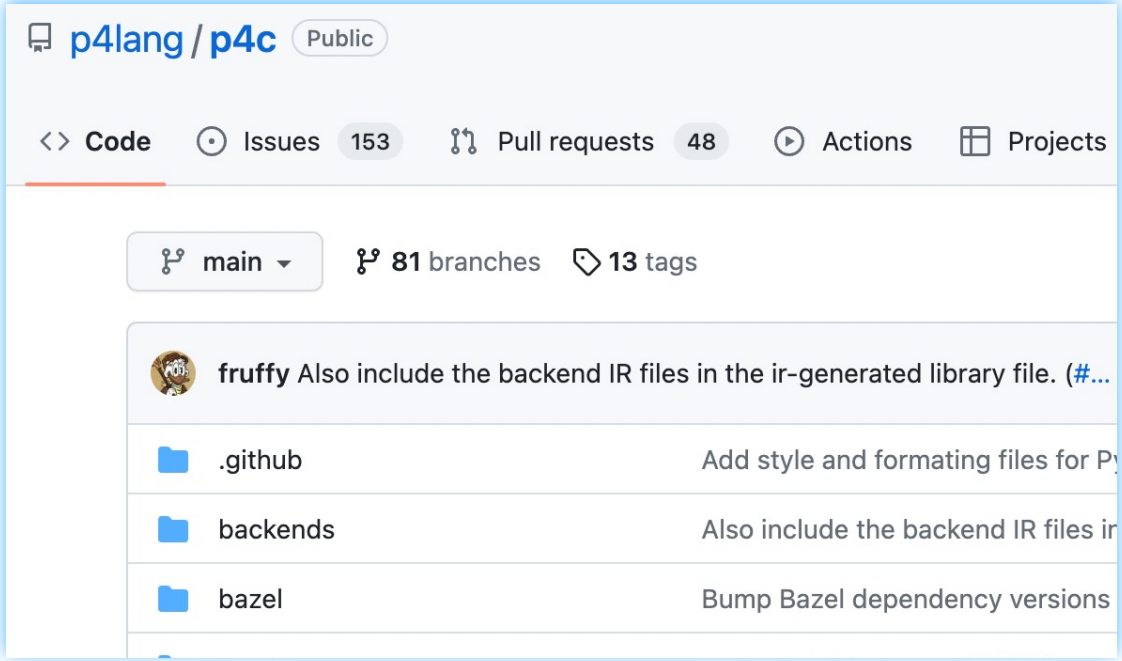
version 1.2.3

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 $P4_{16}$ 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



p4lang / p4c Public

Code Issues 153 Pull requests 48 Actions Projects

main 81 branches 13 tags

fruffy Also include the backend IR files in the ir-generated library file. (#...

.github	Add style and formating files for P
backends	Also include the backend IR files ir
bazel	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 -> t) in list of explicit casts #1241

Open · hackedy opened th

Allow easier cas … bit width #1242

Open · qobilidop opened th

When does an implicit cast occur?

- Missing.
- Ambiguous.
- Spread out.

Allow using … lls #884

Open · 2 of

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

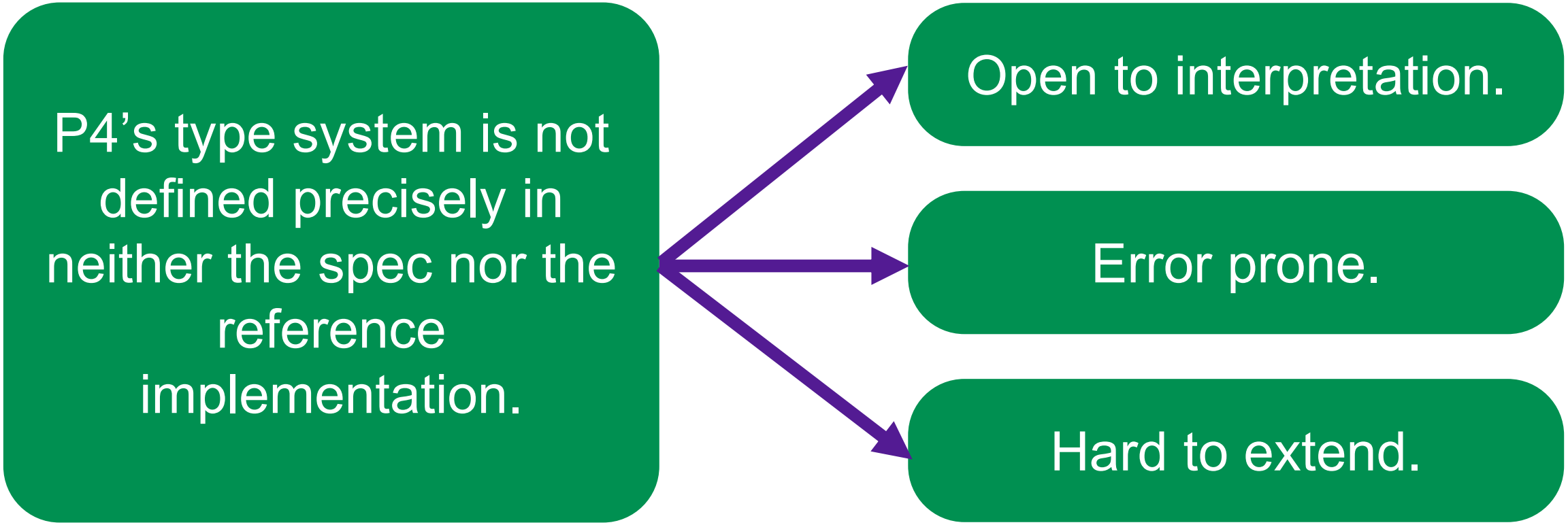
P4’s type system in p4c

p4c/frontends/p4/frontend.cpp

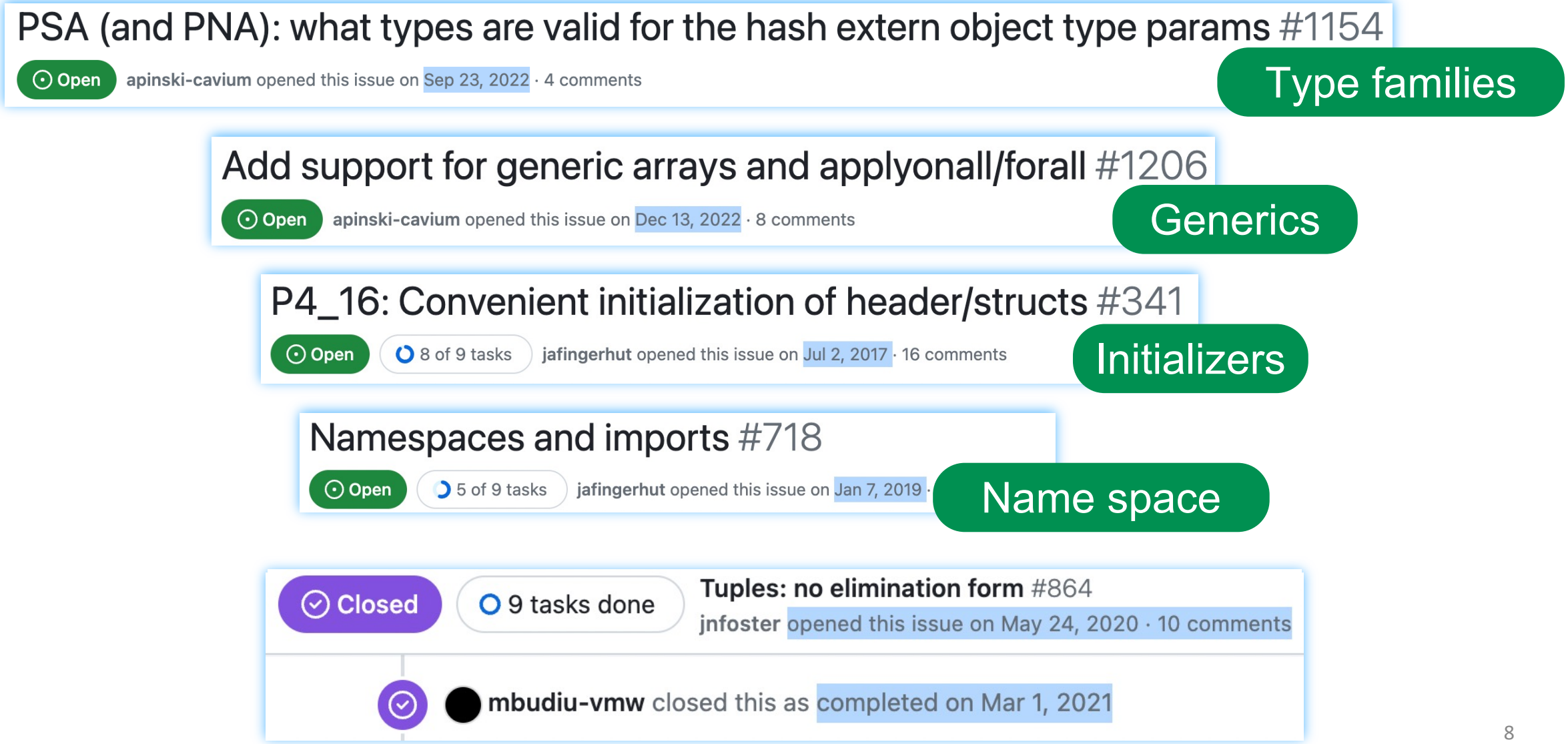
```
PassManager passes({
    new P4V1::getV1ModelVersion,
    // Parse annotations
    new ParseAnnotationBodies(&parseAnnotations, &typeMap),
    new PrettyPrint(options),
    // Simple checks on parsed program
    new ValidateParsedProgram(),
    // Synthesize some built-in constructs
    new CreateBuiltins(),
    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



Extending P4's type system is hard



Prescription: formalization

Why?

Gives precise definition to the type system.

No holes. No ambiguity.

Unifies different implementations.

Prescription: formalization

How?

Precise mathematical descriptions.

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

HeaderStackIndex-E

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

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

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

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

$$\frac{\tau_1 ==_\Gamma \tau_2}{\mathsf{set}\langle\tau_1\rangle ==_\Gamma \mathsf{set}\langle\tau_2\rangle}\ \textsc{Sets-T} \qquad \frac{\tau_1 ==_\Gamma \tau_2}{\{\tau_1\} ==_\Gamma \{\tau_2\}}\ \textsc{Lists-T}$$

$$\frac{\overline{\tau_1} ==_\Gamma \overline{\tau_2}}{\mathsf{tuple}\langle\overline{\tau_1}\rangle ==_\Gamma \mathsf{tuple}\langle\overline{\tau_2}\rangle}\ \textsc{Tuples-T}$$

$$\frac{\tau_1 ==_\Gamma \tau_2 \qquad \overline{f_1} = \overline{f_2} \qquad \overline{\tau_1} ==_\Gamma \overline{\tau_2}}{\mathsf{enum}\ \tau_1\ t_1\ \{\overline{f_1}\} ==_\Gamma \mathsf{enum}\ \tau_2\ t_2\ \{\overline{f_2}\}}\ \textsc{SerializableEnums-T}$$

$$\frac{\overline{f_1} = \overline{f_2} \qquad \overline{\tau_1} ==_\Gamma \overline{\tau_2}}{\mathsf{enum}\ \tau_1\ \{t_1\}\overline{f}_1 ==_\Gamma \mathsf{enum}\ \tau_2\ \{t_2\}\overline{f}_2}\ \textsc{Enums-T}$$

Remove ambiguity —example

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

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

Formal spec

$$\frac{\mathsf{tuple}\langle\overline{\tau}\rangle ==_{\Gamma,[\,]} \mathsf{tuple}\langle\overline{\tau'}\rangle}{\Delta,\Theta \vdash \mathsf{tuple}\langle\overline{\tau}\rangle \rightarrow \ \mathsf{struct}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{TupleToStruct-AE-2})$$

$$\frac{\Delta,\Theta \vdash \overline{\tau} \rightarrow \ \overline{\tau'}}{\Delta,\Theta \vdash \mathsf{record}\{\overline{f:\tau}\} \rightarrow \ \mathsf{header}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{RecordToHeader-AE-1})$$

$$\frac{\mathsf{record}\{\overline{f:\tau}\} ==_{\Gamma,[\,]} \mathsf{record}\{\overline{f:\tau'}\}}{\Delta,\Theta \vdash \mathsf{record}\{\overline{f:\tau}\} \rightarrow \ \mathsf{header}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{RecordToHeader-AE-2})$$

$$\frac{\Delta,\Theta \vdash \overline{\tau} \rightarrow \ \overline{\tau'}}{\Delta,\Theta \vdash \mathsf{record}\{\overline{f:\tau}\} \rightarrow \ \mathsf{struct}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{RecordToStruct-AE-1})$$

$$\frac{\mathsf{record}\{\overline{f:\tau}\} ==_{\Gamma,[\,]} \mathsf{record}\{\overline{f:\tau'}\}}{\Delta,\Theta \vdash \mathsf{record}\{\overline{f:\tau}\} \rightarrow \ \mathsf{struct}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{RecordToStruct-AE-2})$$

$$\frac{}{\Delta,\Theta \vdash \mathsf{enum}\ \rho\ t\ \{\overline{f}\} \rightarrow_e \rho} \quad (\textsc{EnumToUnderlyingType-AE})$$

$$\frac{}{\Delta,\Theta \vdash \mathsf{enum}\ \rho\ t\ \{\overline{f}\} \rightarrow_i \rho'} \quad (\textsc{EnumToUnderlyingType-AE})$$

$$\frac{\rho ==_{\Gamma,[\,]} \rho'}{\Delta,\Theta \vdash \mathsf{enum}\ \rho\ t\ \{\overline{f}\} \rightarrow_e \mathsf{enum}\ \rho'\ t'\ \{\overline{f'}\}} \quad (\textsc{Enums-AE})$$

$$\frac{}{\Delta,\Theta \vdash \rho \rightarrow_e \mathsf{enum}\ \rho\ t\ \{\overline{f}\}} \quad (\textsc{TypeToEnumWithUndrlyingType-AE})$$

$$\frac{\Delta,\Theta \vdash \overline{\tau} \rightarrow \ \overline{\tau'}}{\Delta,\Theta \vdash \mathsf{tuple}\langle\overline{\tau}\rangle \rightarrow \ \mathsf{header}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{TupleToHeader-AE-1})$$

$$\frac{\mathsf{tuple}\langle\overline{\tau}\rangle ==_{\Gamma,[\,]} \mathsf{tuple}\langle\overline{\tau'}\rangle}{\Delta,\Theta \vdash \mathsf{tuple}\langle\overline{\tau}\rangle \rightarrow \ \mathsf{header}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{TupleToHeader-AE-2})$$

$$\frac{\Delta,\Theta \vdash \overline{\tau} \rightarrow \ \overline{\tau'}}{\Delta,\Theta \vdash \mathsf{tuple}\langle\overline{\tau}\rangle \rightarrow \ \mathsf{struct}\langle\overline{X}\rangle\{\overline{f:\tau'}\}} \quad (\textsc{TupleToStruct-AE-1})$$

. . .

Simplify details —example

Formal spec

LogicalOps-AE($\odot$ = &&, ||)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \mathsf{bool}, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \mathsf{bool}, d_2 \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \mathsf{bool}, d}$$

NumericOps-E($\odot$ = +, -, *)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad is\_numeric(exp_1,\tau_1) \qquad is\_numeric(exp_2,\tau_2) \qquad \tau_1 ==_{\Gamma,[\,]} \tau_2 \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \tau_1, d}$$

EqualityChecks-E($\odot$ = ==, !=)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad if\_int\_is\_compile\_time\_known(\tau_1, exp_1) \qquad if\_int\_is\_compile\_time\_known(\tau_2, exp_2) \qquad \tau_1 ==_{\Gamma,[\,]} \tau_2 \qquad has\_equality(\tau_1)_\Gamma \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \mathsf{bool}, d}$$

OpSat-E($\odot$ = |+|, |-|)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad width\_int(\tau_1) = \_ \qquad \tau_1 ==_{\Gamma,[\,]} \tau_2 \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \tau_1, d}$$

BitwiseOps-E($\odot$ = &, |, ^, ~)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad width\_int(\tau_1) = \_ \qquad \tau_1 ==_{\Gamma,[\,]} \tau_2 \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \tau_1, d}$$

BitstringConcatenation-E

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad concat\_type(\tau_1,\tau_2) = \tau \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 {+\!\!+} exp_2 \rightsquigarrow exp_1' {+\!\!+} exp_2', \tau, d}$$

ComparisonOps-E($\odot$ = <, <=, >, >=)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad is\_numeric(exp_1,\tau_1) \qquad is\_numeric(exp_2,\tau_2) \qquad \tau_1 ==_{\Gamma,[\,]} \tau_2 \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \mathsf{bool}, d}$$

DivOps-E($\odot$ = /, %)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \mathsf{int}, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \mathsf{int}, d_2 \qquad is\_nonneg\_numeric(exp_1)_\Gamma \qquad is\_nonneg\_numeric(exp_2)_\Gamma \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \mathsf{int}, d}$$

ShiftOps-E($\odot$ = <<, >>)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad is\_numeric(exp_1,\tau_1) \qquad shift\_condition(\tau_2, exp_2)_{\Sigma,\Delta} \qquad in\_or\_directionless(d_1,d_2) = d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp_1' \odot exp_2', \tau_1, d}$$

BinOps-InsertImplicitCast-E($\odot$ = +, -, *, ==, !=, ++, <, <=, >, >=)

$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad insert\_implicit\_cast(exp_1,\tau_1) = exp_1'' \qquad insert\_implicit\_cast(exp_2,\tau_2) = exp_2'' \qquad \Sigma,\Gamma,\Delta,c \vdash exp_1'' \odot exp_2'' \rightsquigarrow exp, \tau, d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp, \tau, d}$$

BinOps-Insert-Enum-ImplicitCast-E($\odot$ = ++, <<, >>)

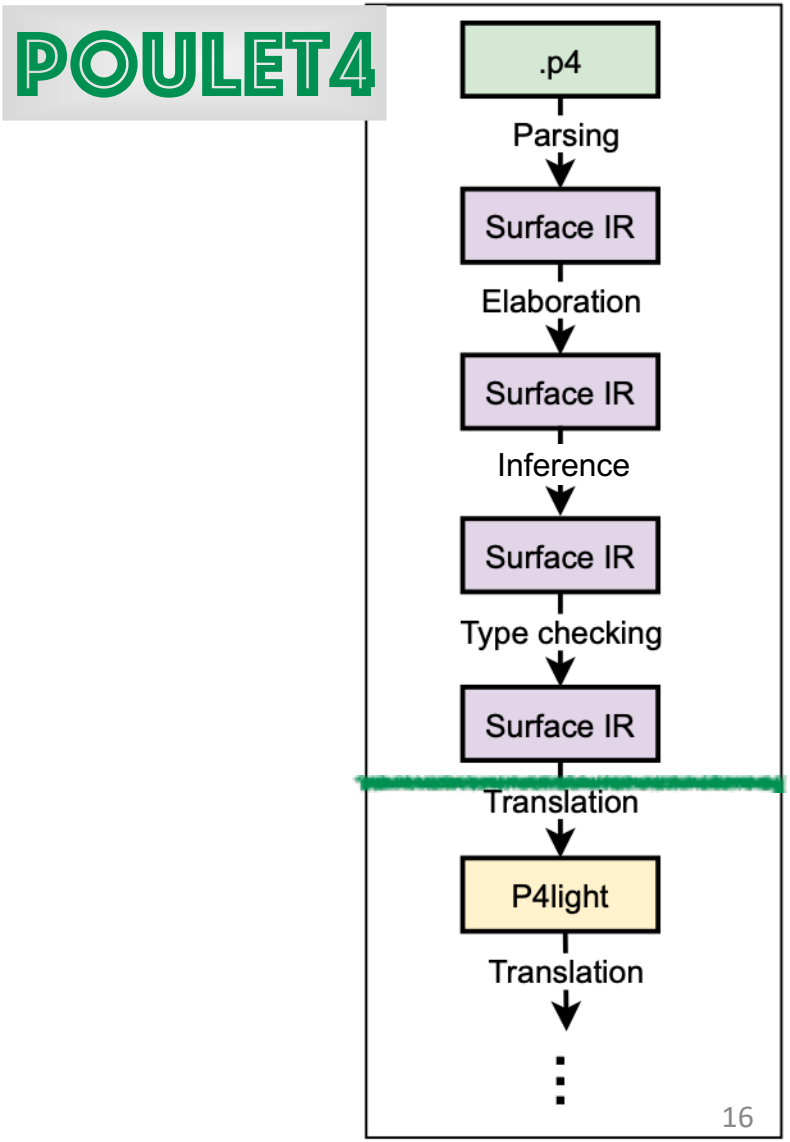
$$\frac{\Sigma,\Gamma,\Delta,c \vdash exp_1 \rightsquigarrow exp_1', \tau_1, d_1 \qquad \Sigma,\Gamma,\Delta,c \vdash exp_2 \rightsquigarrow exp_2', \tau_2, d_2 \qquad insert\_enum\_implicit\_cast(exp_1,\tau_1) = exp_1'' \qquad insert\_enum\_implicit\_cast(exp_2,\tau_2) = exp_2'' \qquad \Sigma,\Gamma,\Delta,c \vdash exp_1'' \odot exp_2'' \rightsquigarrow exp, \tau, d}{\Sigma,\Gamma,\Delta,c \vdash exp_1 \odot exp_2 \rightsquigarrow exp, \tau, d}$$

Formalization to implementation

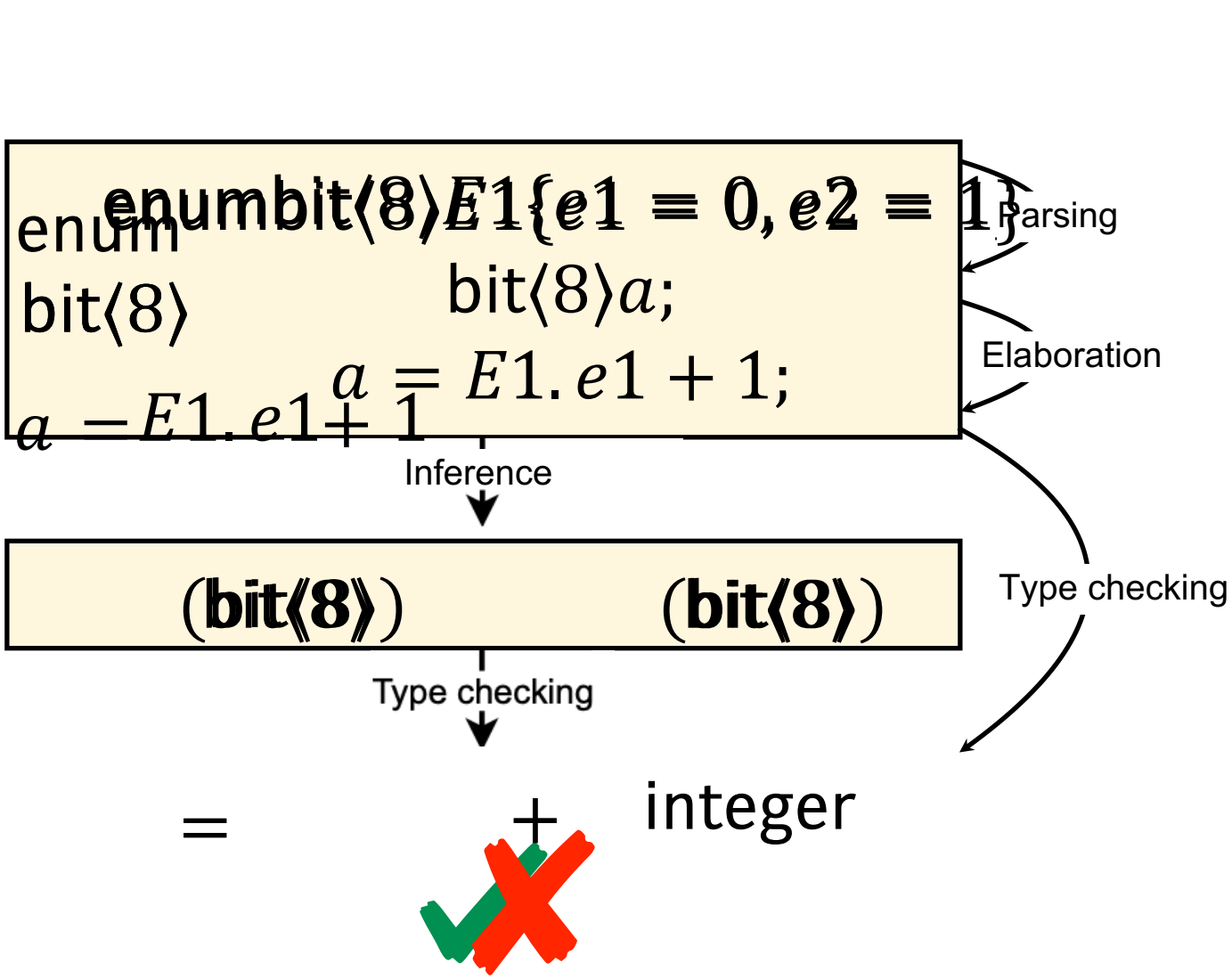
A simple surface IR.

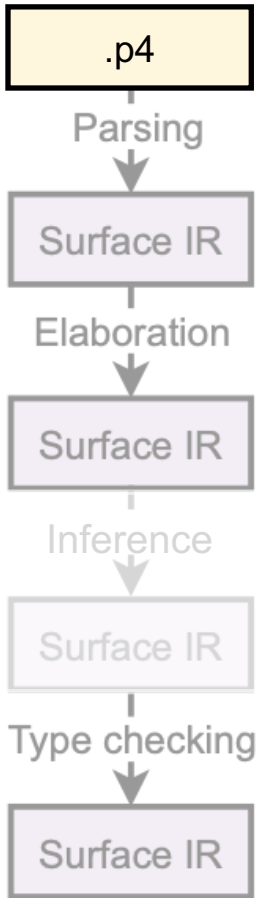
Type system is carried out on surface IR.

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



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