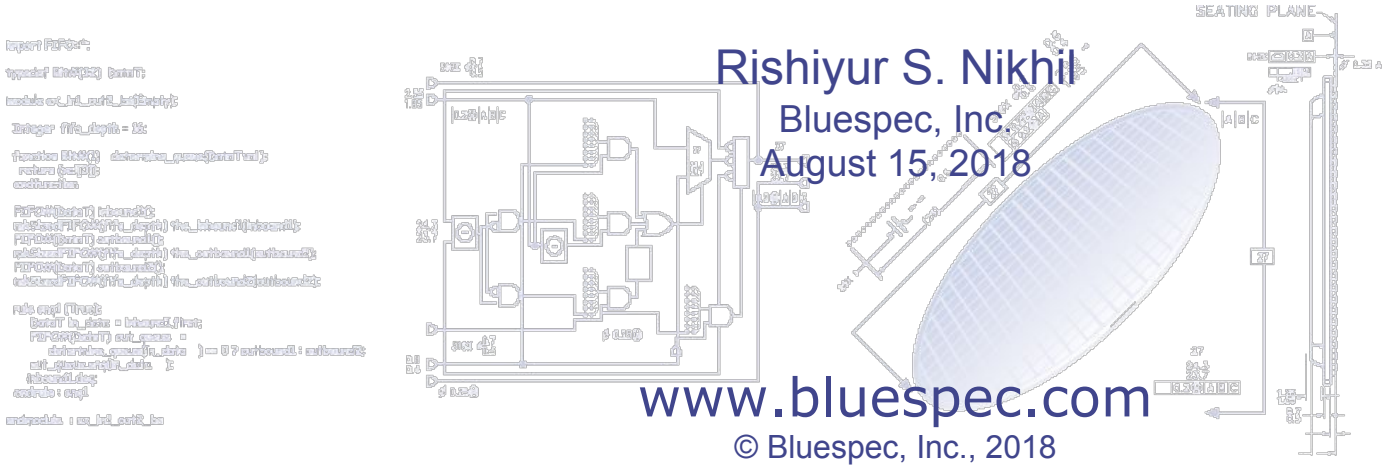
An Experimental Formal Feature Model

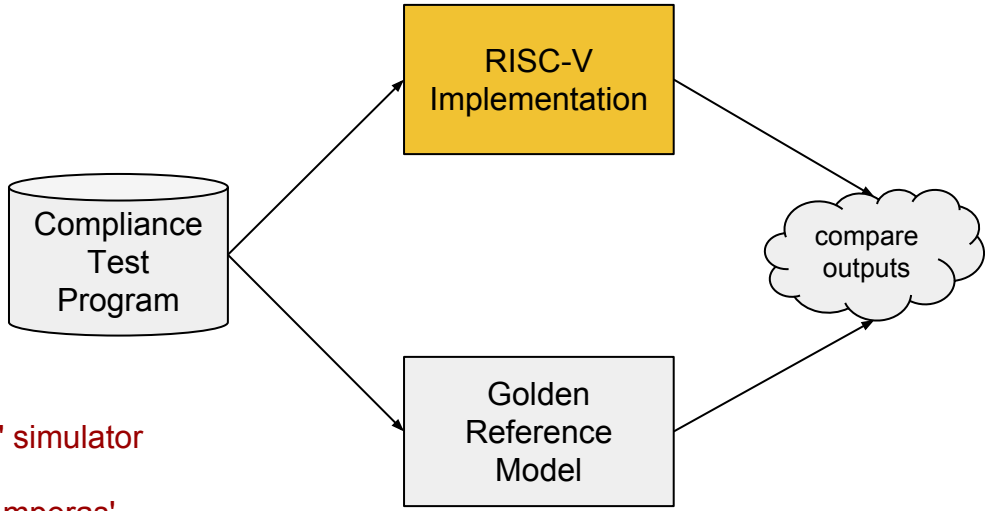
https://github.com/rsnikhil/Experimental_RISCV_Feature_Model



Rishiyur S. Nikhil
Bluespec, Inc.
August 15, 2018

www.bluespec.com

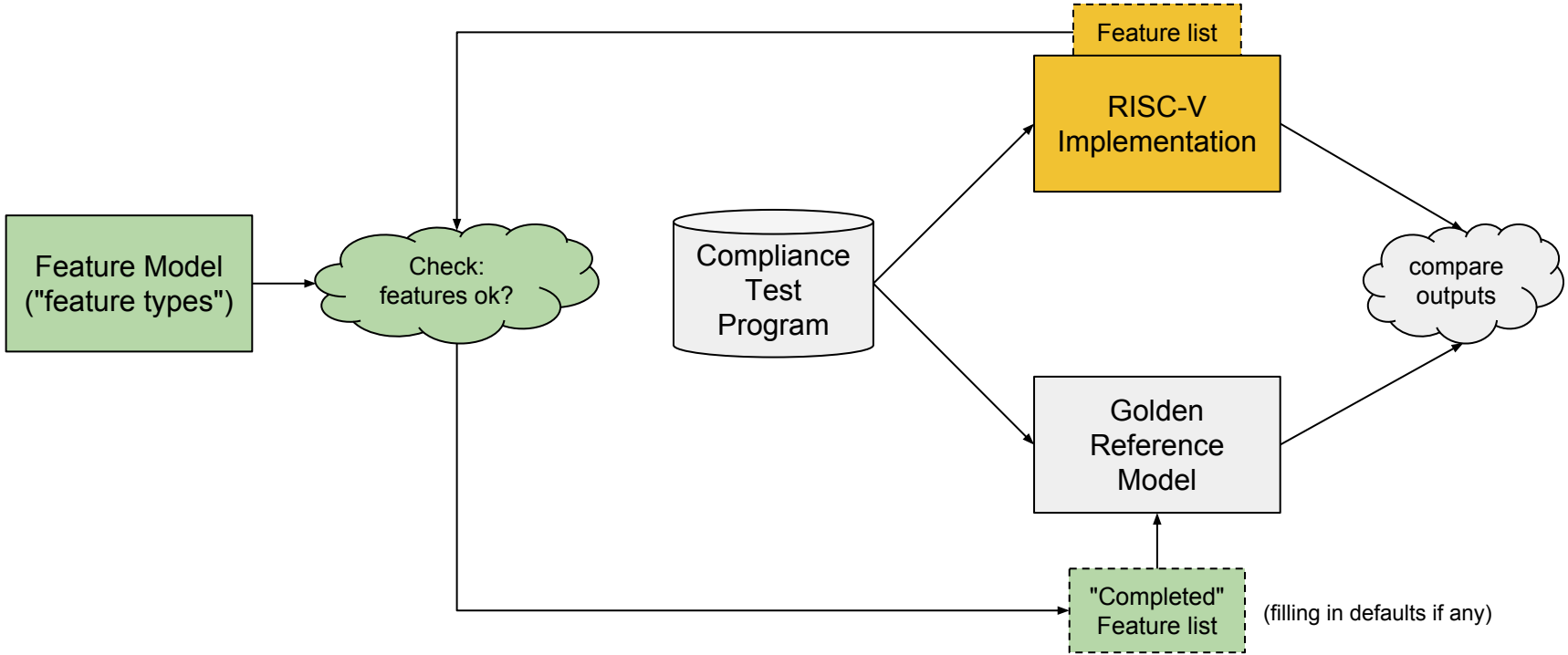
Rôle of Feature Model in Compliance-checking



- The Golden Reference Model is a "universal" simulator (can model any legal RISC-V behavior)
  - Examples: Spike, Formal ISA Specs, Imperas' simulator, Bluespec's Cissr, ...
- Needs to be constrained to just those behaviors that the implementation can exhibit
  - (implemented features)

bluespec

Rôle of Feature Model in Compliance-checking



bluespec

This Experimental Feature Model

- Experimental: Quick-and-dirty Python and YAML prototype just to explore ideas:
  - Expressive power? (e.g., how rich a constraint language?)
  - Tool flow?
- But imagine a standalone DSL (Domain Specific Language) and tool

Please play with it!

- https://github.com/rsnikhil/Experimental_RISCV_Feature_Model
- Suggest implementation choices that are missing
- Suggest corrections/improvements to the constraints
- Any other improvements/changes?

Features and Constraints

- Features: RV32, M, S, Sv48, traps_on_unaligned, …, address_map, ...
- Constraints: legality conditions for features (types, allowed values, dependence on other features, …)

- A constraint is a boolean expression that must evaluate True
  - For engineering/user-friendliness reasons, we lift out "pre-conditions" as separate expressions
    - E.g., RV64 and S are pre-conditions for Sv39
  - Technically, a precond P and constraint C are equivalent to a composite constraint (P && C) || (!P)
    i.e., a constraint is trivially True if its precondition is False.

- Feature values, defaults and constraints are expressions built from *simple first-order terms*:

```
E ::= scalar_const
    | list_const
    | var
    | Op (E, E, …, E)
```

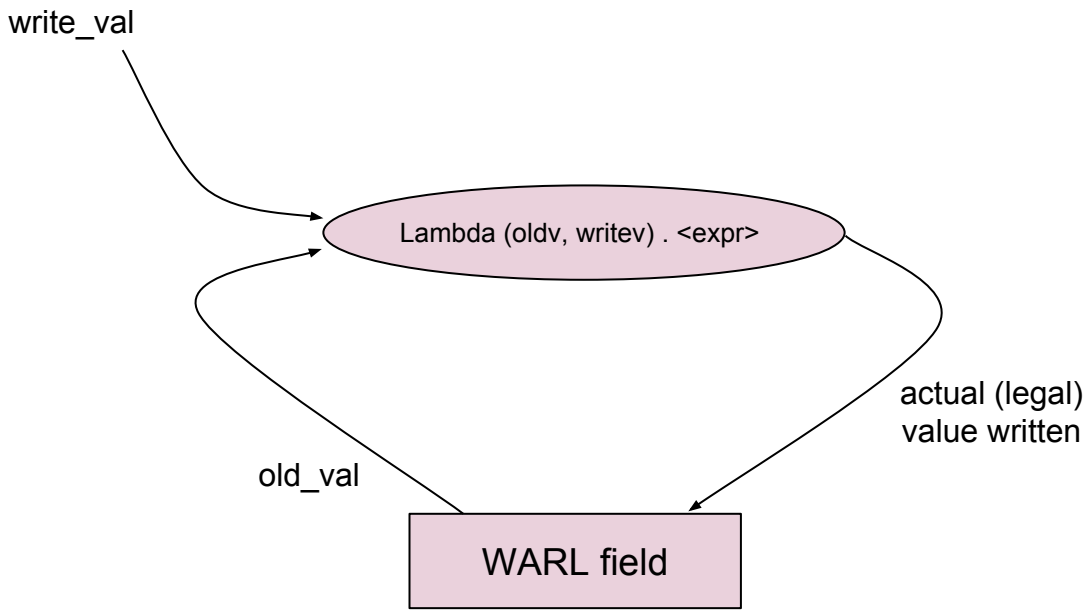
- Lots of standard 'Op's: type-test, arithmetic, logic, …
  Some specialized Ops:

In (x, ys)	Test if x is a member of ys
Range (n1,n2)	List n1,n1+1,...,n2-1
Fval (f)	Value of feature f in implementation feature list
Are_hartids (xs)	xs is a set, including '0'
XLEN_code	32 → 1, 64 → 2

WARL ("Write Any, Read Legal") features

- A WARL feature's value is a WARL-function

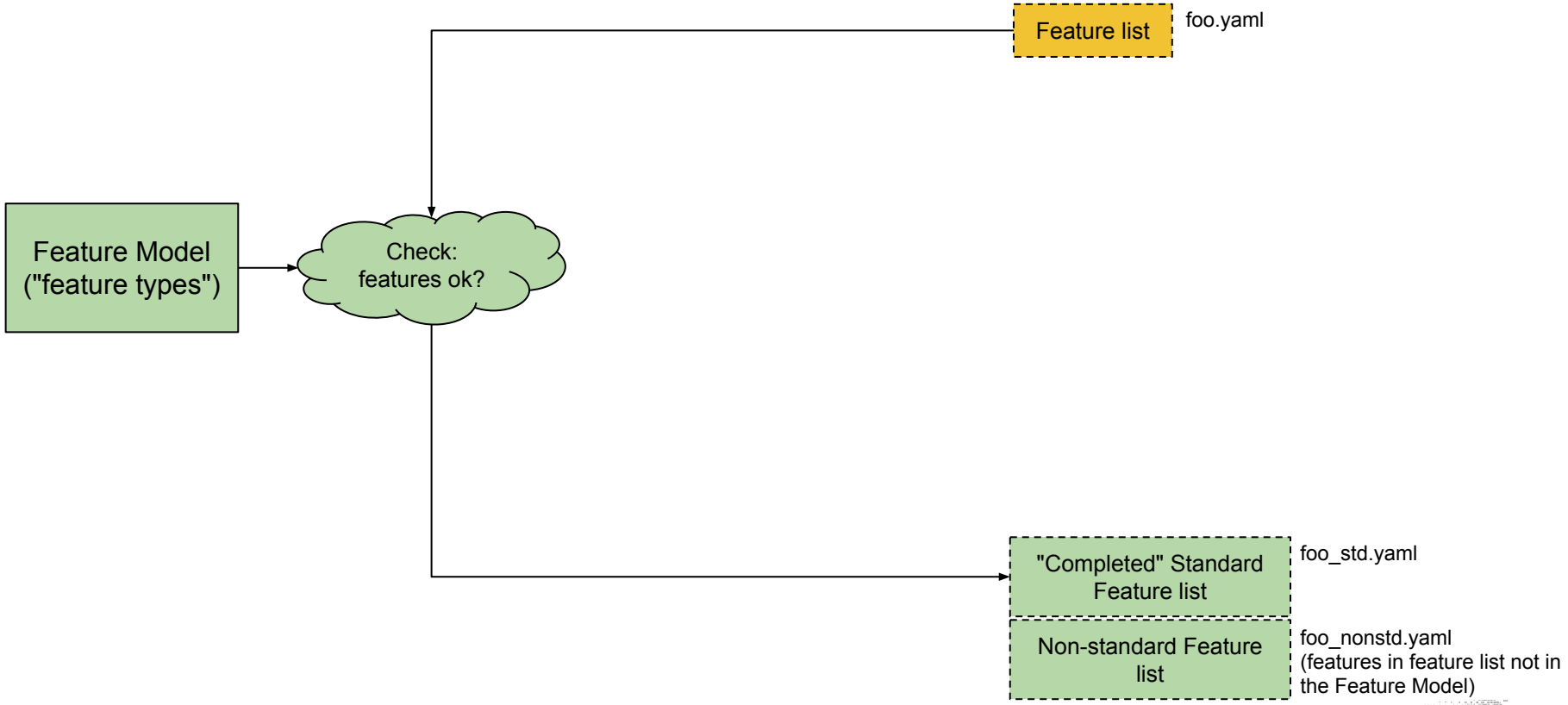
WARL_fn ::= Lambda (var, var) . E

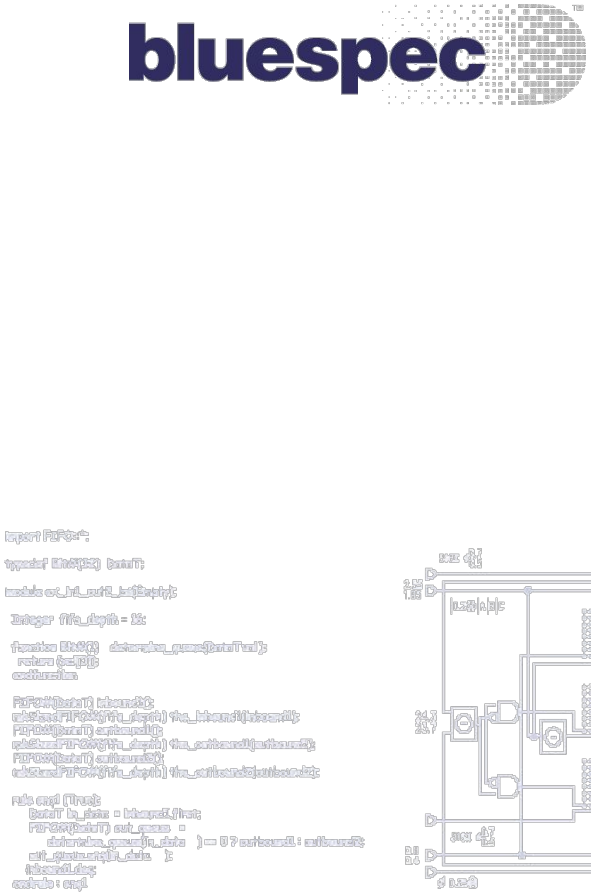


- The constraint on a WARL function is that it produces a legal value for that field (this has to be proved by the tool; this is feasible for this limited language)

Legal_WARL_fn (x, f)

Demo code





End