# A Validated Parser for Stan

Brian Ward

Boston College Computer Science Department 2021

Advisors: Joseph Tassarotti and Jean-Baptiste Tristan

# Two Different Kinds of Programming

## Probabilistic Programming Languages

- Programs describe probability densities and perform inference
  - Compiler actually generates code in a different high-level language
- Examples: **Stan**, BUGS, Edward, Infer.Net

## Proof Assistants

- Programs are conventional software and formal proofs of properties
  - Can also automate parts of these proofs and check their correctness
- Examples: **Coq**, Agda, Isabelle, F*

# A computer can prove things?

```
Theorem plus_assoc : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof.
  intros n m p. induction n as [| n' IHn'].
  - (* n = 0 *)
    reflexivity.
  - (* n = S n' *)
    simpl.
    rewrite -> IHn'.
    reflexivity.
Qed.
```

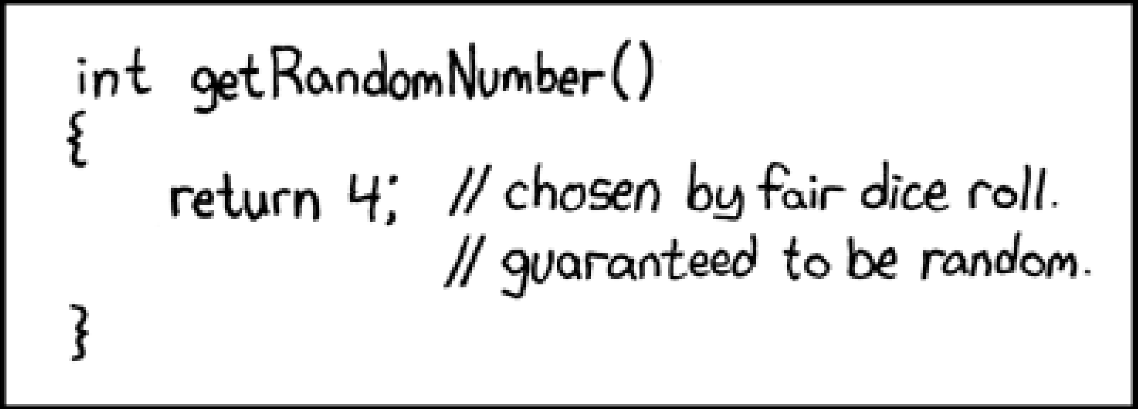
*Theorem*: For any n, m and p,
n + (m + p) = (n + m) + p.

*Proof*: By induction on n.

- First, suppose n = 0. We must show that
  0 + (m + p) = (0 + m) + p.
  - This follows directly from the definition of +.
- Next, suppose n = S n', where
  n' + (m + p) = (n' + m) + p.
  - We must now show that
    (S n') + (m + p) = ((S n') + m) + p.
  - By the definition of +, this follows from
    S (n' + (m + p)) = S ((n' + m) + p),
  - which is immediate from the induction hypothesis.
- *Qed*.

## Why bother?

- Sometimes it’s easier to prove things than test them – especially when randomness is involved!



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

https://xkcd.com/221/

# Parsing

- Compilation step which verifies input is well formed and builds syntax tree
  - Also responsible for syntax errors

E ::= (E + E) | a $\Rightarrow$

```
  +             +
 / \           / \
a   +    ?    +   a
   / \       / \
  a   a     a   a
```

```
>>> print("Hello)
File "<stdin>", line 1
print("Hello)
            ^
SyntaxError: EOL while scanning string literal
```

## How to Verify a Parser

1. Write Coq-friendly grammar specification for Menhir
2. Translate AST and semantic actions into Coq
3. Generate a *sound*, *complete*, and *safe* parser
4. Connect to the rest of your compiler

## How to Improve Your Verified Parser

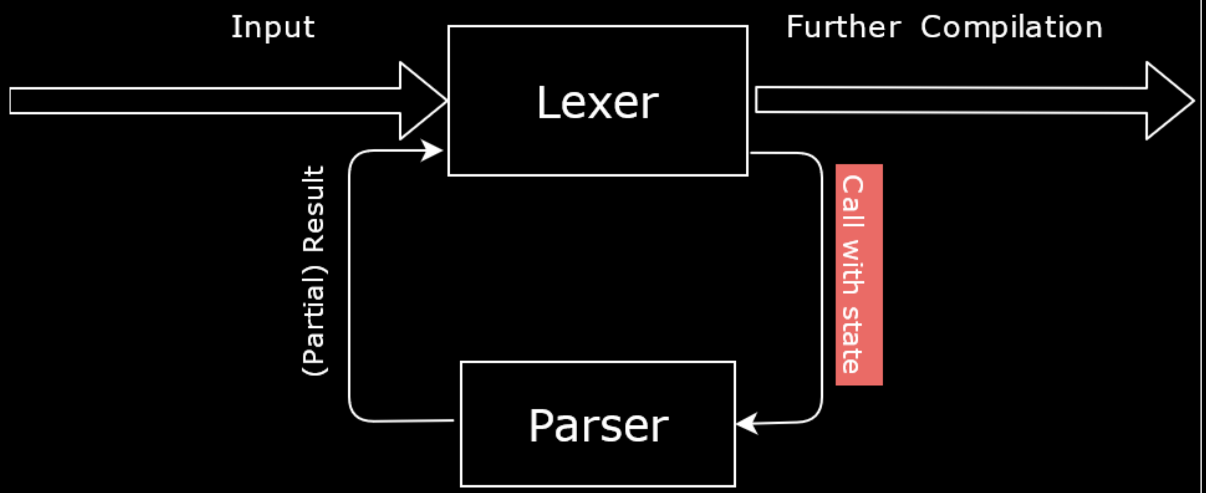
1. Notice an area for improvement in your tools
2. Learn enough to modify them

## Decision: How to handle errors

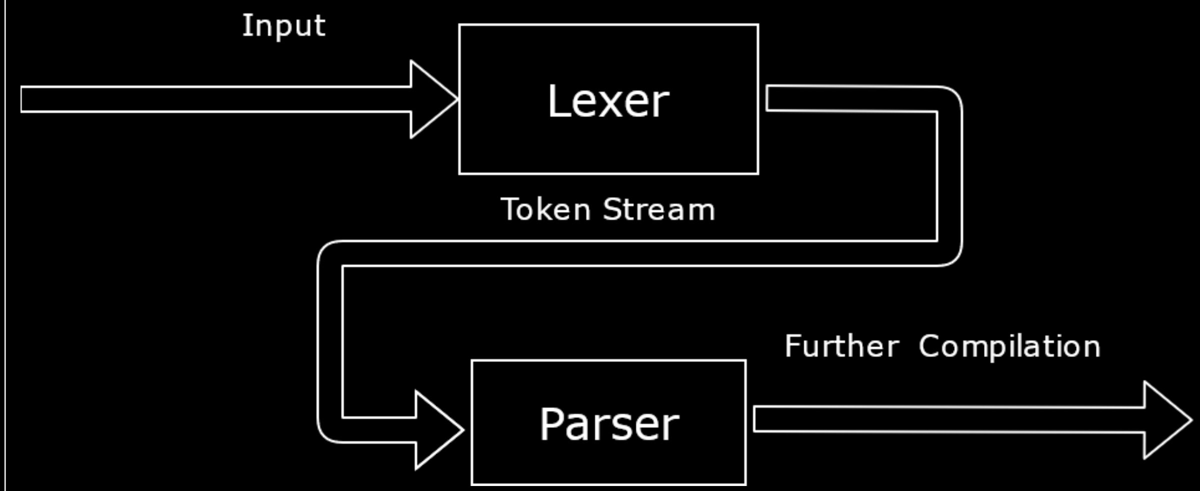
1. Do what CompCert, a large verified compiler, does, and parse the language twice.
   - This runs an unverified parser in an ‘incremental’ mode specifically for errors
2. Allow the Coq mode of Menhir to be run incrementally.

# Who is in charge?

## Table mode puts the lexer in charge



## The verified mode has the parser as a pure function



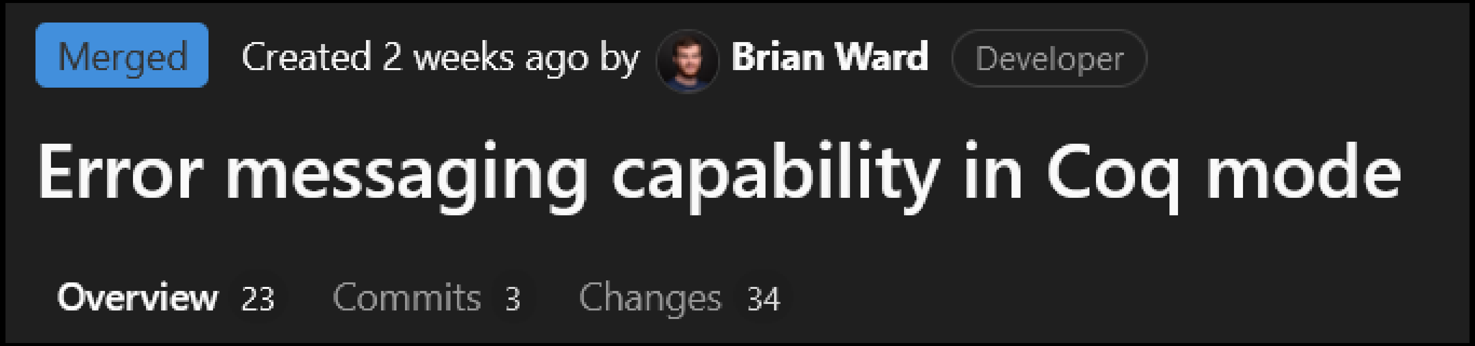
## Decision: How to handle errors

1. Do what CompCert, a large verified compiler, does, and parse the language twice.
   - This runs an unverified parser in an ‘incremental’ mode specifically for errors
2. Allow the Coq mode of Menhir to be run incrementally.
   - This requires trusting the code running the parser.
3. Return extra information if the parser fails.
   - This was ultimately chosen as the simplest and most elegant solution

## How to Improve Your Verified Parser

1. Notice an area for improvement in your tools
2. Learn enough to modify them
3. Make the change and have it included in the tool

3b. Write a few hundred error messages by hand


Merged Created 2 weeks ago by Brian Ward Developer

Error messaging capability in Coq mode

Overview 23 Commits 3 Changes 34

## Further Work

- More features for Menhir: associativity and precedence

Thank you!