Agda formalisation of an elaborator for a language based on simply typed lambda calculus

Zahorán Barnabás

Supervisor:
Kaposi Ambrus

Budapest, 2024

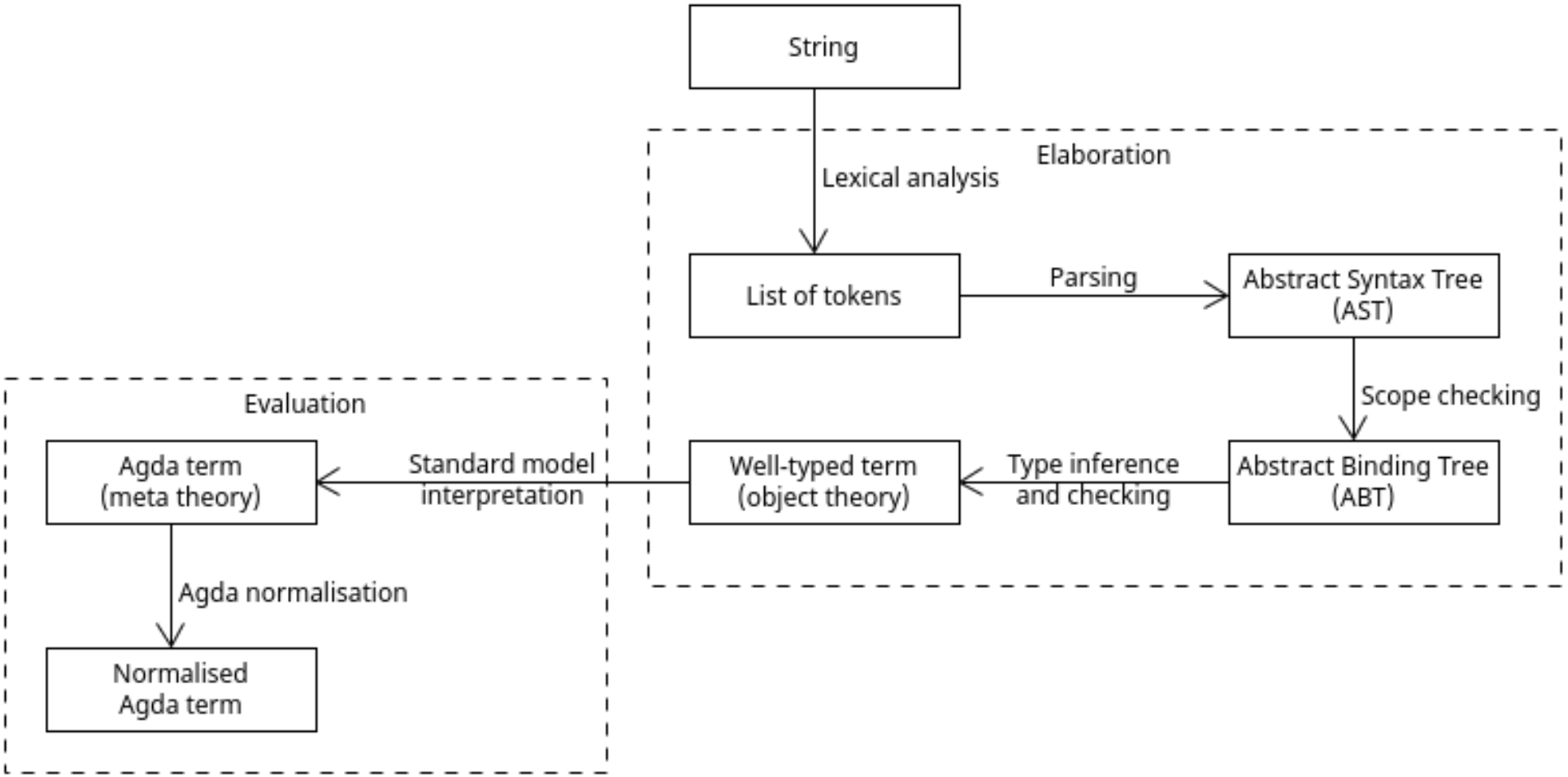
Research goals

- **Elaborator formalisation**
- **Translating strings to an algebraic description (QIIT) of simply typed λ-calculus**
- **Both the models and translation to be correct-by-construction**
- **Implement a framework that is transparent and easy to use for educational or future research purposes**

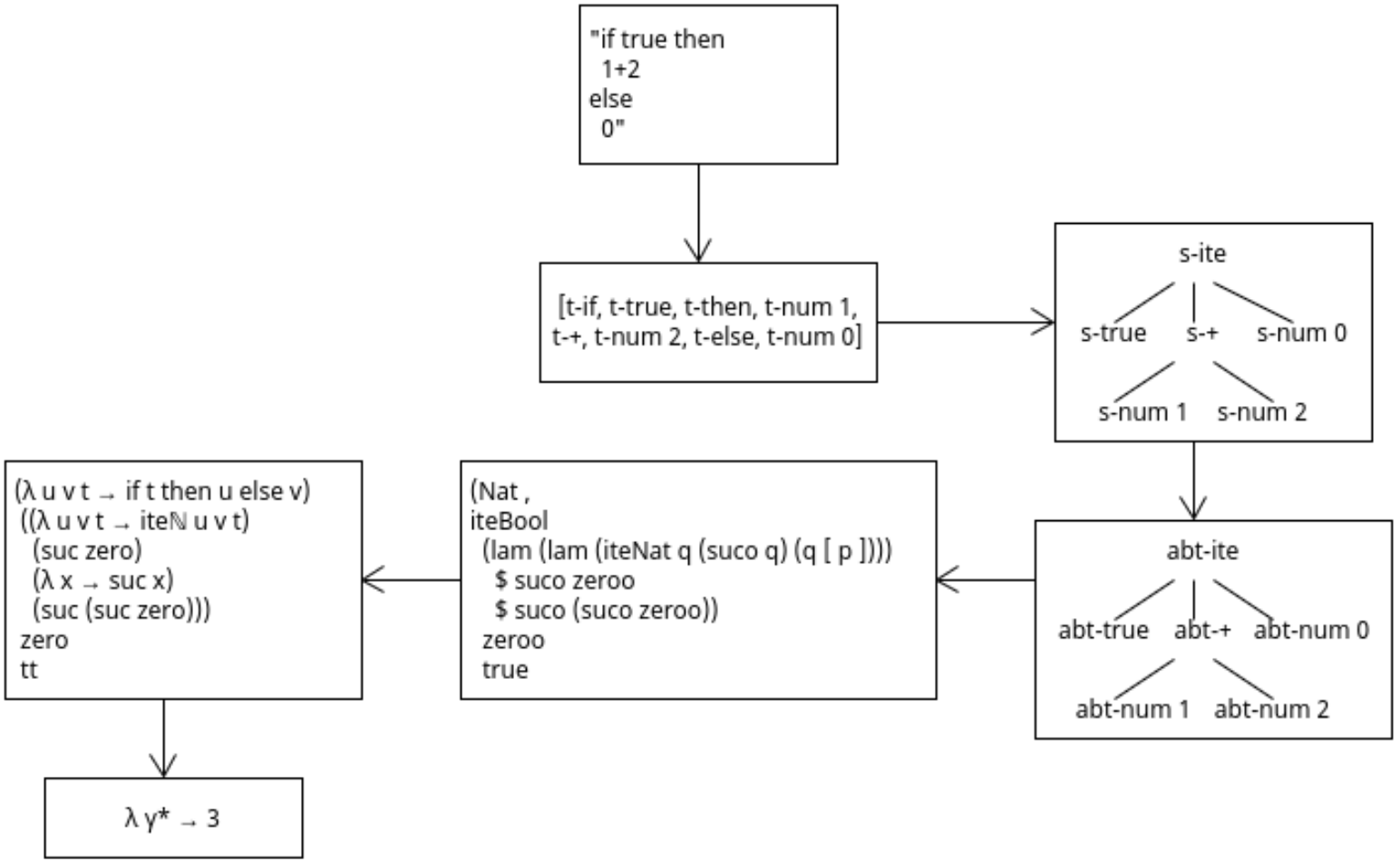
Methods

- **Agda** – programming language and theorem prover based on Intuitionistic type theory
- **Dependent types** – first order statements and proofs by Curry-Howard isomorphism
- **Totality** – termination of functions is guaranteed, no unhandled cases or runtime exceptions
- **agdarsec** – parser combinator library
- **Bidirectional type checking** – inference and checking rules

Levels of Abstraction



Elaboration example



Implementation

- **ABT cannot be badly scoped, well-typed terms cannot be badly typed by definition, e.g.:**

```
data ABT (n : ℕ) : Set where
abt-var : Fin n → ABT n
```

- **elaborate function returns each level of abstraction until evaluation result or error**
- **compile and eval only return well-typed terms and standard model interpretations**
- **Errors are signaled to the user: syntax-error, scope-error, type-error**

Results

- **Easy to use framework had been implemented**
- **Modular**
  Lexer.agda, Parser.agda, Scopecheck.agda, etc.
- **Extendable**
  e.g.: new built-in operations _-_, _*_
- **Performance could be improved in the future (implicit argument instancing)**
- **Tests and examples in the public repository**