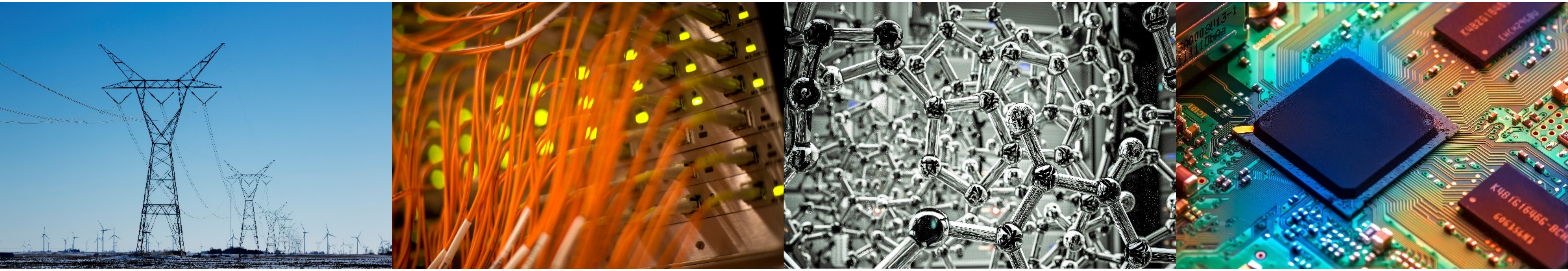
A STATIC SEMANTICS FOR HASKELL UTILIZING THE K-FRAMEWORK

Bradley Morrell

Graduate Student: Liyi Li, Advisor: Professor Elsa Gunter, Coadvisor: Professor Andrew Miller



ILLINOIS

Electrical & Computer Engineering

COLLEGE OF ENGINEERING

K-Framework

- Developed at University of Illinois at Urbana-Champaign
- Used to make executable formal specifications
- Can run example programs
- Can be used with test sets
  - Validate test sets with formal specification
- Can be automatically translated to theorem prover (Isabelle)

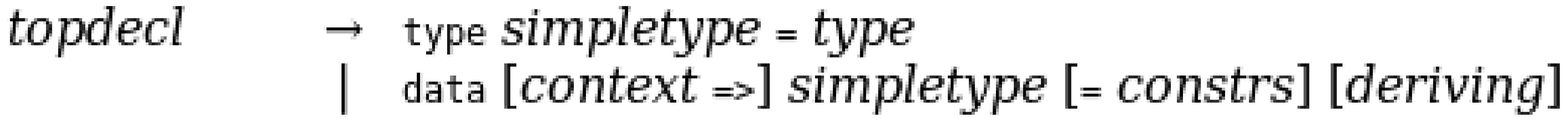
Haskell

- Purely Functional Language
  - Functions are only dependent on input
- Strong Typing
  - Function application applied to only correctly typed arguments
- Static Typing
  - Type Checking run before execution

Syntax

- Implementation of complete syntax of Haskell in K (No Sugar)
- Syntax details exactly what is and what is not a valid expression

Haskell 2010 Report



```
topdecl    →   type simpletype = type
           |   data [context =>] simpletype [= constrs] [deriving]
```

K Syntax

```
syntax TopDecl ::= "type" SimpleType "=" Type [klabel('type)]
                 | "data" OptContext SimpleType OptConstrs OptDeriving  [klabel('data)]
```

Context Sensitive Checks

- From testing the standard Haskell Compiler GHC
- Ensure sanity of syntactically correct programs
- Module system complications

BAD

data ~~Date~~ = Date Int
;type ~~Date~~ = Datetwo Int

BAD

data Date = ~~Date~~ Int
;type Datetwo = ~~Date~~ Int

GOOD

data Date = Date Int
;type Datetwo = Datetwo Int

Type System and Inference

- Gave full formal type system for Haskell
  - As a family of mutually inductive rules
- Implemented type inference for this system
  - Algorithm based on Hindley-Milner
  - Supports Mutual Recursion
    - Default: All declarations in Haskell modules mutually recursive
  - Collected user defined data types
    - Placed into proper type structures

OKAY

```
f x = y x
;y x = f x
```

Conclusion

- Implemented
  - Syntax
  - Checks
  - Type Inference
  - Multiple Modules
- Future Work
  - Fits into complete semantics of Haskell in K