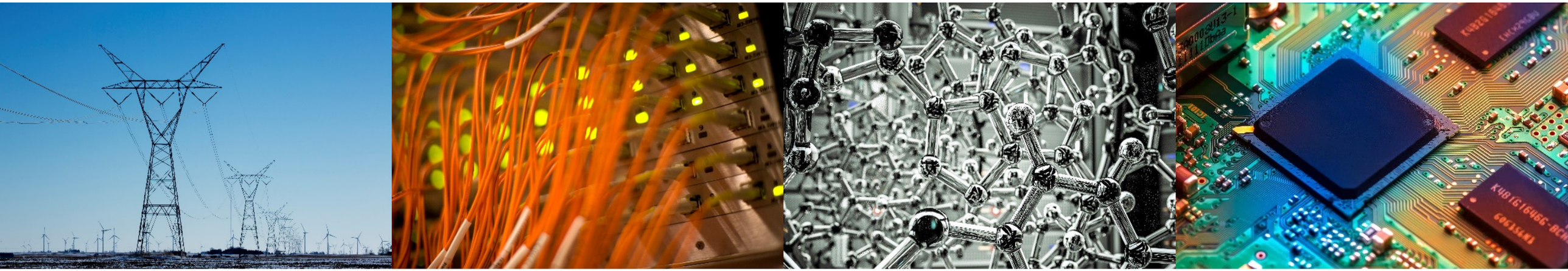
A STATIC SEMANTICS FOR HASKELL UTILIZING THE K-FRAMEWORK

Bradley Morrell
Adviser: Professor Elsa Gunter



ILLINOIS
Electrical & Computer Engineering
COLLEGE OF ENGINEERING

K-Framework

- Used to make executable formal specifications
- Can be used with test sets
- Can be translated to automatic 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
```

Inference

- Collected user defined data types
- Placed into proper type structures
- Inference algorithm similar to Hindley-Milner
- Mutual Recursion
  - Mutually recursive functions are allowed in Haskell

OKAY

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

Conclusion

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