09 - Type Checking

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Maryville COLLEGE

Type Productions

- Each expression produces some type.

Type Productions

- Each expression produces some type.
- Types are context sensitive. Why?

Type Productions

- Each expression produces some type.
- Types are context sensitive. Why?
- The symbol table is an integral part of type checking.

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.
- Type checking is typically performed either during syntax analysis or on a parse tree.

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.
- Type checking is typically performed either during syntax analysis or on a parse tree.
- Each production in the language has a type production. In ledgard, the possible results of a production are:

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.
- Type checking is typically performed either during syntax analysis or on a parse tree.
- Each production in the language has a type production. In ledgard, the possible results of a production are:
  - **Simple Type** - integer, boolean

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.
- Type checking is typically performed either during syntax analysis or on a parse tree.
- Each production in the language has a type production. In ledgard, the possible results of a production are:
  - **Simple Type** - integer, boolean
  - **Type** - integer, boolean, array

Maryville COLLEGE

Type Rules

- Along with the BNF context-free portion of the language, we can specify a series of productions for type rules.
- For instance, in ledgard, the type rules surrounding integer expressions are as follows:
  - $\langle integer - literal \rangle \Rightarrow \{integer\}$
  - $\{integer\}(+|-|*|/)\{integer\} \Rightarrow \{integer\}$
- This forms a grammar which an be readily checked for correctness.
- Type checking is typically performed either during syntax analysis or on a parse tree.
- Each production in the language has a type production. In ledgard, the possible results of a production are:
  - **Simple Type** - integer, boolean
  - **Type** - integer, boolean, array
  - **void** - Nothing is returned

Maryville COLLEGE

Type Representation

- Types can be represented in much the same way as lexemes.

Type Representation

- Types can be represented in much the same way as lexemes.
- As the parse tree is constructed, the types are checked, and the types productions are noted in the tree.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.
- Reporting can be tricky in situations with multiple legal productions.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.
- Reporting can be tricky in situations with multiple legal productions.
- The basic procedure is this:

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.
- Reporting can be tricky in situations with multiple legal productions.
- The basic procedure is this:
  1. Check the types of the current parse tree node.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.
- Reporting can be tricky in situations with multiple legal productions.
- The basic procedure is this:
  1. Check the types of the current parse tree node.
  2. If there is an error, report it.

Type Error Reporting and Recovery

- Error checking and reporting is analogous to that of parsing.
- Recovery is typically easier, because for each production we typically know a valid type production.
- Reporting can be tricky in situations with multiple legal productions.
- The basic procedure is this:
  1. Check the types of the current parse tree node.
  2. If there is an error, report it.
  3. Mark the current node as if the production succeeded and continue (or maybe just stop all together!).

Exercise: Looplang Type Checking

- How many types does looplang have?

Exercise: Looplang Type Checking

- How many types does looplang have?
- Let's write the type rules for looplang!

Exercise: Looplang Type Checking

- How many types does looplang have?
- Let's write the type rules for looplang!
- Now, let's add type checking to looplang.