Semantic Analysis

Compiler Construction

2024-11-19

Prof. Dr. Michael Kuhn

michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

Semantic Analysis

- Overview
- Type Checking
- B-Minor
- Summary

Semantic Analysis

- Semantic analysis analyzes semantics, that is, meaning
- Type checking allows catching type errors at compile time
    - Requires name resolution and a symbol table
- Can also include checking array bounds, pointer traversal, control flow etc.

Type Systems

- Type systems can help with correctness, performance and expressiveness
- Safe vs. unsafe, static vs. dynamic and explicit vs. implicit
- B-Minor is safe, static and explicit

Outline

Semantic Analysis

- Overview
- Type Checking
- B-Minor
- Summary

Atomic and Compound Types

- `integer`: 64-bit integer
- `boolean`: `true` or `false`
- `char`: ASCII
- `string`: ASCII, null-terminated
- `void`: function with no return value

- `array [size] type`
- `function type ( a: type, ... )`

Group Exercise

- Determine the symbol table and look up variables for the following program.

```
a: integer = 42;
f: function void ( x: boolean ) = {
    if (x) {
        # TODO: Symbol table and symbol lookup for "a" and "f"
    } else
        a: integer;
        for (a = 0; a < 10; a++)
            # TODO: Symbol table and symbol lookup for "a" and "f"
}
main: function void () = {
    a: boolean = true;
    # TODO: Symbol table and symbol lookup for "a" and "f"
    f(a)
}
```

Group Exercise

- Determine the symbol table and look up variables for the following program.

Name	Kind	Type
a	local	boolean
a	global	integer
f	global	function
main	global	function

Name	Kind	Type
	local	
	local	
x	param	boolean
a	local	boolean
a	global	integer
f	global	function
main	global	function

Name	Kind	Type
	local	
x	param	boolean
a	local	boolean
a	global	integer
f	global	function
main	global	function

Group Exercise

- Perform type checking for the following program.

```
f: function boolean ( x: array [20] integer ) = {
    if (x[0] == 42 || !x[1])
        return false;
    return (x > 0);
}
main: function void () = {
    a: integer = true;
    b: boolean = (a < 23);
    c: array [10] array [10] integer;
    c[0] = c[1] + c[2];
    f(c[0]);
}
```

Outline

Semantic Analysis

- Overview
- Type Checking
- B-Minor
- Summary

Group Exercise

- Create a context-free grammar for B-Minor in the following order: types, expressions, statements and declarations.
    - `INTEGER`, `STRING` and `NAME` stand for the respective types

Group Exercise

- Create a context-free grammar for B-Minor in the following order: types, expressions, statements and declarations.
  - `INTEGER`, `STRING` and `NAME` stand for the respective types

5. T ⟶ void | boolean | char | integer | string | array [E] T | function T (P)
6. P ⟶ N: T | N: T, P
7. N ⟶ `NAME`

Group Exercise

- Create a context-free grammar for B-Minor in the following order: types, expressions, statements and declarations.
  - `INTEGER`, `STRING` and `NAME` stand for the respective types

3. $E \longrightarrow E = E \mid E \mathbin{\&\&} E \mid E \parallel E \mid E < E \mid E <= E \mid E > E \mid E >= E \mid E == E \mid E \mathbin{!=} E \mid E + E \mid E - E \mid E * E \mid E / E \mid E \% E \mid E \hat{} E \mid -E \mid !E \mid E{+}{+} \mid E{-}{-} \mid E[E] \mid E() \mid E(E_A) \mid \texttt{INTEGER} \mid \texttt{STRING} \mid N$
4. $E_A \longrightarrow E \mid E, E_A$
5. $T \longrightarrow$ void | boolean | char | integer | string | array [E] T | function T (P)
6. $P \longrightarrow N{:}\ T \mid N{:}\ T, P$
7. $N \longrightarrow \texttt{NAME}$

Group Exercise

- Create a context-free grammar for B-Minor in the following order: types, expressions, statements and declarations.
    - `INTEGER`, `STRING` and `NAME` stand for the respective types

2. S → D | E | if (E) S | if (E) S else S | for (E; E; E) S | print E | return E | { S } | S ; S
3. E → E = E | E && E | E || E | E < E | E <= E | E > E | E >= E | E == E | E != E | E + E | E - E | E * E | E / E | E % E | E ^ E | -E | !E | E++ | E-- | E[E] | E() | E(E_A) | `INTEGER` | `STRING` | N
4. E_A → E | E, E_A
5. T → void | boolean | char | integer | string | array [E] T | function T (P)
6. P → N: T | N: T, P
7. N → `NAME`

Group Exercise

- Create a context-free grammar for B-Minor in the following order: types, expressions, statements and declarations.
  - `INTEGER`, `STRING` and `NAME` stand for the respective types

1. D → N: T; | N: T = E; | N: function T (P) = { S ; } | D D
2. S → D | E | if (E) S | if (E) S else S | for (E; E; E) S | print E | return E | { S } | S ; S
3. E → E = E | E && E | E || E | E < E | E <= E | E > E | E >= E | E == E | E != E | E + E | E - E | E * E | E / E | E % E | E ^ E | -E | !E | E++ | E-- | E[E] | E() | E(E_A) | `INTEGER` | `STRING` | N
4. E_A → E | E, E_A
5. T → void | boolean | char | integer | string | array [E] T | function T (P)
6. P → N: T | N: T, P
7. N → `NAME`

Outline

Semantic Analysis

- Overview
- Type Checking
- B-Minor
- Summary

Summary

- Semantic analysis contains name resolution and type checking
    - Not all programs generated by the grammar might be accepted
- Helpful error messages are important for debugging
    - Messages should contain code positions
- It is not straightforward to come up with an LR grammar for B-Minor
    - Naive grammars contain shift-reduce conflicts

References

[Thain, 2020] Thain, D. (2020). ***Introduction to Compilers and Language Design: Second Edition.*** `http://compilerbook.org/`.