Intermediate Representation

Compiler Construction

2024-11-26

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

Intermediate Representation

- Overview
- B-Minor
- Summary

Intermediate Representation

- Intermediate representation (IR) is somewhere between source and target language
- Optimizations are applied to the intermediate representation

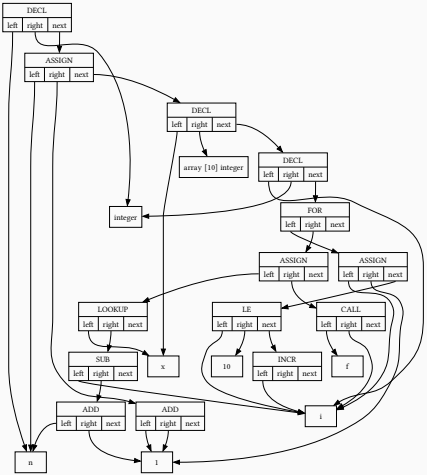
Group Exercise

- Create a DAG for the following program.

```
1  main: function void () = {
2      n: integer = 1 + 1;
3      x: array [10] integer;
4      i: integer;
5      for (i = 1; i <= 10; i++) {
6          x[i - n + 1] = f(i);
7      }
8  }
```

Group Exercise

- Create a DAG for the following program.



Group Exercise

- How would constant folding look like for the previous example?

Group Exercise

- How would constant folding look like for the previous example?
- ASSIGN of ADD 1 + 1 becomes ASSIGN of 2

Group Exercise

- How would constant folding look like for the previous example?
- ASSIGN of ADD 1 + 1 becomes ASSIGN of 2
- `i - n + 1` would require constant propagation

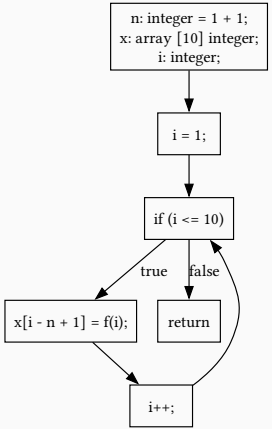
Group Exercise

- Create a CFG for the following program.

```
1  main: function void () = {
2      n: integer = 1 + 1;
3      x: array [10] integer;
4      i: integer;
5      for (i = 1; i <= 10; i++) {
6          x[i - n + 1] = f(i);
7      }
8  }
```

Group Exercise

- Create a CFG for the following program.



Group Exercise

- Create the SSA form for the following snippet.

```
x: integer = 23;
if (x < 10) {
    x = x + 10;
} else if (x < 20) {
    x = x + 20;
} else {
    x = x + 42;
}
```

Group Exercise

- Create the SSA form for the following snippet.

```
int x_1 = 23;
if (x_1 < 10) {
    x_2 = x_1 + 10;
} else if (x_1 < 20) {
    x_3 = x_1 + 20;
} else {
    x_4 = x_1 + 42;
}
x_5 = phi(x2, phi(x_3, x_4))
```

Outline

Intermediate Representation

- Overview
- B-Minor
- Summary

Group Exercise

- Convert the following CFG into an LR grammar.

1. P $\longrightarrow$ E
2. E $\longrightarrow$ E + E
3. E $\longrightarrow$ `INTEGER`

Group Exercise

- Convert the following CFG into an LR grammar.

1. P ⟶ E
2. E ⟶ E + E
3. E ⟶ `INTEGER`

1. P ⟶ E
2. E ⟶ E + E’
3. E ⟶ E’
4. E’ ⟶ `INTEGER`

Group Exercise

- Convert the B-Minor CFG into an LR grammar.

1. P → D
2. D → N: T; | N: T = E; | N: function T (A) = { S ; } | D D
3. S → D | E | if (E) S | if (E) S else S | for (E; E; E) S | print E | return E | { S } | S ; S
4. 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
5. E_A → E | E, E_A
6. T → void | boolean | char | integer | string | array [E] T ~~| function T (A)~~
7. A → N: T | N: T, A
8. N → `NAME`

Group Exercise

- Convert the B-Minor CFG into an LR grammar.

7. $E_A \rightarrow E \mid E, E_A$
8. $T \rightarrow$ void | boolean | char | integer | string | array [E] T
9. $A \rightarrow N{:}\ T \mid N{:}\ T, A$
10. $N \rightarrow$ NAME

Group Exercise

- Convert the B-Minor CFG into an LR grammar.

1. P $\longrightarrow$ DL
2. DL $\longrightarrow$ D | D DL
3. D $\longrightarrow$ N: T; | N: T = E; | N: function T (A) = { SL }

7. $E_A \longrightarrow$ E | E, E_A
8. T $\longrightarrow$ void | boolean | char | integer | string | array [E] T
9. A $\longrightarrow$ N: T | N: T, A
10. N $\longrightarrow$ `NAME`

- Convert the B-Minor CFG into an LR grammar.

1. P → DL
2. DL → D | D DL
3. D → N: T; | N: T = E; | N: function T (A) = { SL }
4. SL → S ; | S ; SL
5. S → N: T; | N: T = E; | E | if (E) { SL } | if (E) { SL } else { SL } | for (E; E; E) { SL } | print E | return E | { SL }

7. E_A → E | E, E_A
8. T → void | boolean | char | integer | string | array [E] T
9. A → N: T | N: T, A
10. N → NAME

Group Exercise

- Convert the B-Minor CFG into an LR grammar.

1. P → DL
2. DL → D | D DL
3. D → N: T; | N: T = E; | N: function T (A) = { SL }
4. SL → S ; | S ; SL
5. S → N: T; | N: T = E; | E | if (E) { SL } | if (E) { SL } else { SL } | for (E; E; E) { SL } | print E | return E | { SL }
6. ...
7. E_A → E | E, E_A
8. T → void | boolean | char | integer | string | array [E] T
9. A → N: T | N: T, A
10. N → NAME

Group Exercise

- Convert the B-Minor CFG into an LR grammar.

1. $E \rightarrow E_0$
2. $E_0 \rightarrow E_0 = E_1 \mid E_1$
3. $E_1 \rightarrow E_1 \,||\, E_2 \mid E_2$
4. $E_2 \rightarrow E_2 \,\&\&\, E_3 \mid E_3$
5. $E_3 \rightarrow E_3 == E_4 \mid E_3 \,!= E_4 \mid E_4$
6. $E_4 \rightarrow E_4 < E_5 \mid E_4 <= E_5 \mid E_4 > E_5 \mid E_4 >= E_5 \mid E_5$
7. $E_5 \rightarrow E_5 + E_6 \mid E_5 - E_6 \mid E_6$
8. $E_6 \rightarrow E_6 * E_7 \mid E_6 / E_7 \mid E_6 \,\%\, E_7 \mid E_7$
9. $E_7 \rightarrow \mid E_7 \,\hat{}\, E_8 \mid E_8$
10. $E_8 \rightarrow -E_8 \mid !E_8 \mid E_9$
11. $E_9 \rightarrow E_9{+}{+} \mid E_9{-}{-} \mid E_9[E_{10}] \mid E_9() \mid E_9(E_A) \mid E_{10}$
12. $E_{10} \rightarrow \texttt{INTEGER} \mid \texttt{STRING} \mid N$

Outline

Intermediate Representation

Overview

B-Minor

Summary

Summary

- DAGs and CFGs can be easier to handle than the AST
  - DAG represents expressions well, while CFGs show control flow
- Optimizations like constant folding can be applied to the graphs
  - More sophisticated optimizations are typically applied to the IR
- IR often uses SSA form
  - Makes it easier to determine the lifetime of variables

References

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