Compiler Design

Lecture-24

Local Optimization

Topics Covered

- Optimization of Basic Blocks
- DAG representation of Basic Block
- Construction of DAG

Optimization of Basic Blocks

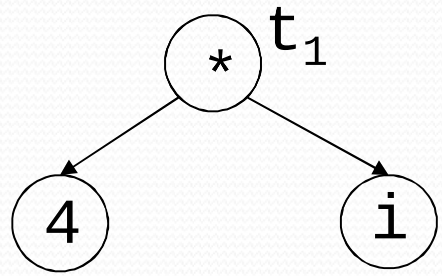
- Many structure preserving transformations can be implemented by construction of DAGs of basic blocks

DAG representation of Basic Block (BB)

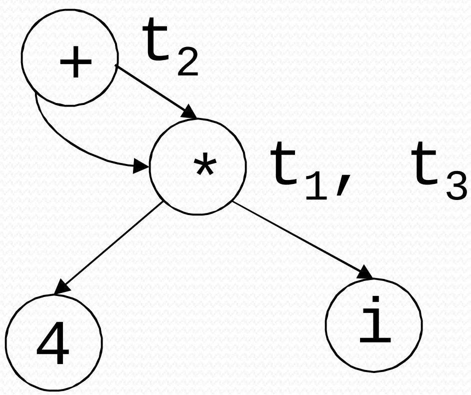
- Leaves are labeled with unique identifier (var name or const)
- Interior nodes are labeled by an operator symbol
- Nodes optionally have a list of labels (identifiers)
- Edges relates operands to the operator (interior nodes are operator)
- Interior node represents computed value
  - Identifier in the label are deemed to hold the value

Example: DAG for BB

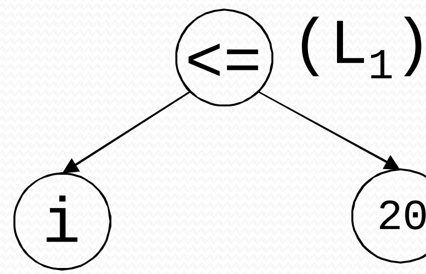
$t_1 := 4 * i$



$t_1 := 4 * i$
$t_3 := 4 * i$
$t_2 := t_1 + t_3$



if (i <= 20)goto L_1



Construction of DAGs for BB

- I/p: Basic block, *B*
- O/p: A DAG for *B* containing the following information:
  1) A label for each node
  2) For leaves the labels are ids or consts
  3) For interior nodes the labels are operators
  4) For each node a list of attached ids (possible empty list, no consts)

Construction of DAGs for BB

- Data structure and functions:
  - Node:
    1) Label: label of the node
    2) Left: pointer to the left child node
    3) Right: pointer to the right child node
    4) List: list of additional labels (empty for leaves)
  - **Node (*id*)**: returns the most recent node created for *id*. Else return *undef*
  - **Create(*id,l,r*)**: create a node with label *id* with *l* as left child and *r* as right child. *l* and *r* are optional params.

Construction of DAGs for BB

- Method:

  For each 3AC, *A* in *B*

  A if of the following forms:

  1. $x := y$ op z
  2. $x :=$ op y
  3. $x := y$

  1. if ((n_y = node(y)) == *undef*)

     n_y = Create (y);

     if (A == type 1)

     and ((n_z = node(z)) == *undef*)

     n_z = Create(z);

Construction of DAGs for BB

2. If (A == type 1)

   Find a node labelled '*op*' with left and right as n_y and n_z respectively [determination of common sub-expression]

   If (not found) n = Create (op, n_y, n_z);

   If (A == type 2)

   Find a node labelled '*op*' with a single child as n_y

   If (not found) n = Create (op, n_y);

   If (A == type 3) n = Node (y);

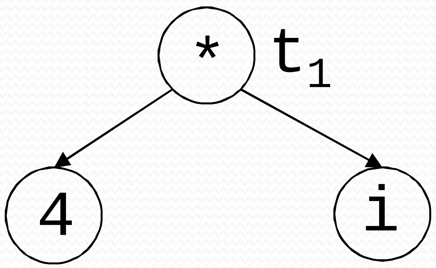
3. Remove x from Node(x).list

   Add x in n.list

   Node(x) = n;

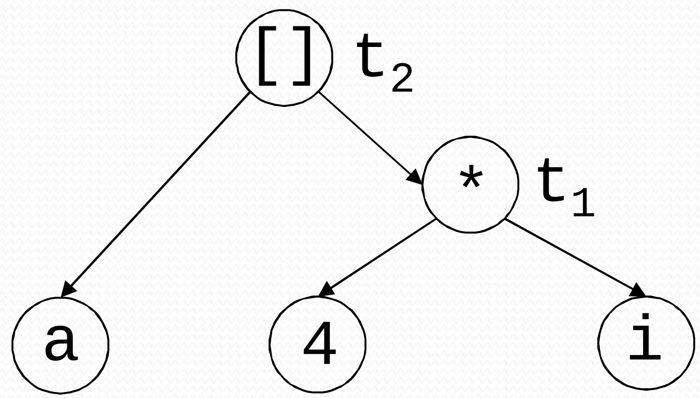
Example: DAG construction from BB

t_1 := 4 * i



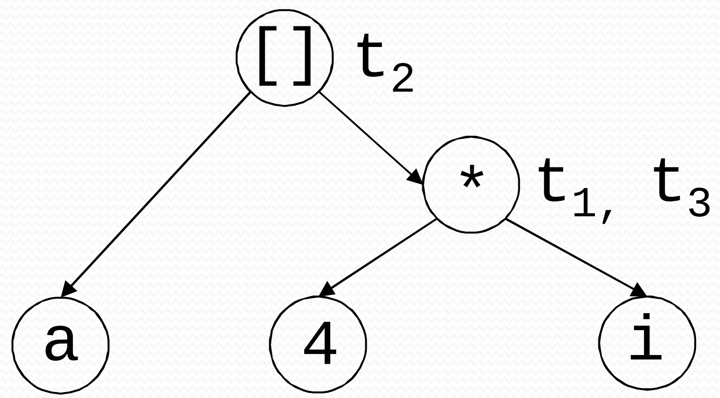
Example: DAG construction from BB

```
t1 := 4 * i
t2 := a [ t1 ]
```



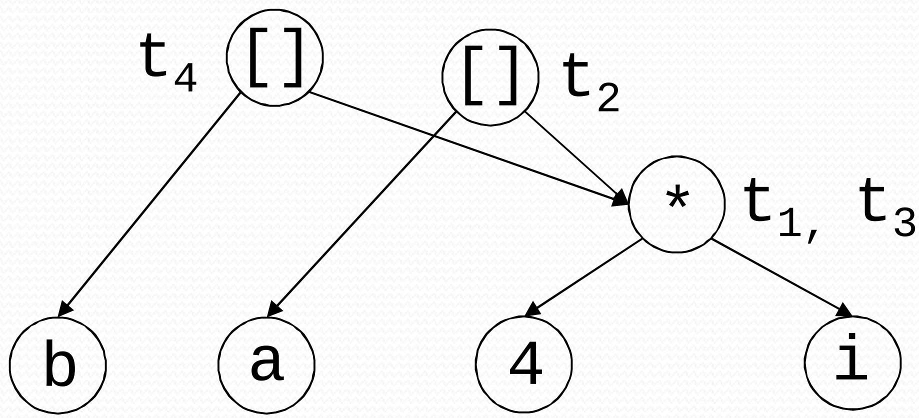
Example: DAG construction from BB

```
t1 := 4 * i
t2 := a [ t1 ]
t3 := 4 * i
```



Example: DAG construction from BB

```
t1 := 4 * i
t2 := a [ t1 ]
t3 := 4 * i
t4 := b [ t3 ]
```



Example: DAG construction from BB

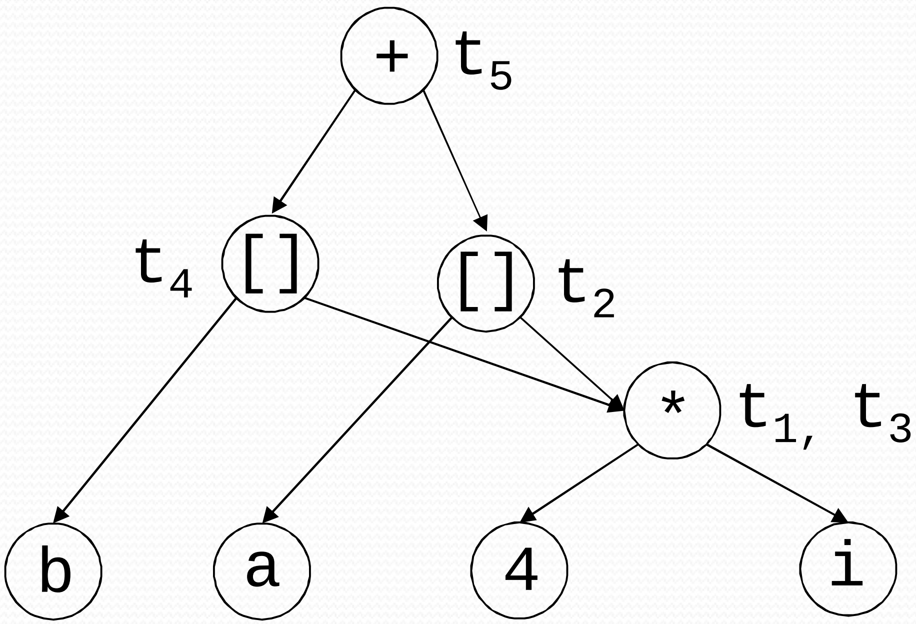
t_1 := 4 * i

t_2 := a [t_1]

t_3 := 4 * i

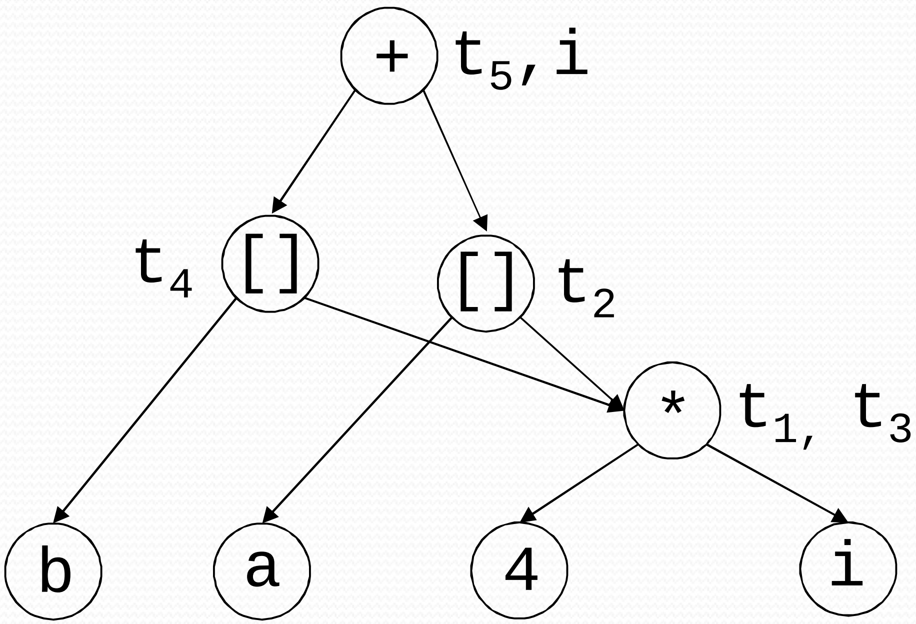
t_4 := b [t_3]

t_5 := t_2 + t_4



Example: DAG construction from BB

```
t1 := 4 * i
t2 := a [ t1 ]
t3 := 4 * i
t4 := b [ t3 ]
t5 := t2 + t4
i := t5
```



DAG of a Basic Block

- Observations:
  - A leaf node for the initial value of an id
  - A node *n* for each statement *s*
  - The children of node *n* are the last definition (prior to *s*) of the operands of *n*

Optimization of Basic Blocks

- Common sub-expression elimination: by construction of DAG
  - Note: for common sub-expression elimination, we are actually targeting for expressions that compute the same value.

```
a := b + c
b := b – d
c := c + d
e := b + c
```

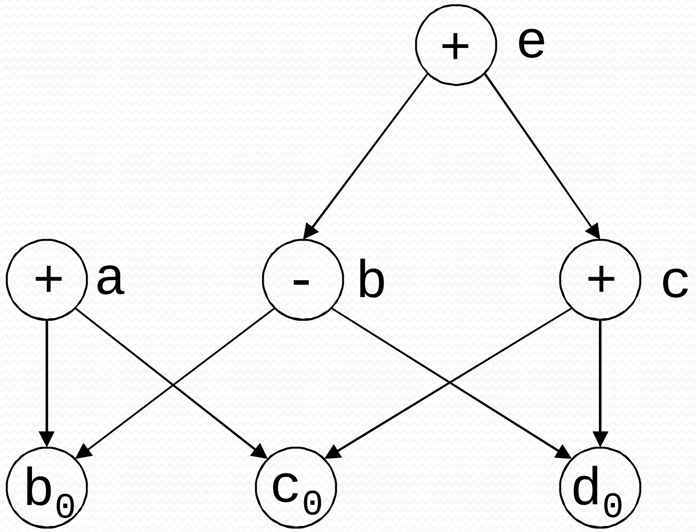
Common expressions

But do not generate the same result

Optimization of Basic Blocks

- DAG representation identifies expressions that yield the same result

```
a := b + c
b := b – d
c := c + d
e := b + c
```

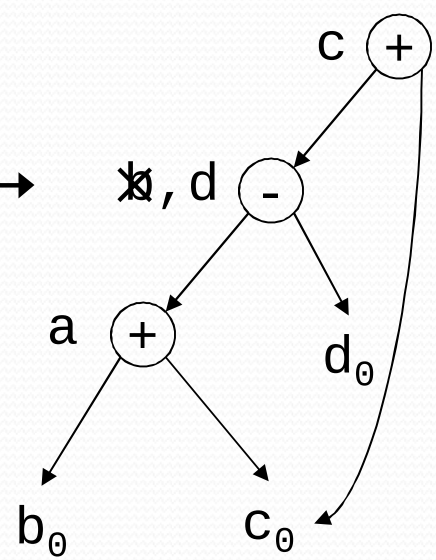


Optimization of Basic Blocks

- Dead code elimination: Code generation from DAG eliminates dead code.

```
a := b + c
b := a – d
d := a – d
c := d + c
```

b is not live




```
a := b + c
d := a - d
c := d + c
```