Generalized Index-Set Splitting

Arie Tal

2003/10/07

CASCON 2003

Generalized Index-Set Splitting

- Overview
- Compile-time unknown split points
- Loops with multiple split points
- Removing “dead” inductive-branches
- Nested inductive-branches
- Controlling code growth

Index-Set Splitting - Overview

- A loop transformation that divides the index-set (range) of a loop into sub-ranges. Each sub-range is then handled as a separate loop.
- Purpose: removing inductive-branches.

```
for (i=0; i < 100; i++) {
   if (i < 5)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Index-Set Splitting - Overview

- A loop transformation that divides the index-set (range) of a loop into sub-ranges. Each sub-range is then handled by a separate loop.
- Purpose: removing inductive-branches.

```
for (i=0; i < 100; i++) {
   if (i < 5)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

```
for (i=0; i < 5; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=5; i < 100; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Index-Set Splitting - Overview

- Some inductive-branches may take three sub-ranges to eliminate:

```
for (i=0; i < 100; i++) {
   if (i != 20)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Index-Set Splitting - Overview

- Some inductive-branches may take three sub-ranges to eliminate:

```
for (i=0; i < 100; i++) {
   if (i != 20)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

→

```
for (i=0; i < 20; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=20; i < 21; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
for (i=21; i < 100; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
```

Index-Set Splitting - Overview

- Some inductive-branches may take three sub-ranges to eliminate:

```
for (i=0; i < 100; i++) {
   if (i != 20)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

```
for (i=0; i < 20; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}

a[20] = 5*a[20];
b[20] = a[20]*a[20];

for (i=21; i < 100; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- When the lower-bound, upper-bound or split points are unknown at compile time:

```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- When the lower-bound, upper-bound or split points are unknown at compile time:

```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

```
for (i=0; i < m; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=m; i < 100; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- When the lower-bound, upper-bound or split points are unknown at compile time:

```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   else
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

→

```
for (i=0; i < m; i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=m; i < 100; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

- Problem: m may be smaller than 0 or greater than 100

Compile-time unknown spilt points

- To overcome the problem, we will use the following sub-ranges:
  - First loop: 0 – min(m, 100)
  - Second loop: max(m, 0) – 100

```
for (i=0; i < min(m,100); i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,0); i < 100; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- The cases:
  - m < 0: Only the second loop will iterate from 0 to 100.
  - 0 ≤ m < 100: The first loop will iterate from 0 to m, and the second loop will iterate from m to 100.
  - 100 < m: Only the first loop will iterate from 0 to 100.

```
for (i=0; i < min(m,100); i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,0); i < 100; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- In general:
  - Given a range with lower bound lb, upper bound ub, and a split point sp, we define two sub-ranges:
    - lb – min(sp, ub)
    - max(sp, lb) - ub

```
for (i=lb; i < min(m,ub); i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,lb); i < ub; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Compile-time unknown spilt points

- In general:
  - Given a range with lower bound lb, upper bound ub, and a split point sp, we define two sub-ranges:
    - lb – min(sp, ub)
    - max(sp, lb) - ub

```
for (i=lb; i < min(m,ub); i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,lb); i < ub; i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

No other conditions are needed in addition to the loop structure.

Loops with multiple split points

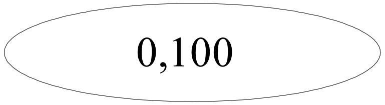
- Multiple split points can be handled iteratively

```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

Loops with multiple split points

- Multiple split points can be handled iteratively
- Instead, we are going to build a sub-range tree, first:

```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```

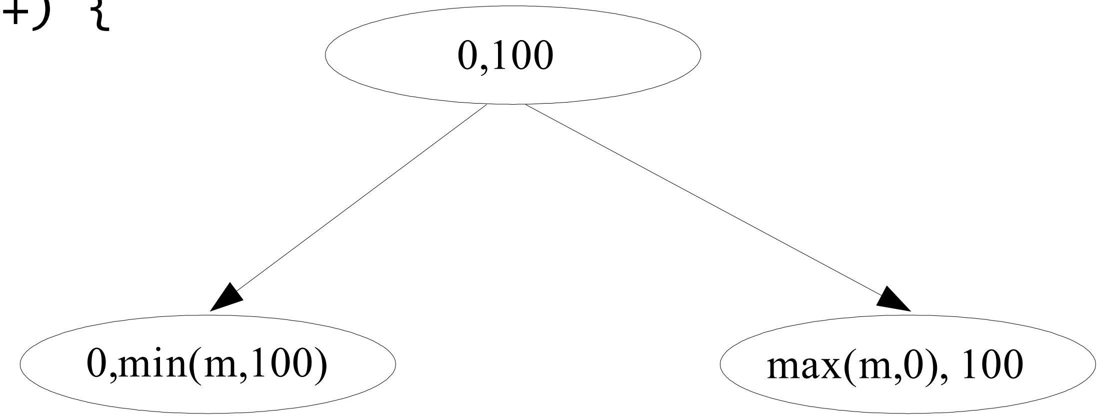


0,100

Loops with multiple split points

- Multiple split points can be handled iteratively
- Instead, we are going to build a sub-range tree, first:

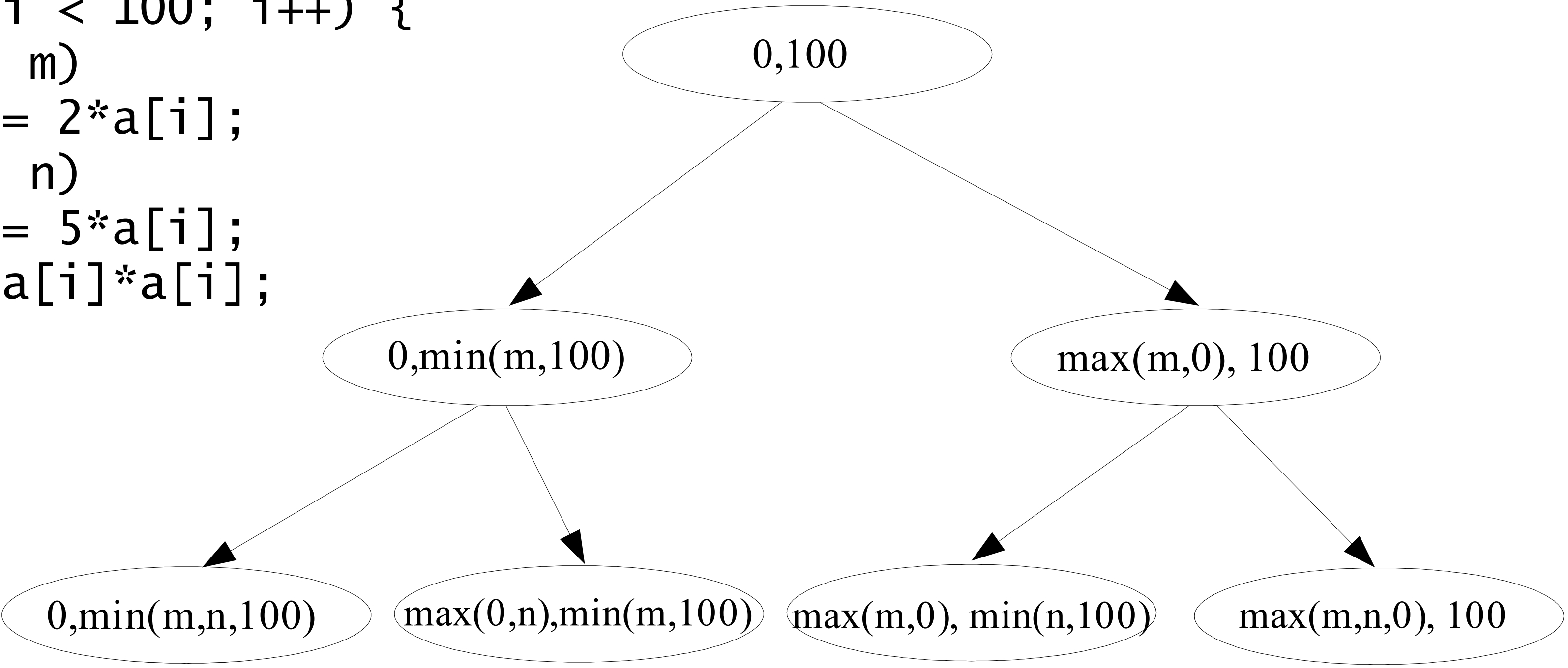
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Loops with multiple split points

- Multiple split points can be handled iteratively
- Instead, we are going to build a sub-range tree, first:

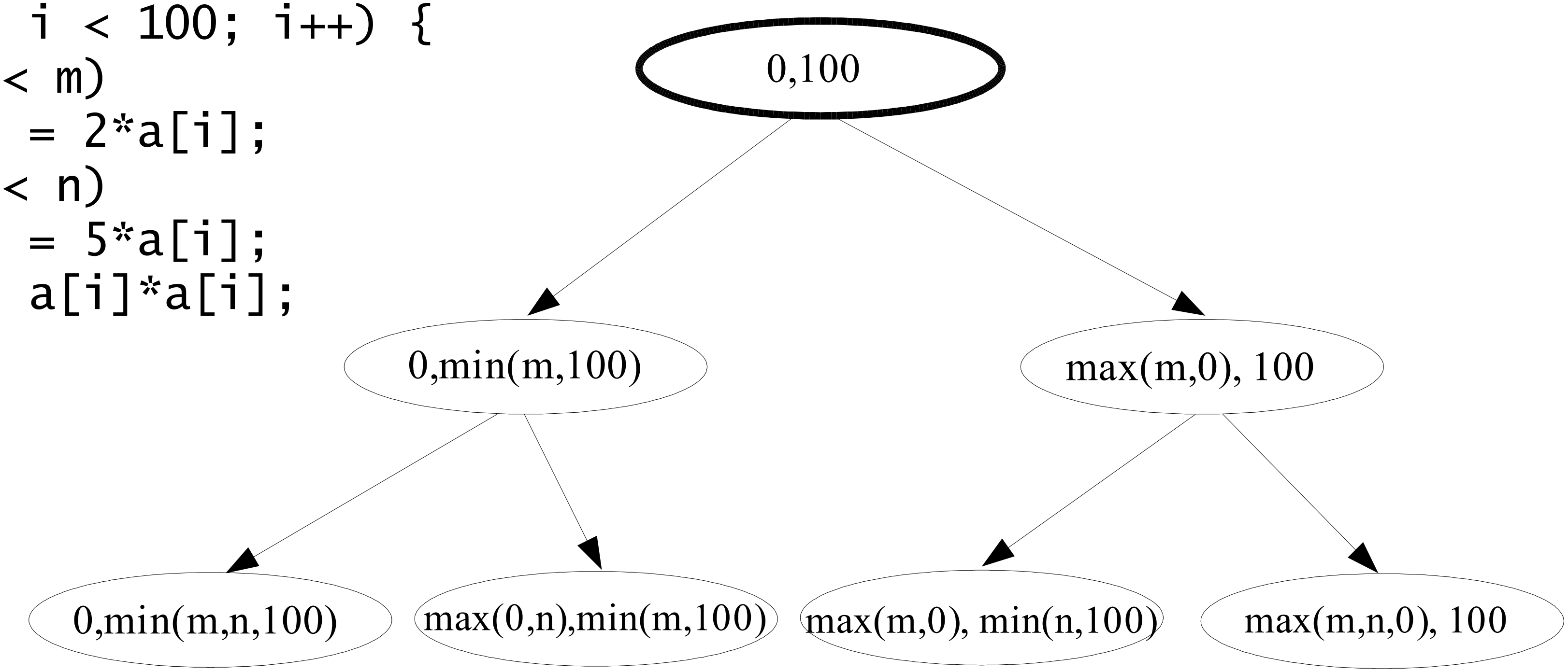
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Loops with multiple split points

- The first level in the sub-range tree corresponds to the original loop range.

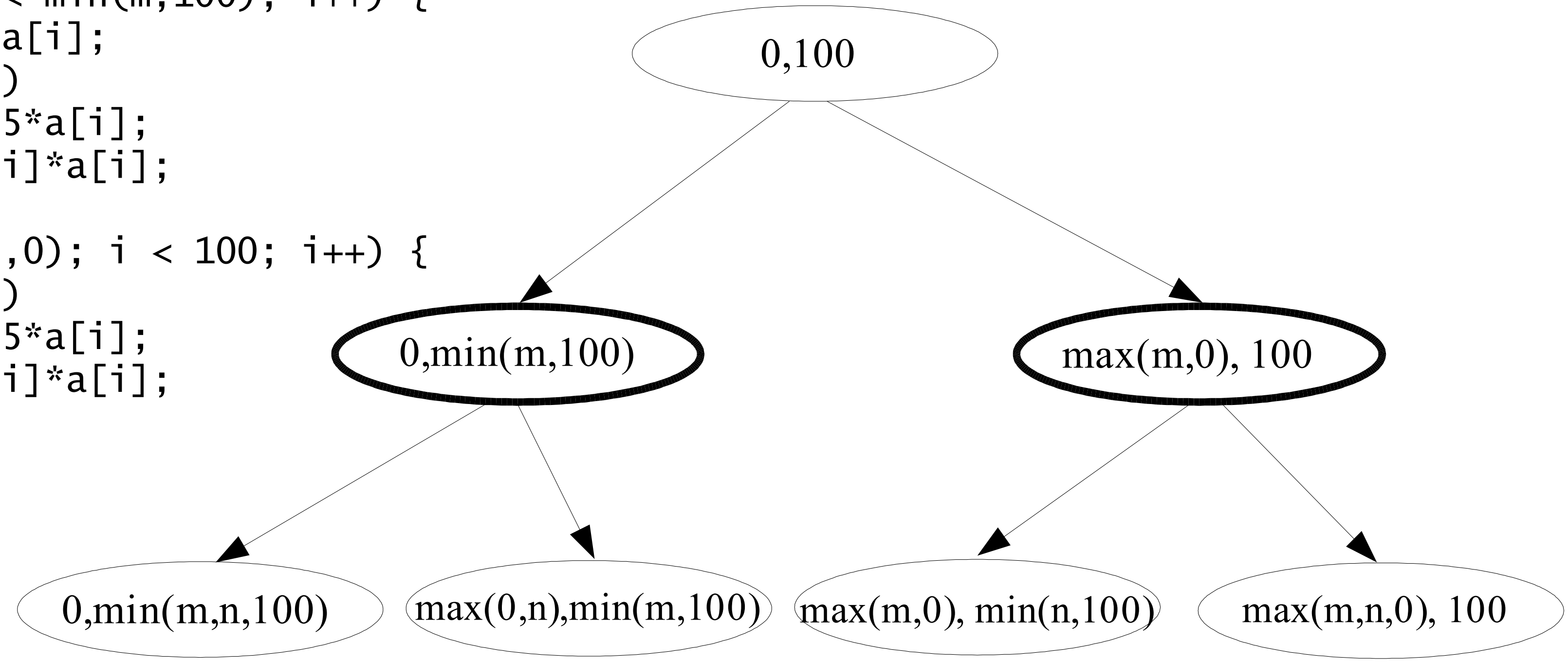
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Loops with multiple split points

- The second level of the tree corresponds to the two loops created to remove the (i < m) inductive-branch.

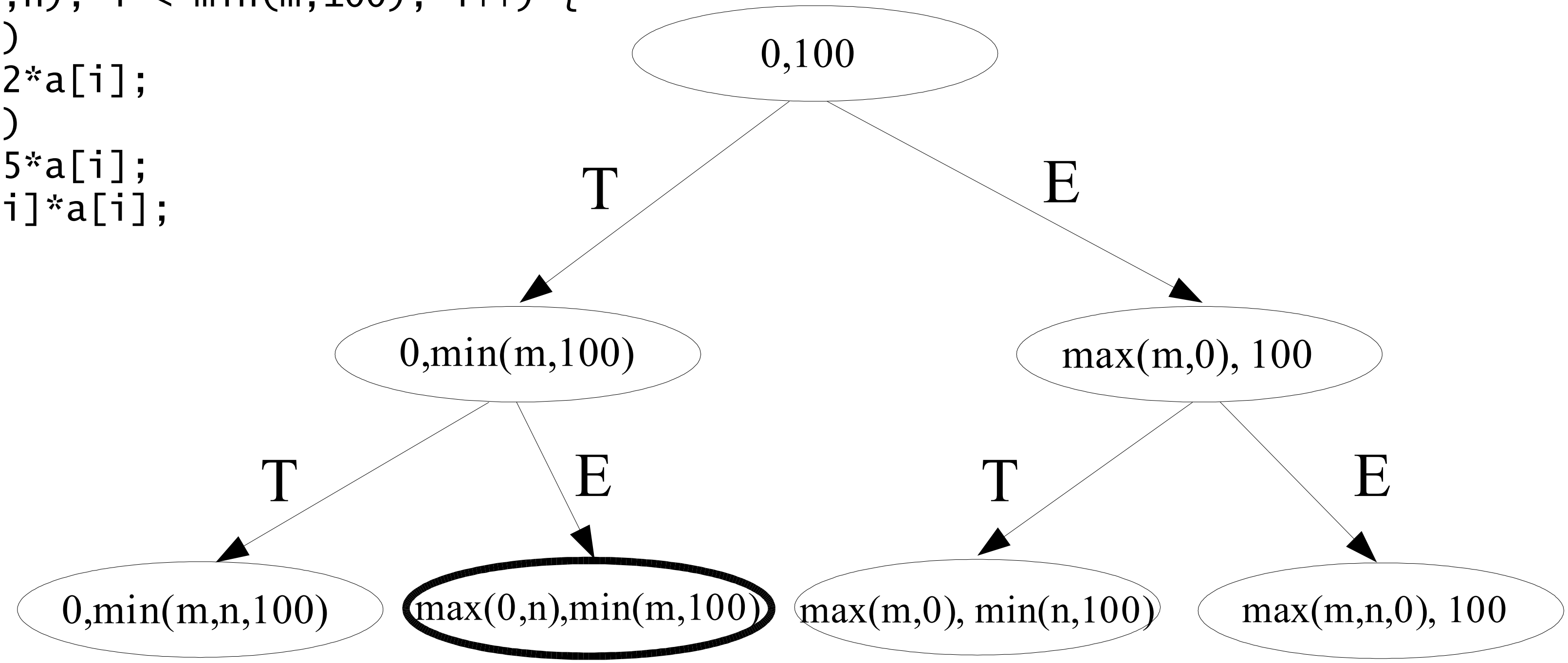
```
for (i=0; i < min(m,100); i++) {
   a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,0); i < 100; i++) {
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Removing “dead” inductive-branches

- To easily remove “dead” inductive-branches, we mark the edges in with Then / Else to correspond with the condition:

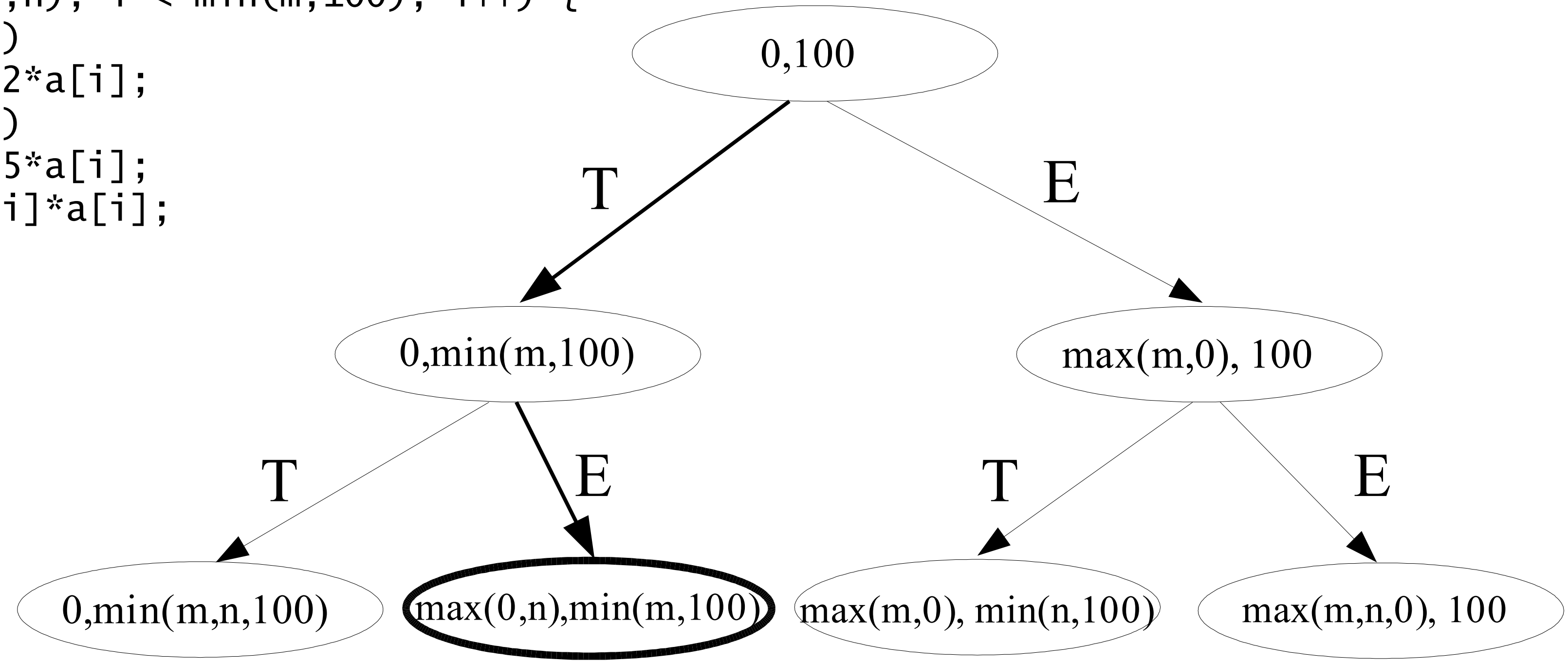
```
for (i=max(0,n); i < min(m,100); i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Removing “dead” inductive-branches

- We examine the path that leads from the root of the sub-range tree to the desired sub-range node.

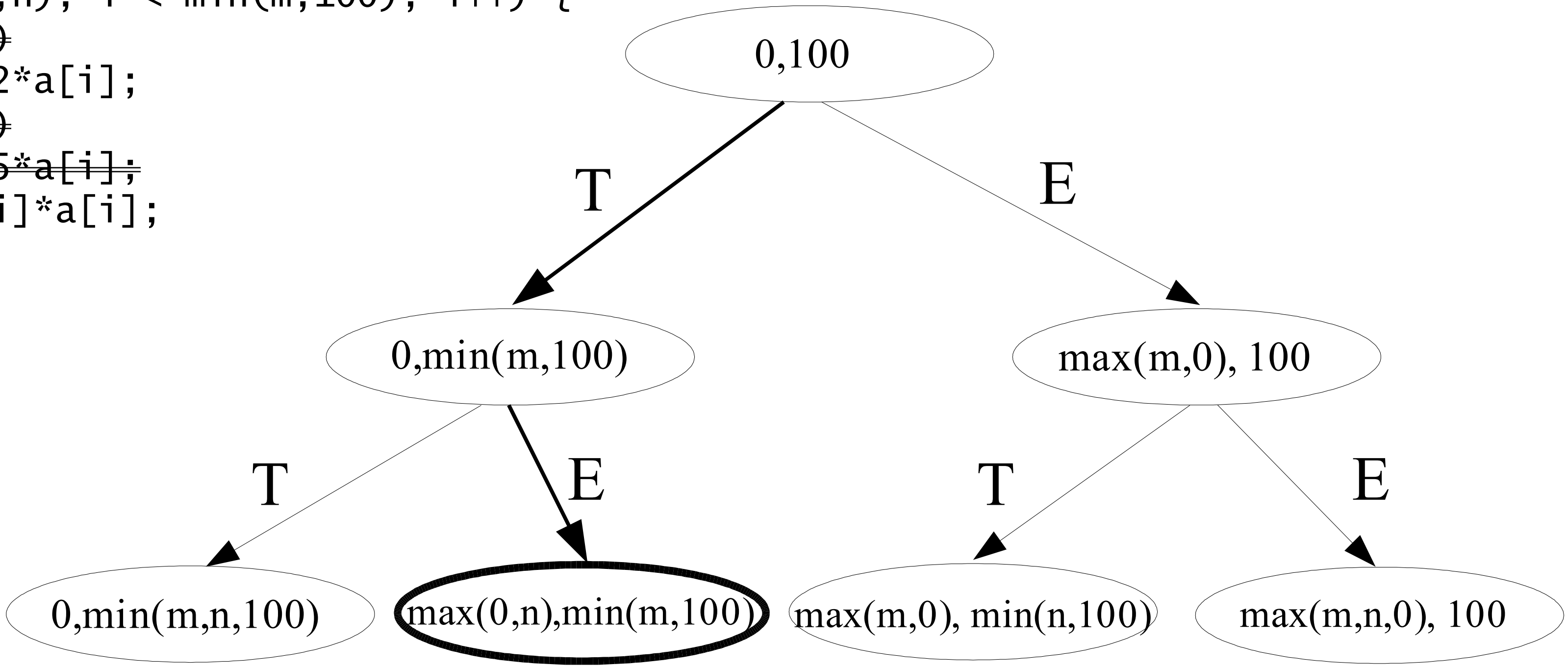
```
for (i=max(0,n); i < min(m,100); i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Removing "dead" inductive-branches

- We examine the path that leads from the root of the sub-range tree to the desired sub-range node.
- And, eliminate branches accordingly

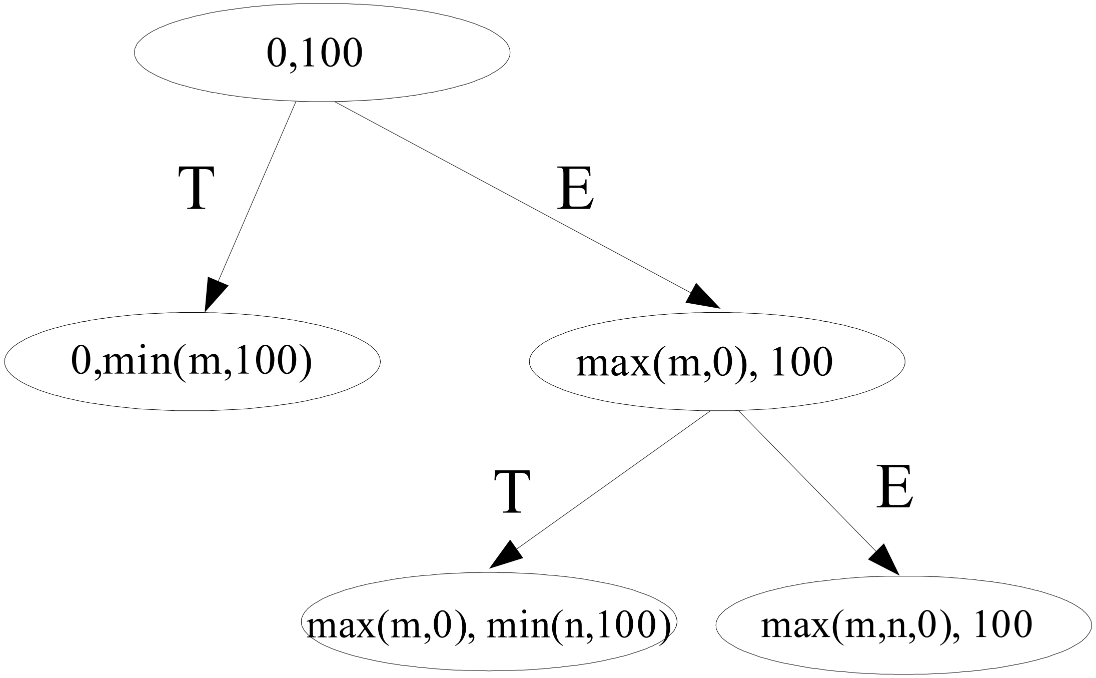
```
for (i=max(0,n); i < min(m,100); i++) {
   ~~if (i < m)~~
     a[i] = 2*a[i];
   ~~if (i < n)~~
     ~~a[i] = 5*a[i];~~
   b[i] = a[i]*a[i];
}
```



Nested inductive-branches

- Nested branches only split the nodes of the ranges to which they apply.

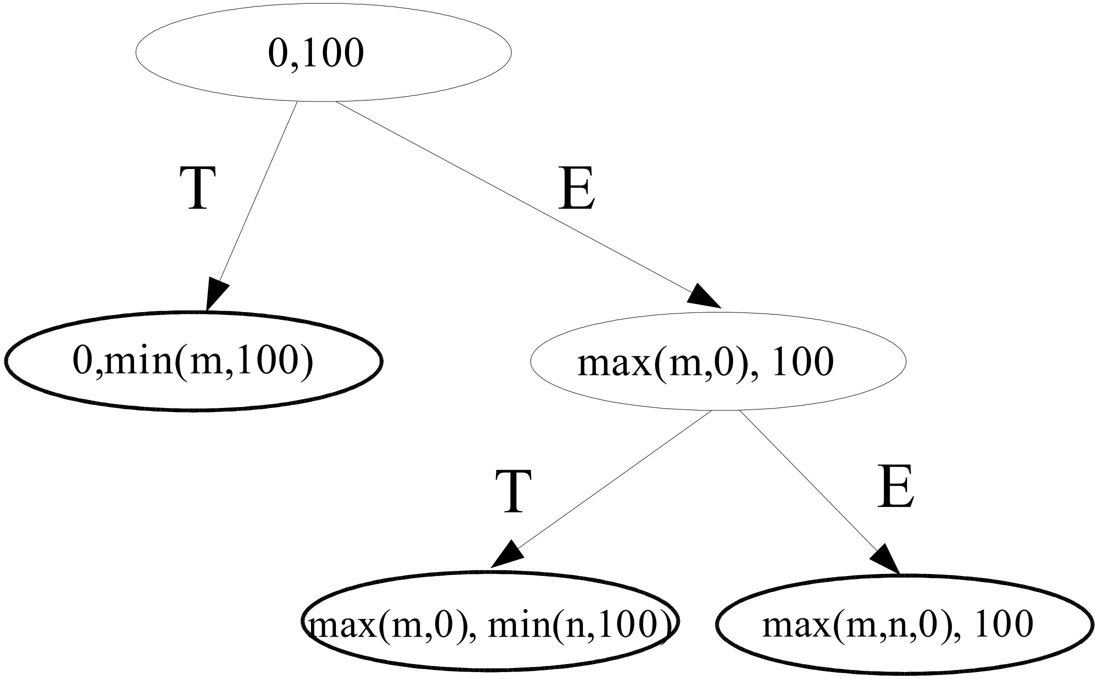
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   else
     if (i < n)
       a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Nested inductive-branches

- Nested branches only split the nodes of the ranges to which they apply.
- The result will be three loops:

```
for (i=0; i < min(m,100); i++) {
   a[i] = 2*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,0); i<min(n,100); i++)
{
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,n,0); i<100; i++) {
   b[i] = a[i]*a[i];
}
```



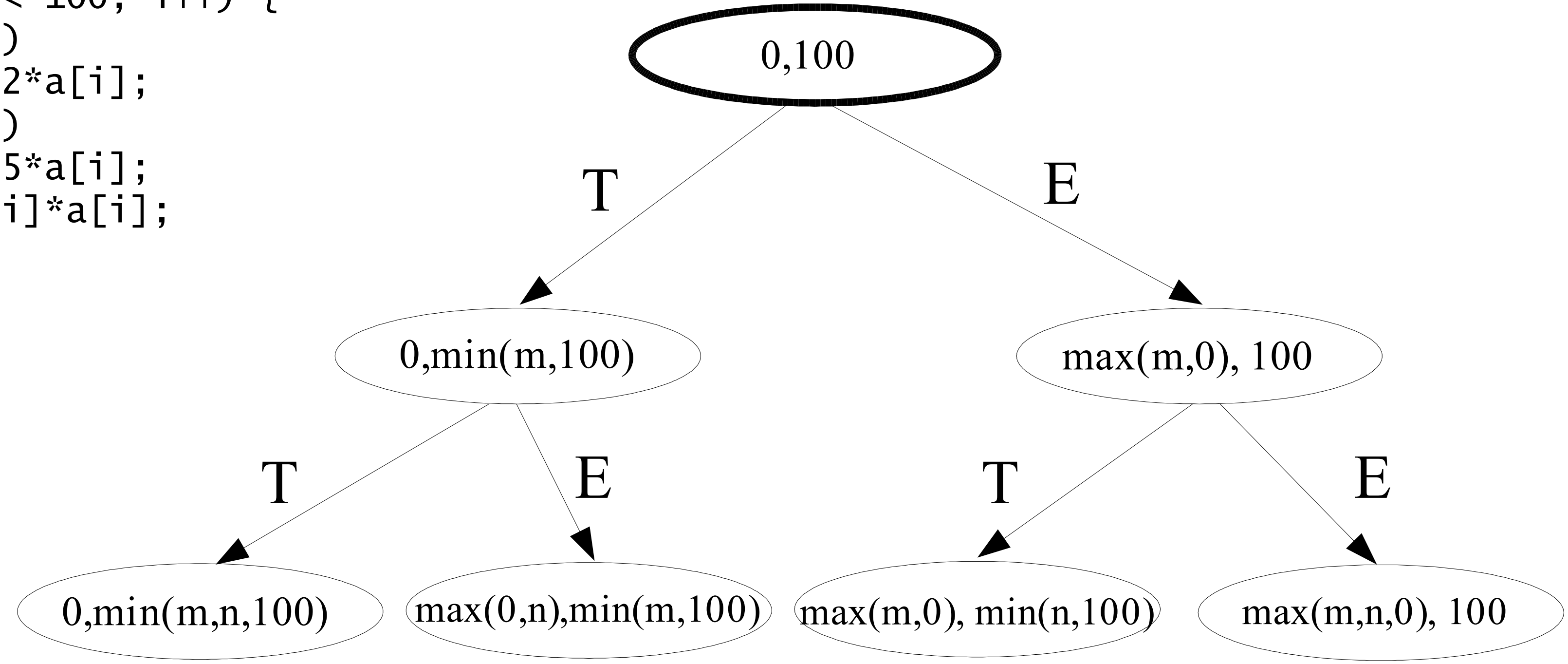
Controlling code growth

- We mark the root of the sub-range tree with the code size estimate of the original loop
- For every sub-node, we compute the code size estimate by subtracting the removed section's size estimate.
- We end up with a tree where all the nodes, which represent sub-ranges of loops, have an associated code size estimates for their corresponding loops.

Controlling code growth

- We descend down the sub-range tree using a greedy bredth-first like approach:

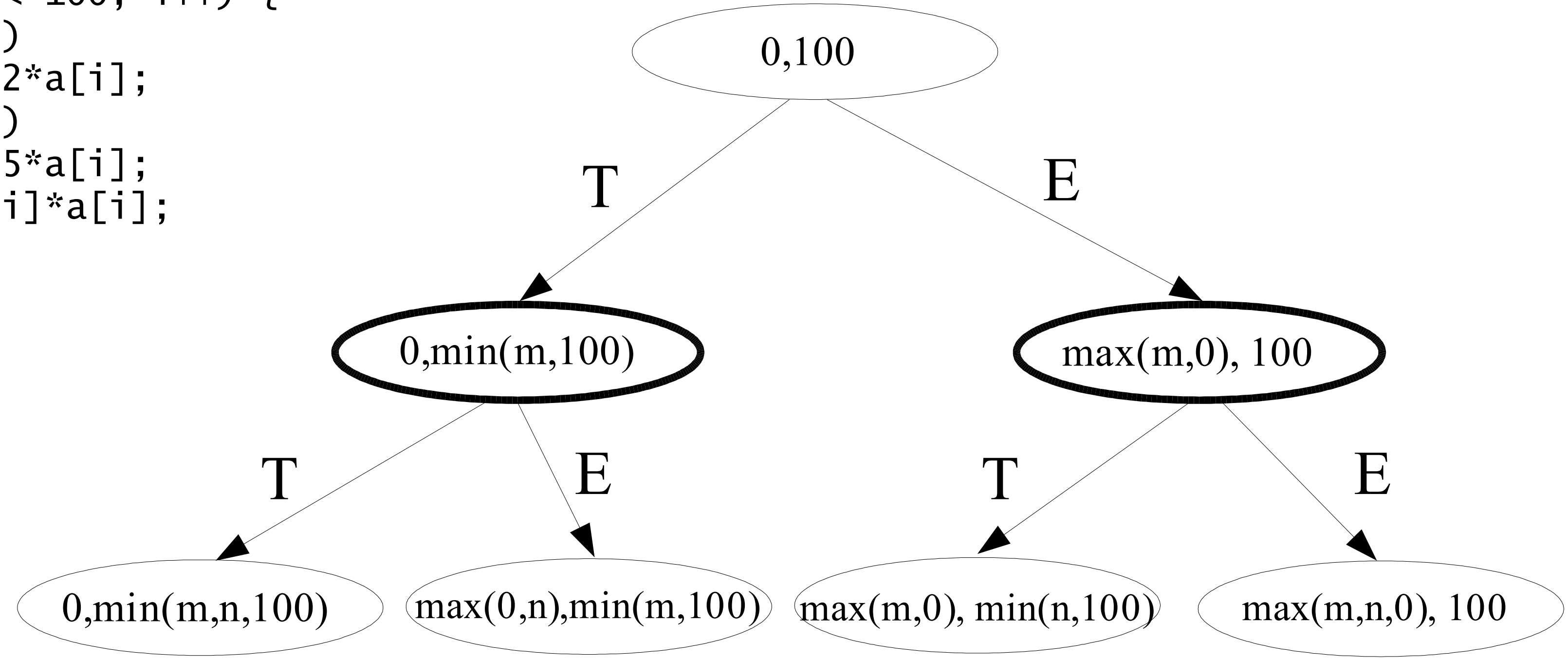
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Controlling code growth

- We descend down the sub-range tree using a greedy bredth-first like approach:

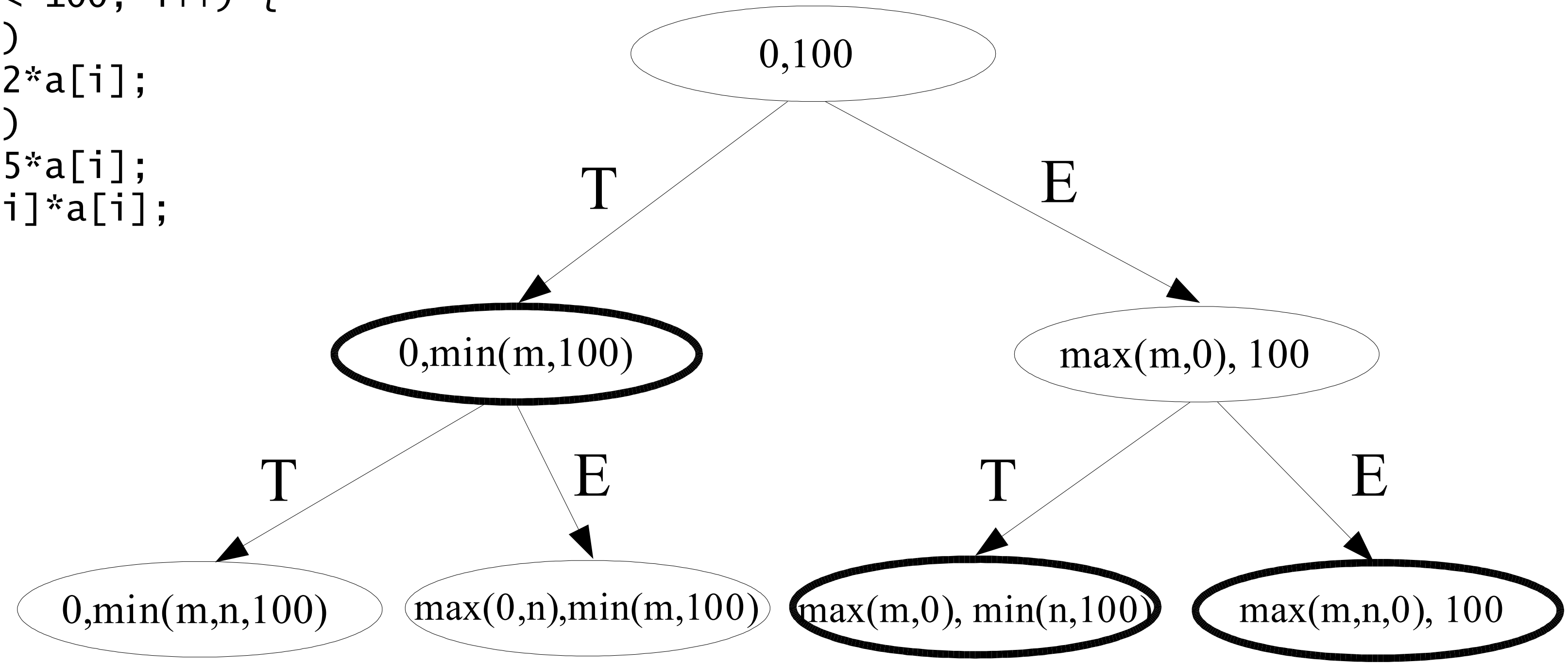
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Controlling code growth

- We descend down the sub-range tree using a greedy bredth-first like approach:

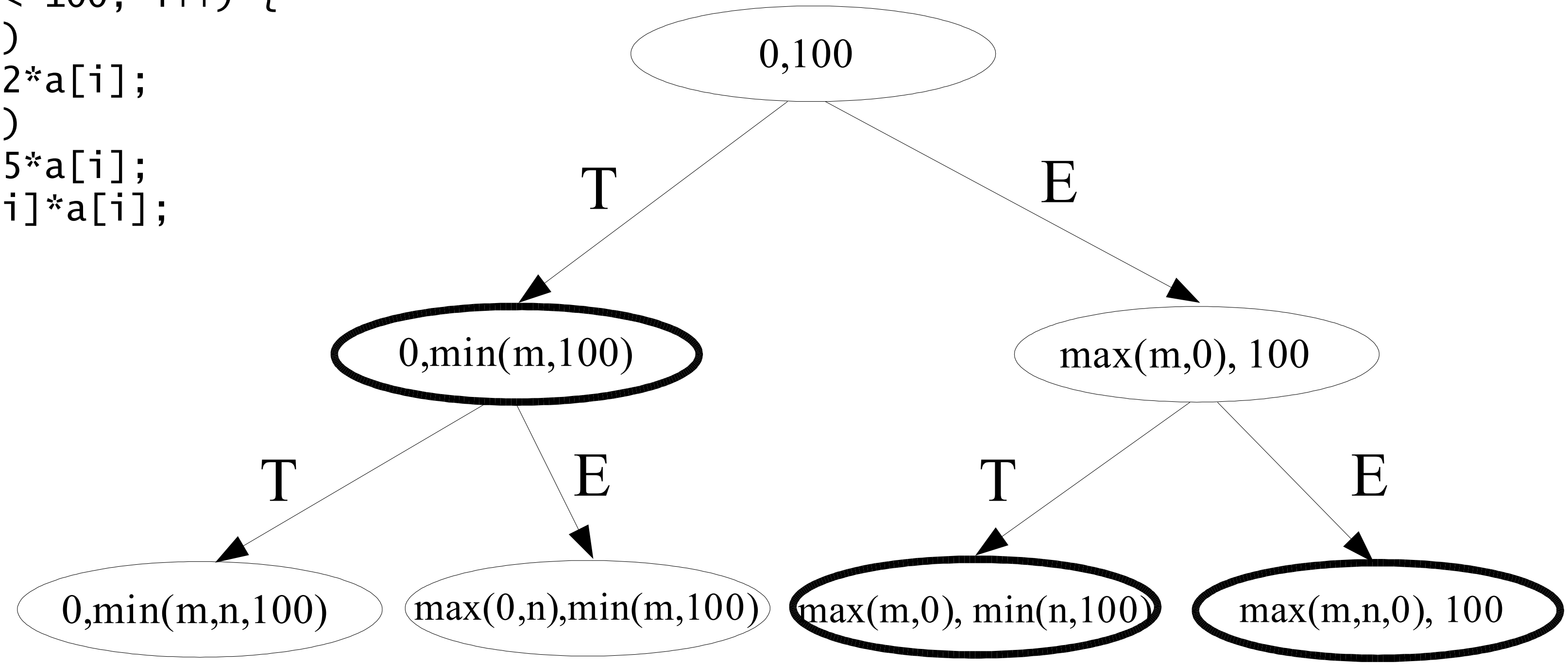
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Controlling code growth

- We stop when we reached all the leafs, or the code grwoth limit.

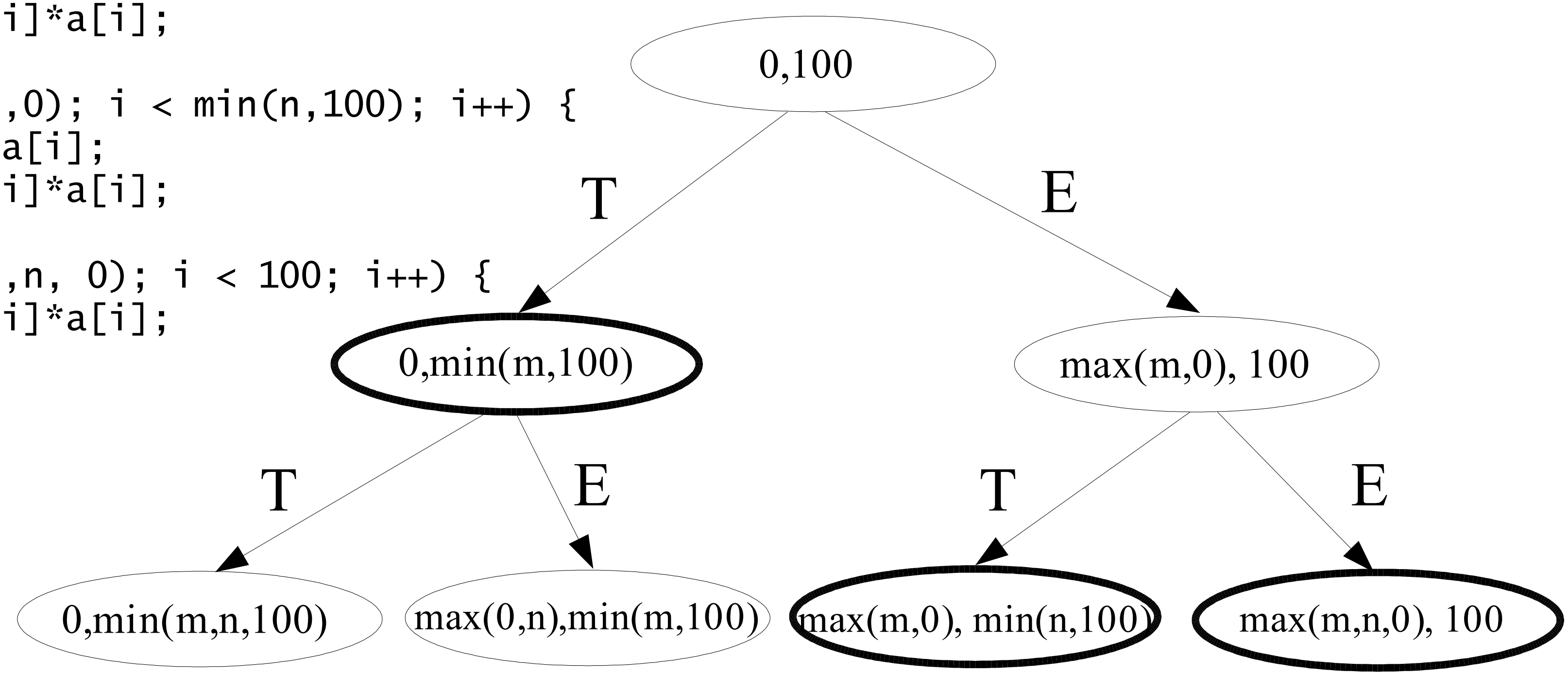
```
for (i=0; i < 100; i++) {
   if (i < m)
     a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
```



Controlling code growth

- The selected nodes designate the loops that will be generated.

```
for (i=0; i < min(m,100); i++) {
   a[i] = 2*a[i];
   if (i < n)
     a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,0); i < min(n,100); i++) {
   a[i] = 5*a[i];
   b[i] = a[i]*a[i];
}
for (i=max(m,n, 0); i < 100; i++) {
   b[i] = a[i]*a[i];
}
```



Generalized Index-Set Splitting

Q&A