Introduction to Program Analysis

Uday Khedker
(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

July 2018
Part 1

About These Slides
Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

  (Indian edition published by Ane Books in 2013)

Apart from the above book, some slides are based on the material from the following books


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.*
Motivating the Need of Program Analysis

- Some representative examples
  - Classical optimizations performed by compilers
  - Optimizing heap memory usage
- Course details, schedule, assessment policies etc.
- Program Model
- Soundness and Precision
Part 2

Classical Optimizations
Examples of Optimising Transformations (ALSU, 2006)

A C program and its optimizations

```c
void quicksort(int m, int n)
{
    int i, j, v, x;
    if (n <= m) return;
    i = m-1; j = n; v = a[n];        /* v is the pivot */
    while(1)                        /* Move values smaller */
    {
        do i = i + 1; while (a[i] < v); /* than v to the left of */
        do j = j - 1; while (a[j] > v); /* the split point (sp) */
        if (i >= j) break;             /* and other values */
        x = a[i]; a[i] = a[j]; a[j] = x; /* to the right of sp */
    }
    x = a[i]; a[i] = a[n]; a[n] = x; /* of the split point */
    quicksort(m,i); quicksort(i+1,n); /* Move the pivot to sp */
    /* sort the partitions to */
}                                      /* the left of sp and to the right of sp independently */
```
Intermediate Code

For the boxed source code

1. \( i = m - 1 \)
2. \( j = n \)
3. \( t1 = 4 \times n \)
4. \( t6 = a[t1] \)
5. \( v = t6 \)
6. \( i = i + 1 \)
7. \( t2 = 4 \times i \)
8. \( t3 = a[t2] \)
9. if \( t3 < v \) goto 6
10. \( j = j - 1 \)
11. \( t4 = 4 \times j \)
12. \( t5 = a[t4] \)
13. if \( t5 > v \) goto 10
14. if \( i \geq j \) goto 25
15. \( t2 = 4 \times i \)
16. \( t3 = a[t2] \)
17. \( x = t3 \)
18. \( t2 = 4 \times i \)
19. \( t4 = 4 \times j \)
20. \( t5 = a[t4] \)
21. \( a[t2] = t5 \)
22. \( t4 = 4 \times j \)
23. \( a[t4] = x \)
24. goto 6
25. \( t2 = 4 \times i \)
26. \( t3 = a[t2] \)
27. \( x = t3 \)
28. \( t2 = 4 \times i \)
29. \( t1 = 4 \times n \)
30. \( t6 = a[t1] \)
31. \( a[t2] = t6 \)
32. \( t1 = 4 \times n \)
33. \( a[t1] = x \)
Intermediate Code: Observations

- Multiple computations of expressions
- Simple control flow (conditional/unconditional goto)
  Yet undecipherable!
- Array address calculations
Understanding Control Flow

- Identify maximal sequences of linear control flow
  ⇒ Basic Blocks

- No transfer into or out of basic blocks except the first and last statements
  Control transfer into the block: only at the first statement.
  Control transfer out of the block: only at the last statement.
### Intermediate Code with Basic Blocks

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>i = m - 1</td>
</tr>
<tr>
<td>2.</td>
<td>j = n</td>
</tr>
<tr>
<td>3.</td>
<td>( t1 = 4 \times n )</td>
</tr>
<tr>
<td>4.</td>
<td>t6 = a[t1]</td>
</tr>
<tr>
<td>5.</td>
<td>v = t6</td>
</tr>
<tr>
<td>6.</td>
<td>i = i + 1</td>
</tr>
<tr>
<td>7.</td>
<td>t2 = 4 \times i</td>
</tr>
<tr>
<td>8.</td>
<td>t3 = a[t2]</td>
</tr>
<tr>
<td>9.</td>
<td>if t3 &lt; v goto 6</td>
</tr>
<tr>
<td>10.</td>
<td>j = j - 1</td>
</tr>
<tr>
<td>11.</td>
<td>t4 = 4 \times j</td>
</tr>
<tr>
<td>12.</td>
<td>t5 = a[t4]</td>
</tr>
<tr>
<td>13.</td>
<td>if t5 &gt; v goto 10</td>
</tr>
<tr>
<td>14.</td>
<td>if i &gt;= j goto 25</td>
</tr>
<tr>
<td>15.</td>
<td>t2 = 4 \times i</td>
</tr>
<tr>
<td>16.</td>
<td>t3 = a[t2]</td>
</tr>
<tr>
<td>17.</td>
<td>x = t3</td>
</tr>
<tr>
<td>18.</td>
<td>t2 = 4 \times i</td>
</tr>
<tr>
<td>19.</td>
<td>t4 = 4 \times j</td>
</tr>
<tr>
<td>20.</td>
<td>t5 = a[t4]</td>
</tr>
<tr>
<td>21.</td>
<td>a[t2] = t5</td>
</tr>
<tr>
<td>22.</td>
<td>t4 = 4 \times j</td>
</tr>
<tr>
<td>23.</td>
<td>a[t4] = x</td>
</tr>
<tr>
<td>24.</td>
<td>goto 6</td>
</tr>
<tr>
<td>25.</td>
<td>t2 = 4 \times i</td>
</tr>
<tr>
<td>26.</td>
<td>t3 = a[t2]</td>
</tr>
<tr>
<td>27.</td>
<td>x = t3</td>
</tr>
<tr>
<td>28.</td>
<td>t2 = 4 \times i</td>
</tr>
<tr>
<td>29.</td>
<td>t1 = 4 \times n</td>
</tr>
<tr>
<td>30.</td>
<td>t6 = a[t1]</td>
</tr>
<tr>
<td>31.</td>
<td>a[t2] = t6</td>
</tr>
<tr>
<td>32.</td>
<td>t1 = 4 \times n</td>
</tr>
<tr>
<td>33.</td>
<td>a[t1] = x</td>
</tr>
</tbody>
</table>
Program Flow Graph

\begin{align*}
B1 & : i &= m - 1 \\
& & j &= n \\
& & t1 &= 4 \times n \\
& & t6 &= a[t1] \\
& & v &= t6 \\
B2 & : i &= i + 1 \\
& & t2 &= 4 \times i \\
& & t3 &= a[t2] \\
& & \text{if } t3 < v \text{ goto } B2 \\
B3 & : j &= j - 1 \\
& & t4 &= 4 \times j \\
& & t5 &= a[t4] \\
& & \text{if } t5 > v \text{ goto } B3 \\
B4 & : \text{if } i \geq j \text{ goto } B6 \\
& & t1 &= 4 \times n \\
& & t6 &= a[t1] \\
& & a[t2] &= t6 \\
& & a[t2] &= t6 \\
& & t1 &= 4 \times n \\
& & a[t1] &= x \\
B5 & : t2 &= 4 \times i \\
& & t3 &= a[t2] \\
& & x &= t3 \\
& & t2 &= 4 \times i \\
& & t4 &= 4 \times j \\
& & t5 &= a[t4] \\
& & a[t2] &= t5 \\
& & t4 &= 4 \times j \\
& & a[t4] &= x \\
& & \text{goto } B2 \\
\end{align*}
## Program Flow Graph: Observations

<table>
<thead>
<tr>
<th>Nesting Level</th>
<th>Basic Blocks</th>
<th>No. of Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B1, B6</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>B4, B5</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>B2, B3</td>
<td>8</td>
</tr>
</tbody>
</table>
Local Common Subexpression Elimination

B1
i = m - 1
j = n
t1 = 4 * n
t6 = a[t1]
v = t6

B2
i = i + 1
t2 = 4 * i
t3 = a[t2]
if t3 < v goto B2

B3
j = j - 1
t4 = 4 * j
t5 = a[t4]
if t5 > v goto B3

B4
if i >= j goto B6

B5
t2 = 4 * i
t3 = a[t2]
x = t3
t2 = 4 * i
t4 = 4 * j
t5 = a[t4]
a[t2] = t5
t4 = 4 * j
a[t4] = x
goto B2

B6
t2 = 4 * i
t3 = a[t2]
x = t3
t2 = 4 * i
t1 = 4 * n
t6 = a[t1]
a[t2] = t6
t1 = 4 * n
a[t1] = x
Local Common Subexpression Elimination

B1

\begin{align*}
i & = m - 1 \\
j & = n \\
t1 & = 4 \times n \\
t6 & = a[t1] \\
v & = t6
\end{align*}

B2

\begin{align*}
i & = i + 1 \\
t2 & = 4 \times i \\
t3 & = a[t2] \\
\text{if } t3 & < v \text{ goto B2}
\end{align*}

B3

\begin{align*}
\text{if } i & > j \text{ goto B6} \\
j & = j - 1 \\
t4 & = 4 \times j \\
t5 & = a[t4] \\
\text{if } t5 & > v \text{ goto B3}
\end{align*}

B4

\begin{align*}
\text{if } i & \geq j \text{ goto B6} \\
t2 & = 4 \times i \\
t3 & = a[t2] \\
x & = t3 \\
a[t2] & = t6 \\
a[t1] & = x
\end{align*}

B5

\begin{align*}
t2 & = 4 \times i \\
t3 & = a[t2] \\
x & = t3 \\
t4 & = 4 \times j \\
t5 & = a[t4] \\
a[t2] & = t5 \\
a[t4] & = x \\
goto & \text{ B2}
\end{align*}
Global Common Subexpression Elimination

```plaintext
B1
i = m - 1
j = n

B2
i = i + 1

B3
j = j - 1

B4
if i >= j goto B6

B5
if t5 > v goto B3

B6
t1 = 4 * n
t2 = 4 * i
t3 = a[t2]
x = t3
t4 = 4 * j
t5 = a[t4]

B7
v = t6

B8
a[t1] = x

B9
t2 = 4 * i
t3 = a[t2]

B10
if t3 < v goto B2

B11
B6

t1 = 4 * n
t6 = a[t1]

B12
v = t6

B13
a[t2] = t6

B14
a[t1] = x

B15
t2 = 4 * i

t3 = a[t2]

B16
if t3 < v goto B2
```
Global Common Subexpression Elimination

\[ i = m - 1 \]
\[ j = n \]
\[ t1 = 4 \times n \]
\[ v = t6 \]
\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ \text{if } t3 < v \text{ goto } B2 \]
\[ j = j - 1 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ \text{if } t5 > v \text{ goto } B3 \]
\[ \text{if } i \geq j \text{ goto } B6 \]
\[ a[t1] = x \]

\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ x = t3 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ a[t2] = t5 \]
\[ a[t4] = x \]
\[ \text{goto } B2 \]

\[ t2 = 4 \times i \]
\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t2 = 4 \times i \]
Global Common Subexpression Elimination

\[ i = m - 1 \]
\[ j = n \]
\[ t1 = 4 \times n \]
\[ t6 = a[t1] \]
\[ v = t6 \]

\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ \text{if } t3 < v \text{ goto B2} \]

\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ \text{if } t3 < v \text{ goto B2} \]

\[ j = j - 1 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ \text{if } t5 > v \text{ goto B3} \]

\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ x = t3 \]

\[ j = j - 1 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ \text{if } t5 > v \text{ goto B3} \]

\[ \text{if } i \geq j \text{ goto B6} \]

\[ a[t1] = x \]

\[ a[t2] = t6 \]

\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ x = t3 \]

\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ a[t2] = t5 \]

\[ a[t4] = x \]
\[ \text{goto B2} \]
Global Common Subexpression Elimination

i = m - 1
j = n

B1

B6

t1 = 4 * n

B2

t6 = a[t1]

B3

j = j - 1

B4

if i >= j goto B6

if i = i + 1

B5

t2 = 4 * i

t3 = a[t2]

if t3 < v goto B2

a[t2] = t6

a[t2] = t6

v = t6

if t5 > v goto B3

i = i + 1

B2

t2 = 4 * i

t3 = a[t2]

if t3 < v goto B2

j = j - 1

B3

j = j - 1

B3

t4 = 4 * j

t5 = a[t4]

if t5 > v goto B3

a[t2] = t6

a[t2] = t6

x = t3

B5

t4 = 4 * j

t5 = a[t4]

a[t2] = t5

a[t4] = x

go to B2

a[t2] = t6

a[t1] = x
Global Common Subexpression Elimination

B1

\[i = m - 1\]
\[j = n\]
\[t1 = 4 \times n\]
\[t6 = a[t1]\]
\[v = t6\]

B2

\[i = i + 1\]
\[t2 = 4 \times i\]
\[t3 = a[t2]\]
\[\text{if } t3 < v \text{ goto B2}\]

B3

\[j = j - 1\]
\[t4 = 4 \times j\]
\[t5 = a[t4]\]
\[\text{if } t5 > v \text{ goto B3}\]

B4

\[\text{if } i \geq j \text{ goto B6}\]

B5

\[x = t3\]
\[a[t2] = t5\]
\[a[t4] = x\]

B6

\[x = t3\]
\[t6 = a[t1]\]
\[a[t2] = t6\]
\[a[t1] = x\]

July 2018
Other Classical Optimizations

- Copy propagation
- Strength Reduction
- Elimination of Induction Variables
- Dead Code Elimination
Copy Propagation and Dead Code Elimination

B1

\[ i = m - 1 \]
\[ j = n \]
\[ t1 = 4 \times n \]
\[ t6 = a[t1] \]
\[ v = t6 \]

B2

\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
\[ \text{if } t3 < v \text{ goto B2} \]

B3

\[ j = j - 1 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
\[ \text{if } t5 > v \text{ goto B3} \]

B4

\[ \text{if } i \geq j \text{ goto B6} \]

B5

\[ x = t3 \]
\[ a[t2] = t5 \]
\[ a[t4] = x \]
\[ \text{goto B2} \]

B6

\[ x = t3 \]
\[ t6 = a[t1] \]
\[ a[t2] = t6 \]
\[ a[t1] = x \]
Copy Propagation and Dead Code Elimination

B1

\[ \begin{align*}
    i &= m - 1 \\
    j &= n \\
    t1 &= 4 \times n \\
    t6 &= a[t1] \\
    v &= t6
\end{align*} \]

B2

\[ \begin{align*}
    i &= i + 1 \\
    t2 &= 4 \times i \\
    t3 &= a[t2] \\
    \text{if } t3 < v \text{ goto B2}
\end{align*} \]

B3

\[ \begin{align*}
    j &= j - 1 \\
    t4 &= 4 \times j \\
    t5 &= a[t4] \\
    \text{if } t5 > v \text{ goto B3}
\end{align*} \]

B4

\[ \begin{align*}
    \text{if } i \geq j \text{ goto B6}
\end{align*} \]

B5

\[ \begin{align*}
    x &= t3 \\
    a[t2] &= t5 \\
    a[t4] &= t3 \\
    \text{goto B2}
\end{align*} \]
Copy Propagation and Dead Code Elimination

B1
\[
\begin{align*}
i &= m - 1 \\
j &= n \\
t1 &= 4 \times n \\
t6 &= a[t1] \\
v &= t6
\end{align*}
\]

B2
\[
\begin{align*}
i &= i + 1 \\
t2 &= 4 \times i \\
t3 &= a[t2] \\
\text{if } t3 < v \text{ goto B2}
\end{align*}
\]

B3
\[
\begin{align*}
j &= j - 1 \\
t4 &= 4 \times j \\
t5 &= a[t4] \\
\text{if } t5 > v \text{ goto B3}
\end{align*}
\]

B4
\[
\begin{align*}
\text{if } i \geq j \text{ goto B6}
\end{align*}
\]

B6
\[
\begin{align*}
t6 &= a[t1] \\
a[t2] &= t6 \\
a[t1] &= t3
\end{align*}
\]

B5
\[
\begin{align*}
a[t2] &= t5 \\
a[t4] &= t3 \\
goto B2
\end{align*}
\]
Strength Reduction and Induction Variable Elimination

B1
\[ i = m - 1 \]
\[ j = n \]
\[ t1 = 4 \times n \]
\[ t6 = a[t1] \]
\[ v = t6 \]

B2
\[ i = i + 1 \]
\[ t2 = 4 \times i \]
\[ t3 = a[t2] \]
if \( t3 < v \) goto B2

B3
\[ j = j - 1 \]
\[ t4 = 4 \times j \]
\[ t5 = a[t4] \]
if \( t5 > v \) goto B3

B4
if \( i \geq j \) goto B6

B5
\[ a[t2] = t5 \]
\[ a[t4] = t3 \]
goto B2

B6
\[ t6 = a[t1] \]
\[ a[t2] = t6 \]
\[ a[t1] = t3 \]
Strength Reduction and Induction Variable Elimination

B1

\[
\begin{align*}
i &= m - 1 \\
j &= n \\
t1 &= 4 \times n \\
t6 &= a[t1] \\
v &= t6 \\
t2 &= 4 \times i \\
t3 &= a[t2] \\
t4 &= 4 \times j
\end{align*}
\]

B2

\[
\begin{align*}
t2 &= t2 + 4 \\
t3 &= a[t2] \\
\text{if } t3 \text{ } < \text{ } v \text{ goto B2}
\end{align*}
\]

B3

\[
\begin{align*}
t4 &= t4 - 4 \\
t5 &= a[t4] \\
\text{if } t5 \text{ } > \text{ } v \text{ goto B3}
\end{align*}
\]

B4

\[
\begin{align*}
t2 &= t2 + 4 \\
t3 &= a[t2] \\
\text{if } t2 \text{ } \geq \text{ } t4 \text{ goto B6}
\end{align*}
\]

B5

\[
\begin{align*}
a[t2] &= t5 \\
a[t4] &= t3 \\
goto B2
\end{align*}
\]
Final Intermediate Code

B1

i = m - 1
j = n
t1 = 4 * n
t6 = a[t1]
v = t6
t2 = 4 * i
t4 = 4 * j

B2

t2 = t2 + 4
t3 = a[t2]
if t3 < v goto B2

B3

t4 = t4 - 4
t5 = a[t4]
if t5 > v goto B3

B4

if t2 >= t4 goto B6

B5

a[t2] = t5
a[t4] = t3
goto B2

B6

t6 = a[t1]
a[t2] = t6
a[t1] = t3
### Optimized Program Flow Graph

If we assume that a loop is executed 10 times, then the number of computations saved at run time

\[
= (14 - 10) + (11 - 4) \times 10 + (8 - 6) \times 10^2 = 4 + 70 + 200 = 274
\]
Observations

- Optimizations are transformations based on some information.
- Systematic analysis required for deriving the information.
- We have looked at data flow optimizations.
  Many control flow optimizations can also be performed.
## Categories of Optimizing Transformations and Analyses

<table>
<thead>
<tr>
<th>Category</th>
<th>Machine Independent/Dependent</th>
<th>Analysis Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Motion Redundancy Elimination Control flow Optimization</td>
<td>Machine Independent</td>
<td>Flow Analysis (Data + Control)</td>
</tr>
<tr>
<td>Loop Transformations</td>
<td>Machine Dependent</td>
<td>Dependence Analysis (Data + Control)</td>
</tr>
<tr>
<td>Instruction Scheduling Register Allocation Peephole Optimization</td>
<td>Machine Dependent</td>
<td>Several Independent Techniques</td>
</tr>
<tr>
<td>Vectorization Parallelization</td>
<td>Machine Dependent</td>
<td>Dependence Analysis (Data + Control)</td>
</tr>
</tbody>
</table>
What is Program Analysis?

Discovering information about a given program
What is Program Analysis?

Discovering information about a given program

- Representing the dynamic behaviour of the program
What is Program Analysis?

Discovering information about a given program

- Representing the dynamic behaviour of the program
- Most often obtained without executing the program
  - Static analysis Vs. Dynamic Analysis
  - Example of loop tiling for parallelization
What is Program Analysis?

Discovering information about a given program

- Representing the dynamic behaviour of the program
- Most often obtained without executing the program
  - Static analysis Vs. Dynamic Analysis
  - Example of loop tiling for parallelization
- Must represent all execution instances of the program
Why is it Useful?

- Code optimization
  - Improving time, space, energy, or power efficiency
  - Compilation for special architecture (e.g., multi-core)
Why is it Useful?

- **Code optimization**
  - Improving time, space, energy, or power efficiency
  - Compilation for special architecture (eg. multi-core)

- **Verification and validation**
  Giving guarantees such as: The program will
  - never divide a number by zero
  - never dereference a NULL pointer
  - close all opened files, all opened socket connections
  - not allow buffer overflow security violation
Why is it Useful?

- **Code optimization**
  - Improving time, space, energy, or power efficiency
  - Compilation for special architecture (e.g., multi-core)

- **Verification and validation**
  Giving guarantees such as:
  - The program will never divide a number by zero
  - Never dereference a NULL pointer
  - Close all opened files, all opened socket connections
  - Not allow buffer overflow security violation

- **Software engineering**
  - Maintenance, bug fixes, enhancements, migration
  - Example: Y2K problem
Why is it Useful?

• Code optimization
  ▶ Improving time, space, energy, or power efficiency
  ▶ Compilation for special architecture (e.g. multi-core)

• Verification and validation
  Giving guarantees such as: The program will
  ▶ never divide a number by zero
  ▶ never dereference a NULL pointer
  ▶ close all opened files, all opened socket connections
  ▶ not allow buffer overflow security violation

• Software engineering
  ▶ Maintenance, bug fixes, enhancements, migration
  ▶ Example: Y2K problem

• Reverse engineering
  To understand the program
Part 3

Optimizing Heap Memory Usage
Standard Memory Architecture of Programs

Heap allocation provides the flexibility of

- **Variable Sizes.** Data structures can grow or shrink as desired at runtime.
  (Not bound to the declarations in program.)

- **Variable Lifetimes.** Data structures can be created and destroyed as desired at runtime.
  (Not bound to the activations of procedures.)
Managing Heap Memory

Decision 1: When to Allocate?

- **Explicit.** Specified in the programs. (eg. Imperative/OO languages)
- **Implicit.** Decided by the language processors. (eg. Declarative Languages)
Managing Heap Memory

Decision 1: When to Allocate?

• **Explicit.** Specified in the programs. (eg. Imperative/OO languages)

• **Implicit.** Decided by the language processors. (eg. Declarative Languages)

Decision 2: When to Deallocate?

• **Explicit.** Manual Memory Management (eg. C/C++)

• **Implicit.** Automatic Memory Management aka Garbage Collection (eg. Java/Declarative languages)
State of Art in Manual Deallocation

- Memory leaks
  10% to 20% of last development effort goes in plugging leaks

- Tool assisted manual plugging
  
  *Purify, Electric Fence, RootCause, GlowCode, yakTest, Leak Tracer, BDW Garbage Collector, mtrace, memwatch, dmalloc etc.*

- All leak detectors
  - are dynamic (and hence specific to execution instances)
  - generate massive reports to be perused by programmers
  - usually do not locate last use but only allocation escaping a call

  ⇒ At which program point should a leak be “plugged”?
Garbage Collection ≡ Automatic Deallocation

- Retain active data structure.
  Deallocate inactive data structure.

- What is an Active Data Structure?
Garbage Collection ≡ Automatic Deallocation

- Retain active data structure.
  Deallocate inactive data structure.

- What is an Active Data Structure?

  If an object does not have an access path, (i.e. it is unreachable)
  then its memory can be reclaimed.
Garbage Collection ≡ AutomaticDeallocation

- Retain active data structure.
  Deallocate inactive data structure.

- What is an Active Data Structure?

  If an object does not have an access path, (i.e. it is unreachable)
  then its memory can be reclaimed.

What if an object has an access path, but is not accessed after the
given program point?
What is Garbage?

We use Java style statements for convenience
Read "x.lptr" as "x→lptr"

1. \( w = x \) // \( x \) points to \( ma \)
2. if (\( x.data < MAX \))
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class_of}_z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

Stack

Heap
What is Garbage?

The blue nodes will be used after statement 4

1. \( w = x \) \( // x \) points to \( m_a \)
2. if \( (x.data < MAX) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class of } z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
What is Garbage?

The blue nodes will be used after statement 4

1. \( w = x \)  // \( x \) points to \( m_a \)
2. if \((x.data < MAX)\)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New} \ \text{class}\_\text{of}\_z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
What is Garbage?

The blue nodes will be used after statement 4

1. \( w = x \)  // \( x \) points to \( m_a \)
2. if \( (x.data < \text{MAX}) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class of } z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

All white nodes are unused and should be considered garbage.
Is Reachable Same as Live?

From www.memorymanagement.org/glossary

live (also known as alive, active) : Memory(2) or an object is live if the program will read from it in future. *The term is often used more broadly to mean reachable.*

It is not possible, in general, for garbage collectors to determine exactly which objects are still live. Instead, they use some approximation to detect objects that are provably dead, *such as those that are not reachable.*

Similar terms: reachable. Opposites: dead. See also: undead.
Is Reachable Same as Live?

• Not really. Most of us know that.

Even with the state of art of garbage collection, 24% to 76% unused memory remains unclaimed

• The state of art compilers, virtual machines, garbage collectors cannot distinguish between the two
Reachability and Liveness

Comparison between different sets of objects:

Live ? Reachable ? Allocated
Reachability and Liveness

Comparison between different sets of objects:

$$ \text{Live} \subseteq \text{Reachable} \subseteq \text{Allocated} $$
Reachability and Liveness

Comparison between different sets of objects:

\[ \text{Live} \subseteq \text{Reachable} \subseteq \text{Allocated} \]

The objects that are not live must be reclaimed.
Reachability and Liveness

Comparison between different sets of objects:

$$\text{Live} \subseteq \text{Reachable} \subseteq \text{Allocated}$$

The objects that are not live must be reclaimed.

$$\neg \text{Live} \quad ? \quad \neg \text{Reachable} \quad ? \quad \neg \text{Allocated}$$
Reachability and Liveness

Comparison between different sets of objects:

Live $\subseteq$ Reachable $\subseteq$ Allocated

The objects that are not live must be reclaimed.

$\neg$ Live $\supseteq$ $\neg$ Reachable $\supseteq$ $\neg$ Allocated
Reachability and Liveness

Comparison between different sets of objects:

\[
\text{Live} \subseteq \text{Reachable} \subseteq \text{Allocated}
\]

The objects that are not live must be reclaimed.

\[
\neg \text{Live} \supseteq \neg \text{Reachable} \supseteq \neg \text{Allocated}
\]

Garbage collectors collect these
Cedar Mesa Folk Wisdom

Make the unused memory unreachable by setting references to NULL. (GC FAQ: http://www.iecc.com/gclist/GC-harder.html)
Cedar Mesa Folk Wisdom

Make the unused memory unreachable by setting references to NULL. (GC FAQ: http://www.iecc.com/gcfaq/GC-harder.html)
Cedar Mesa Folk Wisdom

- Most promising, simplest to understand, yet the hardest to implement.

- Which references should be set to NULL?
  - Most approaches rely on feedback from profiling.
  - No systematic and clean solution.
Distinguishing Between Reachable and Live

The state of art

- Eliminating objects reachable from root variables which are not live.
- Implemented in current Sun JVMs.
- Uses liveness data flow analysis of root variables (stack data).
- What about liveness of heap data?
Liveness of Stack Data: An Informal Introduction (1)

We use Java style statements for convenience
Read "x.lptr" as "x→lptr"

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while (\( x.data < MAX \))
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class of } z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

if changed to while
Liveness of Stack Data: An Informal Introduction (1)

1. \( w = x \) // \( x \) points to \( m_a \)
2. while \( (x.data < \text{MAX}) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class_of_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

What is the meaning of the use of data?
Liveness of Stack Data: An Informal Introduction (1)

1. \( w = x \) \hspace{1cm} // \hspace{1cm} x \text{ points to } m_a
2. while \( (x.data < \text{MAX}) \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New \ class\_of\_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

What is the meaning of the use of data?
Liveness of Stack Data: An Informal Introduction (1)

1. \( w = x \)     // \( x \) points to \( m_a \)
2. while \((x.data < MAX)\)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class_of_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

What is the meaning of the use of data?
Liveness of Stack Data: An Informal Introduction (1)

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while \((x.data < MAX)\)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class_of_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)

Accessing the location and reading its contents

Reading \( x \) (Stack data)

Heap

Stack
Liveness of Stack Data: An Informal Introduction (1)

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while (\( x.data < \text{MAX} \))
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class of } z \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
### Liveness of Stack Data: An Informal Introduction (1)

1. \( \text{w} = \text{x} \) // \( \text{x} \) points to \( m_a \)
2. \( \text{while} \ (\text{x.data} < \text{MAX}) \)
3. \( \quad \text{x} = \text{x.rptr} \)
4. \( \quad \text{y} = \text{x.lptr} \)
5. \( \quad \text{z} = \text{New \ class_of_z} \)
6. \( \quad \text{y} = \text{y.lptr} \)
7. \( \quad \text{z.sum} = \text{x.data} + \text{y.data} \)

---

**Diagram:**

- Stack: \( \text{w}, \text{x}, \text{y}, \text{z} \)
- Heap: Accessing the location and reading its contents

**Legend:**

- **lptr:** Location pointer
- **rptr:** Reference pointer
- **data:** Data value

---

**Reading x.rptr (Heap data):**

Accessing the location and reading its contents.
Liveness of Stack Data: An Informal Introduction (2)

\[
\begin{align*}
w &= x \\
\text{while (x.data < MAX)} & \\
x &= x.rptr \\
y &= x.lptr \\
z &= 	ext{New class of z} \\
y &= y.lptr \\
z.sum &= x.data + y.data
\end{align*}
\]

No variable is used beyond this program point
Liveness of Stack Data: An Informal Introduction (2)

```
while (x.data < MAX)
    x = x.rptr

y = x.lptr

z = New class_of_z

y = y.lptr

z.sum = x.data + y.data
```

Current values of x, y, and z are used beyond this program point.

Live

Dead
Liveness of Stack Data: An Informal Introduction (2)

- Current values of x, y, and z are used beyond this program point
- The value of y is different before and after the assignment to y
Liveness of Stack Data: An Informal Introduction (2)

- The current values of x and y are used beyond this program point
- The current value of z is not used beyond this program point
Liveness of Stack Data: An Informal Introduction (2)

The current values of x is used beyond this program point

Current values of y and z are not used beyond this program point

```
while (x.data < MAX)
    x = x.rptr
    y = x.lptr
    z = New class of z
    y = y.lptr
    z.sum = x.data + y.data
```

w = x

July 2018
Liveness of Stack Data: An Informal Introduction (2)

```
w = x

while (x.data < MAX)
    x = x.rptr

y = x.lptr

z = New class of z

y = y.lptr

z.sum = x.data + y.data
```

- Nothing is known as of now
- Some information will be available in the next iteration point
Liveness of Stack Data: An Informal Introduction (2)

- Current value of x is used beyond this program point
- However its value is different before and after the assignment
Liveness of Stack Data: An Informal Introduction (2)

- Current value of x is used beyond this program point
- There are two control flow paths beyond this program point

```
while (x.data < MAX)
    x = x.rptr

y = x.lptr

z = New_class_of_z

y = y.lptr

z.sum = x.data + y.data
```
Liveness of Stack Data: An Informal Introduction (2)

\[ w = x \]

while \((x.data < MAX)\)

\[ x = x.rptr \]

\[ y = x.lptr \]

\[ z = \text{New class of } z \]

\[ y = y.lptr \]

\[ z.sum = x.data + y.data \]

Current value of \(x\) is used beyond this program point.
Liveness of Stack Data: An Informal Introduction (2)

```
w = x

while (x.data < MAX)
    x = x.rptr

y = x.lptr

z = New class_of_z

y = y.lptr

z.sum = x.data + y.data
```

Current value of x is used beyond this program point
Liveness of Stack Data: An Informal Introduction (2)

\[ w = x \]

\[ \text{while} \ (x.\text{data} < \text{MAX}) \]

\[ x = x.\text{rptr} \]

\[ y = x.\text{lptr} \]

\[ z = \text{New class of } z \]

\[ y = y.\text{lptr} \]

\[ z.\text{sum} = x.\text{data} + y.\text{data} \]

End of iteration #1

Live

Dead

July 2018
Liveness of Stack Data: An Informal Introduction (2)

\[ w \leftarrow x \]
while \((x.data < \text{MAX})\)
\[ x \leftarrow x.rptr \]
\[ y \leftarrow x.lptr \]
\[ z = \text{New class-of-z} \]
\[ y \leftarrow y.lptr \]
\[ z.sum = x.data + y.data \]

End of iteration #2

Live

Dead
Applying Cedar Mesa Folk Wisdom to Heap Data

Liveness Analysis of Heap Data
If the `while` loop is not executed even once.

1. \(w = x\)  // \(x\) points to \(m\)
2. while \((x.data < MAX)\)
3. \(x = x.rptr\)
4. \(y = x.lptr\)
5. \(z = \text{New class of } z\)
6. \(y = y.lptr\)
7. \(z.sum = x.data + y.data\)
Applying Cedar Mesa Folk Wisdom to Heap Data

Liveness Analysis of Heap Data
If the while loop is executed once.

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while \( x.data < \text{MAX} \)
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class_of_z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
Applying Cedar Mesa Folk Wisdom to Heap Data

Liveness Analysis of Heap Data
If the while loop is executed twice.

1. \( w = x \)  // \( x \) points to \( m_a \)
2. while (\( x.data < \text{MAX} \))
3. \( x = x.rptr \)
4. \( y = x.lptr \)
5. \( z = \text{New class}_{\text{of } z} \)
6. \( y = y.lptr \)
7. \( z.sum = x.data + y.data \)
The Moral of the Story

- Mappings between access expressions and l-values keep changing

- This is a *rule* for heap data
  For stack and static data, it is an *exception*!

- Static analysis of programs has made significant progress for stack and static data.

What about heap data?

- Given two access expressions at a program point, do they have the same l-value?
- Given the same access expression at two program points, does it have the same l-value?
Our Solution (1)

1. \( w = x \)
2. \( w = null \)
3. \( \text{while } (x.\text{data} < \text{MAX}) \{
   x.\text{lptr} = null
   x = x.\text{rptr}
\} \)
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class of } z \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
8. \( \text{return } z.\text{sum} \)

July 2018
Our Solution (2)

1. \( y = z = \text{null} \)
2. \( w = x \)
3. \( w = \text{null} \)
4. \( \text{while} \ (x.\text{data} < \text{MAX}) \) {
   \( x.\text{lp} = \text{null} \)
   \( x = x.\text{rp} \)
   \( x.\text{rp} = x.\text{lp}.\text{rp} = \text{null} \)
   \( x.\text{lp}.\text{lp}.\text{lp} = \text{null} \)
   \( x.\text{lp}.\text{lp}.\text{rp} = \text{null} \)
5. \( y = x.\text{lp} \)
6. \( y = y.\text{lp} \)
7. \( y.\text{lp} = y.\text{rp} = \text{null} \)
8. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
9. \( x = y = \text{null} \)
10. \( \text{return} \ z.\text{sum} \)
11. \( z = \text{null} \)

**While loop is not executed even once**

Stack

Heap

- P
- C
- D
- E
- F
- G
- H
- J
- K
- L
- M
- N
- O

- A
- B
- I
- Q
- M
- N
- O

- X
- W
- Y
- Z

Note: The diagram illustrates the heap structure and the stack operations.
Our Solution (2)

```plaintext
y = z = null
1 w = x
   w = null
2 while (x.data < MAX)
   {  x.lptr = null
      x = x.rptr  }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class of z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
     x = y = null
8 return z.sum
   z = null
```

While loop is not executed even once
y = z = null

1 w = x
   w = null

2 while (x.data < MAX)
   { x.lptr = null
     x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

5 z = New class of z
   z.lptr = z.rptr = null

6 y = y.lptr
   y.lptr = y.rptr = null

7 z.sum = x.data + y.data
   x = y = null

8 return z.sum
   z = null

While loop is not executed even once
Our Solution (2)

```
y = z = null
1 w = x
w = null
2 while (x.data < MAX)
{    x.lptr = null
3     x = x.rptr
        x.rptr = x.lptr.rptr = null
        x.lptr.lptr.lptr = null
        x.lptr.lptr.rptr = null
    }
4 y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
5 z = New class of z
z.lptr = z.rptr = null
6 y = y.lptr
y.lptr = y.rptr = null
7 z.sum = x.data + y.data
x = y = null
8 return z.sum
z = null
```

While loop is not executed even once
Our Solution (2)

1. \( w = x \)
2. \( w = \text{null} \)
3. \( \text{while} \ (x.\text{data} < \text{MAX}) \{
   \quad \text{x.lptr} = \text{null}
   \quad x = x.\text{rptr}
\} \)
4. \( \text{x.rptr} = x.\text{lptr}.\text{rptr} = \text{null} \)
5. \( x.\text{lptr}.\text{lptr}.\text{lptr} = \text{null} \)
6. \( x.\text{lptr}.\text{lptr}.\text{rptr} = \text{null} \)
7. \( y = x.\text{lptr} \)
8. \( \text{x.lptr} = y.\text{rptr} = \text{null} \)
9. \( y.\text{lptr}.\text{lptr} = y.\text{lptr}.\text{rptr} = \text{null} \)
10. \( z = \text{New class of } z \)
11. \( z.\text{lptr} = z.\text{rptr} = \text{null} \)
12. \( y = y.\text{lptr} \)
13. \( y.\text{lptr} = y.\text{rptr} = \text{null} \)
14. \( z.\text{sum} = x.\text{data} + y.\text{data} \)
15. \( x = y = \text{null} \)
16. \( \text{return } z.\text{sum} \)
17. \( z = \text{null} \)

While loop is not executed even once
Our Solution (2)

y = z = null

1  w = x
   w = null

2  while (x.data < MAX)
   {
      x.lptr = null
      x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null

3  y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null

4  z = New class_of_z
   z.lptr = z.rptr = null

5  y = y.lptr
   y.lptr = y.rptr = null

6  z.sum = x.data + y.data
   x = y = null

7  return z.sum
   z = null

While loop is not executed even once
Our Solution (2)

1. \(w = x\)
2. \(w = \text{null}\)
3. while (\(x\).data < MAX)
   { \(x\).lptr = \text{null}\)
   \(x = x\).rptr
   \(x\).rptr = \(x\).lptr.rptr = \text{null}\)
   \(x\).lptr.lptr.lptr = \(x\).lptr.lptr.rptr = \text{null}\)
4. \(y = x\).lptr
   \(x\).lptr = \(y\).rptr = \text{null}\)
   \(y\).lptr.lptr = \(y\).lptr.rptr = \text{null}\)
5. \(z = \text{New class of } z\)
   \(z\).lptr = \(z\).rptr = \text{null}\)
6. \(y = y\).lptr
   \(y\).lptr = \(y\).rptr = \text{null}\)
7. \(z\).sum = \(x\).data + \(y\).data
8. return \(z\).sum

Plotted:

While loop is not executed even once
Our Solution (2)

1. w = x
2. w = null
3. while (x.data < MAX)
   4. x = x.rptr
   5. x.rptr = x.lptr.rptr = null
   6. x.lptr.rptr = x.lptr.lptr = null
4. y = x.lptr
   5. y.lptr = y.rptr = null
   6. y.rptr = y.lptr.rptr = null
5. z = New class of z
   7. z.lptr = z.rptr = null
   8. z.sum = x.data + y.data
6. x = y = null
7. return z.sum
   8. z = null

While loop is executed once
Our Solution (2)

```
y = z = null
w = x
w = null
while (x.data < MAX)
    { x.lptr = null
      x = x.rptr
    }
    x.rptr = x.lptr.rptr = null
    x.lptr.lptr.lptr = null
    x.lptr.lptr.rptr = null
y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
z = New class of z
z.lptr = z.rptr = null
y = y.lptr
y.lptr = y.rptr = null
z.sum = x.data + y.data
x = y = null
return z.sum
z = null
```

While loop is executed twice
Some Observations

```
y = z = null
1 w = x
   w = null
2 while (x.data < MAX)
   { x.lptr = null
      x = x.rptr
   }
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
3 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class_of_z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = null
8 return z.sum
   z = null
```
Some Observations

- The memory address that \( x \) holds when the execution reaches a given program point is not an invariant of program execution.
Some Observations

- The memory address that \( x \) holds when the execution reaches a given program point is not an invariant of program execution.
- Whether we dereference lptr out of \( x \) or rptr out of \( x \) at a given program point is an invariant of program execution.

```plaintext
y = z = null
w = x
w = null
while (x.data < MAX)
{
    x.lptr = null
    x = x.rptr
}  
x.rptr = x.lptr.rptr = null
x.lptr.lptr.lptr = null
x.lptr.lptr.rptr = null
y = x.lptr
x.lptr = y.rptr = null
y.lptr.lptr = y.lptr.rptr = null
z = New class_of_z
z.lptr = z.rptr = null
y = y.lptr
y.lptr = y.rptr = null
z.sum = x.data + y.data
x = y = null
return z.sum
z = null
```

July 2018
Some Observations

1. y = z = null
2. w = x
   w = null
3. while (x.data < MAX)
   { x.lptr = null
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
   x = x.rptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
4. y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5. z = New class of z
   z.lptr = z.rptr = null
6. y = y.lptr
   y.lptr = y.rptr = null
7. z.sum = x.data + y.data
8. x = y = null
9. return z.sum
   z = null

- The memory address that x holds when the execution reaches a given program point is not an invariant of program execution.
- Whether we dereference lptr out of x or rptr out of x at a given program point is an invariant of program execution.
- **A static analysis can discover only invariants**
Some Observations

The memory address that \( x \) holds when the execution reaches a given program point is not an invariant of program execution

Whether we dereference lptr out of \( x \) or rptr out of \( x \) at a given program point is an invariant of program execution

A static analysis can discover only some invariants

---

```
y = z = null
1 w = x
   w = null
2 while (x.data < MAX)
   {
      x.lptr = null
   }
3   x = x.rptr
   x.rptr = x.lptr.rptr = null
   x.lptr.lptr.lptr = null
   x.lptr.lptr.rptr = null
4 y = x.lptr
   x.lptr = y.rptr = null
   y.lptr.lptr = y.lptr.rptr = null
5 z = New class_of_z
   z.lptr = z.rptr = null
6 y = y.lptr
   y.lptr = y.rptr = null
7 z.sum = x.data + y.data
   x = y = null
8 return z.sum
   z = null
```
BTW, What is Static Analysis of Heap?

Static

Dynamic
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Dynamic

Program Code

Program Execution

July 2018 IIT Bombay
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Dynamic

Program Execution

Heap Memory

Heap Memory

Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Summary Heap Data

Dynamic

Program Execution

Heap Memory

?
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Summary Heap Data

Dynamic

Profiling

Program Execution

Heap Memory

Heap Memory

Heap Memory

Heap Memory

Heap Memory
BTW, What is Static Analysis of Heap?

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many

Static

Program Code

Static Analysis

Summary Heap Data

Dynamic

Program Execution

Heap Memory

Static Analysis of Heap

July 2018
Part 4

Course Details
The Main Theme of the Course

Constructing *suitable abstractions* for
*sound & precise modelling* of
*runtime behaviour* of programs
*efficiently*
The Main Theme of the Course

Constructing *suitable abstractions* for
*sound & precise modelling* of
*runtime behaviour* of programs
*efficiently*

Abstract, Bounded, Single Instance

Concrete, Unbounded, Infinitely Many
Sequence of Generalizations in the Course Modules

Bit Vector Frameworks
Sequence of Generalizations in the Course Modules

Bit Vector Frameworks

Theoretical abstractions
Sequence of Generalizations in the Course Modules

- Theoretical abstractions
- General frameworks
- Bit Vector Frameworks
Sequence of Generalizations in the Course Modules

- Theoretical abstractions
- General frameworks
- Bit Vector Frameworks
- Intraprocedural Level
Sequence of Generalizations in the Course Modules

- **Intraprocedural Level**
  - General frameworks
  - Bit Vector Frameworks

- **Interprocedural Level**
  - Theoretical abstractions
Course Pedagogy

• Interleaved lectures and tutorials
• Plenty of problem solving
• Practice problems will be provided,
  ▶ Ready-made solutions will not be provided
  ▶ Your solutions will be checked

• Detailed course plan can be found at the course page:
  http://www.cse.iitb.ac.in/~uday/courses/cs618-18/
• Moodle will be used extensively for announcements and discussions
Assessment Scheme

- Tentative plan

<table>
<thead>
<tr>
<th>Assessment</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mid Semester Examination</td>
<td>30%</td>
</tr>
<tr>
<td>End Semester Examination</td>
<td>45%</td>
</tr>
<tr>
<td>Two Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Project</td>
<td>15%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
</tr>
</tbody>
</table>

- Can be fine tuned based on the class feedback
Course Strength and Selection Criteria

- Less than 30 is preferable, 40 is tolerable
  At the moment no plan of restricting the registration

- Course primarily aimed at M.Tech. 1 students
  Follow up course and MTPs
Questions ??
Part 5

Program Model
Program Representation

- Three address code statements
  - Result, operator, operand1, operand2
  - Assignments, expressions, conditional jumps
  - Initially only scalars
    Pointers, structures, arrays modelled later

- Control flow graph representation
  - Nodes represent maximal groups of statements devoid of any control transfer except fall through
  - Edges represent control transfers across basic blocks
  - A unique Start node and a unique End node
    Every node reachable from Start, and End reachable from every node

- Initially only intraprocedural programs
  Function calls brought in later
int main()
{
    int a, b, c, n;

    a = 4;
    b = 2;
    c = 3;
    n = c*2;
    while (a <= n)
    {
        a = a+1;
    }
    if (a < 12)
    {
        a = a+b+c;
        return a;
    }
}
int main()
{
    int a, b, c, n;
    a = 4;
    b = 2;
    c = 3;
    n = c*2;
    while (a <= n)
    {
        a = a + 1;
    }
    if (a < 12)
        a = a + b + c;
    return a;
}
int main()
{
    int a, b, c, n;
    a = 4;
    b = 2;
    c = 3;
    n = c*2;
    while (a <= n)
    {
        a = a+1;
    }
    if (a < 12)
    {
        a = a+b+c;
    }
    return a;
}
Part 6

Requirements of Static Analysis
Important Requirements of Static Analysis

• We discuss the following important requirements
  ▶ Soundness
  ▶ Precision
  ▶ Efficiency
  ▶ Scalability

• Soundness and precision are described more formally later in module 2
Inexactness of Static Analysis Results

- Static analysis predicts run time behaviour of programs
- Static analysis is undecidable
  - there cannot exist an algorithm that can compute
  - exact result for every program
- Possible reasons of undecidability
  - Values of variables not known
  - Branch outcomes not known
  - Infinitely many paths in the presence of loops or recursion
  - Infinitely many values
- Static analysis predictions may not match the actual run time behaviour
Possible Errors in Static Analysis Predictions

- Some predictions may be erroneous because the predicted behaviour
  - may not be found in some execution instances, or
  - may not be found in any execution instance

  (Error ≡ Mismatch between run time behaviour and predicted behaviour)

- Some of these errors may be harmless whereas some may be harmful

- Some of these errors may be unavoidable (recall undecidability)

- How do we characterize, identify, and minimize, these errors?
Examples of Harmless and Harmful Errors in Predictions (1)

• For security check at an airport,
  ▶ Frisking a person more than others on mere suspicion may be an error but it is harmless from the view point of security
  ▶ Not frisking a person much even after a suspicion is an error and it could be a harmful from the view point of security

• For stopping smuggling of contraband goods
  ▶ Not checking every passenger may be erroneous but is harmless
  ▶ Checking every passenger may be right but is harmful

• Weather prediction during rainy season
  ▶ A doubtful prediction of “heavy to very heavy rain” is harmless
  ▶ Not predicting “heavy to very heavy rain” could be harmful
Examples of Harmless and Harmful Errors in Predictions (2)

- For medical diagnosis
  - Subjecting a person to further investigations may be erroneous but in most cases it is harmless
  - Avoiding further investigations even after some suspicions could be harmful

- For establishing justice in criminal courts
  - Starting with the assumption that an accused is innocent may be erroneous but is harmless
  - Starting with the assumption that an accused is guilty may be harmful
Harmless Errors and Harmful Errors in Static Analysis

• For a static analysis,
  ▶ Harmless errors can be tolerated but should be minimized (Precision)
  ▶ Harmful errors MUST be avoided (Soundness)

• Some behaviours concluded by a static analysis are
  ▶ uncertain and cannot be guaranteed to occur at run time,
    (This uncertainty is harmless and hence is conservative)
  ▶ certain and can be guaranteed to occur at run time
    (The absence of this certainty for these behaviours may be harmful)
Examples of Conservative and Definite Information

- **Liveness** is uncertain (also called **conservative**)

  If a variable is declared live at a program point, it may or may not be used beyond that program point at run time

  (Why is it harmless if the variable is not actually used?)

- **Deadness** (i.e. absence of liveness) is certain (also called **definite**)

  If a variable is declared to be dead at a program point, it is guaranteed to be not used beyond that program point at run time

  (Why is it harmful if the variable is not actually dead?)
The table below illustrates the outcomes of hypothesis testing:

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis holds</td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>
\{ \text{True, False} \} \times \{ \text{Positive, Negative} \}

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis does not hold</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hypothesis Accepted</strong></td>
<td><strong>Hypothesis Rejected</strong></td>
</tr>
<tr>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>
\[
\{ \text{True, False} \} \times \{ \text{Positive, Negative} \}
\]

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive</td>
<td></td>
<td>False Negative</td>
</tr>
<tr>
<td>False Positive</td>
<td></td>
<td>True Negative</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypothesis does not hold</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Positive</td>
<td></td>
<td>True Negative</td>
</tr>
</tbody>
</table>
\[ \{\text{True}, \text{False}\} \times \{\text{Positive}, \text{Negative}\} \]

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive</td>
<td>False Negative</td>
<td></td>
</tr>
<tr>
<td>False Positive</td>
<td>True Negative</td>
<td></td>
</tr>
</tbody>
</table>

No mismatch between prediction and reality
\[
\{ \text{True, False} \} \times \{ \text{Positive, Negative} \}
\]

<table>
<thead>
<tr>
<th></th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis holds</td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Mismatch between prediction and reality
The diagram illustrates the possible outcomes of a hypothesis test, which can be categorized into two dimensions:

- **True, False**
- **Positive, Negative**

The outcomes are as follows:

<table>
<thead>
<tr>
<th>Hypothesis Holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>True Positive</strong></td>
<td></td>
<td><strong>False Negative</strong></td>
</tr>
<tr>
<td><strong>False Positive</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **False Positive** indicates a mismatch between prediction and reality where the hypothesis is rejected but it should have been accepted.
- **True Positive** indicates a correct prediction where the hypothesis was accepted and it is true.
- **False Negative** indicates an error where the hypothesis was accepted but it should have been rejected.
- **True Negative** indicates a correct prediction where the hypothesis was rejected and it is false.

A **Harmless Error** is indicated for situations where the hypothesis is accepted and it is true, or rejected and it is false. However, there is a **Mismatch between prediction and reality** that needs to be addressed.
\( \{\text{True, False}\} \times \{\text{Positive, Negative}\} \)

<table>
<thead>
<tr>
<th></th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis holds</td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Mismatch between prediction and reality

- Harmless Error
- Harmful Error
\{\text{True, False}\} \times \{\text{Positive, Negative}\}

- **Acceptance** is a conservative decision based on uncertain information.
- **Rejection** is a definite decision based on certain information.

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive</td>
<td>False Negative</td>
<td></td>
</tr>
<tr>
<td>False Positive</td>
<td>True Negative</td>
<td></td>
</tr>
</tbody>
</table>

Mismatch between prediction and reality

- Harmless Error
- Harmful Error

- Acceptance is a conservative decision based on uncertain information.
- Rejection is a definite decision based on certain information.
Example of \( \{\text{True, False}\} \times \{\text{Positive, Negative}\} \)

Hypothesis: A patient IS suffering from Malaria

Hypothesis Accepted

Hypothesis Rejected
Example of \{True, False\} × \{Positive, Negative\}

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th></th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis holds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

July 2018
IIT Bombay
Example of \(\{\text{True, False}\} \times \{\text{Positive, Negative}\}\)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Blood test is advised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis Accepted</td>
<td>Test should be done</td>
</tr>
</tbody>
</table>

Hypothesis does not hold

Hypothesis Accepted

Hypothesis Rejected
Example of $\{\text{True, False}\} \times \{\text{Positive, Negative}\}$

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th></th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
</table>
| Hypothesis holds | Blood test is advised  
Test should be done | Blood test is not advised  
Test should be done |
| Hypothesis does not hold |                     |                     |
### Example of \( \{ \text{True, False} \} \times \{ \text{Positive, Negative} \} \)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Test should be done</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>Test need not be done</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

July 2018

IIT Bombay
Example of $\{\text{True, False}\} \times \{\text{Positive, Negative}\}$

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holds</td>
<td>Blood test is advised</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td></td>
<td>Test should be done</td>
<td>Test should be done</td>
</tr>
<tr>
<td>Does not hold</td>
<td>Blood test is advised</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td></td>
<td>Test need not be done</td>
<td>Test need not be done</td>
</tr>
</tbody>
</table>

July 2018

IIT Bombay
Example of \( \{ \text{True}, \text{False} \} \times \{ \text{Positive}, \text{Negative} \} \)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Blood test is not advised</td>
<td></td>
</tr>
<tr>
<td>Test should be done</td>
<td>Test should be done</td>
<td></td>
</tr>
<tr>
<td>True Positive</td>
<td>False Negative</td>
<td></td>
</tr>
</tbody>
</table>

| Hypothesis does not hold  | Blood test is advised |
|---------------------------| Test need not be done |
| False Positive            | True Negative        |
Example of \( \{ \text{True}, \text{False} \} \times \{ \text{Positive}, \text{Negative} \} \)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Test should be done</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>True Positive</td>
<td></td>
<td>False Negative</td>
</tr>
<tr>
<td>False Positive</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Harmless error
Example of \( \{ \text{True, False} \} \times \{ \text{Positive, Negative} \} \)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Test should be done</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>True Positive</td>
<td>False Negative</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypothesis does not hold</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Test need not be done</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>False Positive</td>
<td>True Negative</td>
<td></td>
</tr>
</tbody>
</table>

Harmless error

Harmful error
Example of $\{\text{True}, \text{False}\} \times \{\text{Positive}, \text{Negative}\}$

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holds</td>
<td>Blood test is advised Test should be done True Positive</td>
<td>Blood test is not advised Test should be done False Negative</td>
</tr>
<tr>
<td>Does not hold</td>
<td>Blood test is advised Test need not be done False Positive</td>
<td>Blood test is not advised Test need not be done True Negative</td>
</tr>
</tbody>
</table>

Harmless error

Sound

Harmful error
Example of $\{\text{True, False}\} \times \{\text{Positive, Negative}\}$

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td>Test should be done</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>Test need not be done</td>
<td>True Positive</td>
<td>Test should be done</td>
</tr>
<tr>
<td>False Positive</td>
<td></td>
<td>False Negative</td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td>Blood test is advised</td>
<td>Blood test is not advised</td>
</tr>
<tr>
<td>Test need not be done</td>
<td>False Positive</td>
<td>Test need not be done</td>
</tr>
<tr>
<td>True Negative</td>
<td></td>
<td>True Negative</td>
</tr>
</tbody>
</table>
Example of \( \{\text{True, False}\} \times \{\text{Positive, Negative}\} \)

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
<td></td>
</tr>
<tr>
<td>Test should be done</td>
<td></td>
</tr>
<tr>
<td>True Positive</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypothesis does not hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blood test is advised</td>
</tr>
<tr>
<td>Test need not be done</td>
</tr>
<tr>
<td>False Positive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Harmless error</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Precise</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Harmful error</th>
</tr>
</thead>
</table>

We talk about precision only for a sound analysis.

July 2018
Example of \{\text{True, False}\} \times \{\text{Positive, Negative}\}

Hypothesis: A patient IS suffering from Malaria

<table>
<thead>
<tr>
<th></th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis</td>
<td>Blood test is advised.</td>
<td>Blood test is not advised.</td>
</tr>
<tr>
<td>does not hold</td>
<td>Test should be done.</td>
<td>Test need not be done.</td>
</tr>
<tr>
<td></td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td></td>
<td>False Positive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Harmless error</td>
<td>Harmful error</td>
</tr>
</tbody>
</table>

We talk about precision only for a sound analysis

Precise

Imprecise

July 2018 IIT Bombay
The Role of How a Hypothesis is Framed (1)

- The following association critically depends on how a hypothesis is framed
  - False Positive $\equiv$ Imprecise
  - False Negative $\equiv$ Unsound

- In some cases, the hypothesis involves a negation

  For hearing a criminal case, a court begins with the hypothesis *The accused is NOT guilty*

  If a court chooses the non-negated hypothesis *An accused IS guilty* then
  - False Positive $\equiv$ Imprecise Unsound
  - False Negative $\equiv$ Unsound Imprecise
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis does not hold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accused is acquitted</td>
<td>Should be acquitted</td>
<td></td>
</tr>
</tbody>
</table>

July 2018 IIT Bombay
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accused is acquitted</td>
<td>Should be acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td>Should be acquitted</td>
<td></td>
<td>Should be acquitted</td>
</tr>
</tbody>
</table>

Hypothesis does not hold
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accused is acquitted</td>
<td>Should be acquitted</td>
<td></td>
</tr>
<tr>
<td>Accused is sentenced</td>
<td>Should be acquitted</td>
<td></td>
</tr>
</tbody>
</table>

| Hypothesis does not hold | |
|--------------------------| |
| Accused is acquitted | Should be sentenced |
| Accused is sentenced | Should be sentenced |
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>HypothesisRejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accused is acquitted</td>
<td>Should be acquitted</td>
<td>Accused is sentenced</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypothesis does not hold</th>
<th>Accused is acquitted</th>
<th>Should be sentenced</th>
<th>Accused is sentenced</th>
<th>Should be sentenced</th>
</tr>
</thead>
</table>
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis Holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accused is acquitted</td>
<td>Should be acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td>True Positive</td>
<td>False Negative</td>
<td>False Positive</td>
</tr>
</tbody>
</table>

Accused is acquitted

- Should be acquitted
- True Positive

Accused is sentenced

- Should be sentenced
- True Negative
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis holds</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accused is acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td></td>
<td>Should be acquitted</td>
<td>Should be acquitted</td>
</tr>
<tr>
<td></td>
<td><strong>True Positive</strong></td>
<td><strong>False Negative</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypothesis does not hold</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accused is acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td></td>
<td>Should be sentenced</td>
<td>Should be sentenced</td>
</tr>
<tr>
<td></td>
<td><strong>False Positive</strong></td>
<td><strong>True Negative</strong></td>
</tr>
</tbody>
</table>

Harmless error
The Role of How a Hypothesis is Framed (2)

Default hypothesis in criminal proceedings: An accused IS NOT guilty

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis holds</td>
<td>Accused is acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td></td>
<td>Should be acquitted</td>
<td>Should be acquitted</td>
</tr>
<tr>
<td></td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td>Accused is acquitted</td>
<td>Accused is sentenced</td>
</tr>
<tr>
<td></td>
<td>Should be sentenced</td>
<td>Should be sentenced</td>
</tr>
<tr>
<td></td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Harmless error

Harmful error
### The Role of How a Hypothesis is Framed (3)

Assume the non-negated hypothesis: An accused IS guilty

<table>
<thead>
<tr>
<th>Hypothesis Accepted</th>
<th>Hypothesis Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Harmful error</strong></td>
<td><strong>Harmless error</strong></td>
</tr>
<tr>
<td>Hypothesis holds</td>
<td></td>
</tr>
<tr>
<td>Accused is sentenced</td>
<td>Accused is acquitted</td>
</tr>
<tr>
<td>Should be sentenced</td>
<td>Should be acquitted</td>
</tr>
<tr>
<td><strong>True Positive</strong></td>
<td><strong>False Negative</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Hypothesis does not hold</td>
<td></td>
</tr>
<tr>
<td>Accused is sentenced</td>
<td>Accused is acquitted</td>
</tr>
<tr>
<td>Should be acquitted</td>
<td>Should be acquitted</td>
</tr>
<tr>
<td><strong>False Positive</strong></td>
<td><strong>True Negative</strong></td>
</tr>
</tbody>
</table>
Efficiency and Scalability

• Efficiency
  ▶ How well are resources used
  ▶ Measured in terms of work done per unit resource
  ▶ Resources: time, memory, power, energy, processors, network etc.
  ▶ Example: Strike rate of a batter in cricket

• Scalability
  ▶ How large inputs can be handled
  ▶ Measured in terms of size of the input
  ▶ Example: Total runs scored by a batter in cricket

• Efficiency and scalability are orthogonal
  ▶ Efficiency does not necessarily imply scalability
  ▶ Scalability does not necessarily imply efficiency
Efficiency and Scalability May be Unrelated

Examples of the combinations of efficiency and scalability from sorting algorithms

<table>
<thead>
<tr>
<th>Scalable</th>
<th>Efficient</th>
<th>Inefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge Sort</td>
<td>Selection Sort</td>
<td></td>
</tr>
<tr>
<td>Non-scalable</td>
<td>Quicksort</td>
<td>Bubble Sort</td>
</tr>
</tbody>
</table>
The goodness of a static analysis lies in minimizing imprecision without compromising on soundness.

Additional expectations: Efficiency and scalability.

Some applications (e.g. debugging) do not need to cover all traces.

Ex: Traffic police catching people for traffic violations.

Some features of a programming language may not be covered.

(e.g. “eval” in Javascript, aliasing of array indices, effect of libraries)

Accept a “soundy” analysis [Liveshits et. al. CACM 2015]

OR

Tolerate imprecision for complete soundness.