Introduction to Dataflow Analysis

CS 502
Lecture 8
9/30/08

Slides adapted from Nielson, Nielson, Hankin
Principles of Program Analysis
Context

- CPS exposes details about a program’s control-flow. What about dataflow?
- How do values produced by one expression flow to another?
- Dataflow analysis is concerned with defining relationship among program statements based on production and consumption of values produced by these statements.
Example

An example program and its naive realisation

Algol-like arrays:

\[
\begin{align*}
  i &:= 0; \\
  \text{while } i \leq n \text{ do} & \quad \text{while } j \leq m \text{ do} \\
  j &:= 0; & \\
  \text{A}[i,j] &:= \text{B}[i,j] + \text{C}[i,j]; & \\
  j &:= j+1 & \\
  \text{od}; & \\
  i &:= i+1 & \\
  \text{od}
\end{align*}
\]

C-like arrays:

\[
\begin{align*}
  i &:= 0; \\
  \text{while } i \leq n \text{ do} & \quad \text{while } j \leq m \text{ do} \\
  j &:= 0; & \\
  \text{temp} &:= \text{Base(A)} + i \times (m+1) + j; & \\
  \text{Cont(temp)} &:= \text{Cont(Base(B)} + i \times (m+1) + j) & \\
  & \quad + \text{Cont(Base(C)} + i \times (m+1) + j); & \\
  j &:= j+1 & \\
  \text{od}; & \\
  i &:= i+1 & \\
  \text{od}
\end{align*}
\]
Available Expressions analysis
and Common Subexpression Elimination

```
i := 0;
while i <= n do
  j := 0;
  while j <= m do
    temp := Base(A) + i*(m+1) + j;
    Cont(temp) := Cont(Base(B) + i*(m+1) + j) + Cont(Base(C) + i*(m+1) + j);
    j := j+1
  od;
  i := i+1
od
```

first computation

```
t1 := i * (m+1) + j;
temp := Base(A) + t1;
Cont(temp) := Cont(Base(B)+t1) + Cont(Base(C)+t1);
```

re-computations
i := 0;
while i <= n do
    j := 0;
    while j <= m do
        t1 := i * (m+1) + j;
        temp := Base(A) + t1;
        Cont(temp) := Cont(Base(B) + t1) + Cont(Base(C) + t1);
        j := j + 1
    od;
    i := i + 1
od

loop invariant

while j <= m do
    t1 := i * (m+1);
    while j <= m do
        t1 := t2 + j;
        temp := ...
        Cont(temp) := ...
        j := ...
    od
    t2 := i * (m+1);
    while j <= m do
        t1 := t2 + j;
        temp := ...
        Cont(temp) := ...
        j := ...
    od
Analysis and Optimization

Equivalent Expressions analysis and Copy Propagation

i := 0;
t3 := 0;
while i <= n do
  j := 0;
t2 := t3;
  while j <= m do
    t1 := t2 + j;
temp := Base(A) + t1;
    Cont(temp) := Cont(Base(B) + t1) + Cont(Base(C) + t1);
    j := j+1
  od;
i := i+1;
t3 := t3 + (m+1)
while j <= m do
  t1 := t3 + j;
temp := ...;
  Cont(temp) := ...;
  j := ...
o
Analysis and Optimization

Live Variables analysis and Dead Code Elimination

```
i := 0;
t3 := 0;
while i <= n do
  j := 0;
t2 := t3;
  while j <= m do
    t1 := t3 + j;
    temp := Base(A) + t1;
    Cont(temp) := Cont(Base(B) + t1)
    + Cont(Base(C) + t1);
    j := j + 1
  od;
i := i + 1;
t3 := t3 + (m+1)
od
```

```i := 0;
t3 := 0;
while i <= n do
  j := 0;
t2 := t3;
  while j <= m do
    t1 := t3 + j;
    temp := Base(A) + t1;
    Cont(temp) := Cont(Base(B) + t1)
    + Cont(Base(C) + t1);
    j := j + 1
  od;
i := i + 1;
t3 := t3 + (m+1)
od```
# Analysis and Optimization

## Summary of analyses and transformations

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available expressions analysis</td>
<td>Common subexpression elimination</td>
</tr>
<tr>
<td>Detection of loop invariants</td>
<td>Invariant code motion</td>
</tr>
<tr>
<td>Detection of induction variables</td>
<td>Strength reduction</td>
</tr>
<tr>
<td>Equivalent expression analysis</td>
<td>Copy propagation</td>
</tr>
<tr>
<td>Live variables analysis</td>
<td>Dead code elimination</td>
</tr>
</tbody>
</table>

© FyNielson Hyperriy Nielson CyHankin 2005s
Program Analysis

- Predict program behavior statically (at compile-time)

Goals:

- Safe: results of an analysis should be a conservative approximation of a program’s dynamic behavior
- Efficient: cost to compute this approximation should not be prohibitive
Techniques and Approaches

- Dataflow analysis
- Control-flow analysis
  - interprocedural analysis
- Type systems
- Abstract interpretation
  Different approaches to implementing these techniques (algorithmic, semantic, etc.)
- Language paradigms
Reaching Definition

Example program

Program with labels for elementary blocks:

\[y := x\]^1;
\[z := 1\]^2;
while \[y > 0\]^3 do
  \[z := z \ast y\]^4;
  \[y := y - 1\]^5
od;
\[y := 0\]^6

Flow graph:

- \[y := x\]^1
- \[z := 1\]^2
- \[y > 0\]^3
- \[z := z \ast y\]^4
- \[y := y - 1\]^5
- \[y := 0\]^6
Reaching Definitions

The assignment \([x := a]^{\ell}\) reaches \(\ell'\) if there is an execution where \(x\) was last assigned at \(\ell\).
Reaching Definitions

\[
\begin{align*}
[y := x]^{1}; & \quad \{ (x, ?), (y, ?), (z, ?) \} \\
[z := 1]^{2}; & \quad \{ (x, ?), (y, 1), (z, ?) \} \\
\text{while } [y > 0]^{3} \text{ do} & \quad \{ (x, ?), (y, 1), (z, 2) \} \\
[z := z \ast y]^{4}; & \quad \{ (x, ?), (y, 1), (z, 2) \} \\
[y := y - 1]^{5} & \quad \{ (x, ?), (y, 1), (z, 2) \} \\
\text{od;} & \quad \{ (x, ?), (y, 1), (z, 2) \} \\
[y := 0]^{6} & \quad \{ (x, ?), (y, 1), (z, 2) \}
\end{align*}
\]
Reaching Definitions (refinement)

\[ y := x \]

\[ z := 1 \]

while \( y > 0 \) do

\[ z := z \times y \]

\[ y := y - 1 \]

od;

\[ y := 0 \]
Reaching Definitions (optimal)

\[ y := x \]

\[ z := 1 \]

while \[ y > 0 \]

\[ z := z \ast y \]

\[ y := y - 1 \]

od;

\[ y := 0 \]
Reaching Definitions (safe, not optimal)

\[
\begin{align*}
[y := x] & 1; \\
[z := 1] & 2; \\
\text{while } [y > 0] & 3 \text{ do} \\
[z := z \times y] & 4; \\
[y := y - 1] & 5 \\
\text{od; } \\
[y := 0] & 6
\end{align*}
\]

\[
\begin{align*}
\begin{array}{l}
\{(x, ?), (y, ?), (z, ?)\} \\
\{(x, ?), (y, 1), (z, ?)\} \\
\{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
\{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \\
\{(x, ?), (y, 1), (y, 5), \text{(z, 2), (z, 4)}\} \\
\{(x, ?), (y, 1), (y, 5), \text{(z, 2), (z, 4)}\} \\
\{(x, ?), (y, 1), (y, 5), \text{(z, 2), (z, 4)}\} \\
\{(x, ?), (y, 1), (y, 5), \text{(z, 2), (z, 4)}\} \\
\{(x, ?), (y, 6), (z, 2), (z, 4)\} \end{array}
\]
Reaching Definitions (unsafe)

1.3

\[
y := x \]

1

\[
z := 1 \]

2

\[
\text{while } [y > 0] \text{ do }
\]

\[
z := z \times y \]

4

\[
y := y - 1 \]

5

\[
\text{od;}
\]

\[
y := 0 \]

6

\[
\{(x,?), (y,?), (z,?)\}
\]

\[
\{(x,?), (y,1), (z,?)\}
\]

\[
\{(x,?), (y,1), (z,?)\}
\]

\[
\{(x,?), (y,1), (z,2), (y,5), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,2), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,2), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,2), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,4)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,4)\}
\]

\[
\{(x,?)\}
\]

\[
\{(x,?), (y,1), (y,5), (z,4)\}
\]

\[
\{(x,?), (y,6), (z,2), (z,4)\}
\]
Approach

- Automate the analysis
  - Define a set of dataflow equations
  - Solve the equations using an iterative fixpoint algorithm
  - Solution is guaranteed to be the “smallest” solution that is safe:
    - No unnecessary overapproximation
Equations

Two kinds of equations

\[ [x := a]^\ell \]

\[ \text{RD}(\ell) \]

\[ \text{RD}^\bullet(\ell) \]

\[ \text{RD}(\ell) \setminus \{(x, \ell') \mid \ell' \in \text{Lab}\} \cup \{(x, \ell)\} = \text{RD}^\bullet(\ell) \]

\[ \text{RD}(\ell) \]

\[ \text{RD}^\bullet(\ell_1) \]

\[ \text{RD}^\bullet(\ell_2) \]

\[ \text{RD}(\ell) \]

\[ \text{RD}^\bullet(\ell_1) \cup \text{RD}^\bullet(\ell_2) = \text{RD}(\ell) \]
Dataflow Equations

\[ y := x \]

\[ z := 1 \]

while \([y > 0]\) do

\[ z := z * y \]

\[ y := y - 1 \]

od;

\[ y := 0 \]

\[ \text{Lab} = \{1,2,3,4,5,6\} \]

6 equations in \(RD_o(1), \ldots, RD_o(6)\)
Control-flow Considerations

\[ y := x \]
\[ z := 1 \]
while \( y > 0 \) do
\[ z := z \cdot y \]
\[ y := y - 1 \]
end;
\[ y := 0 \]

\( \text{Lab} = \{1,2,3,4,5,6\} \)

\( \text{RD}_o(1) = \{(x,?), (y,?), (z,?)\} \)
\( \text{RD}_o(2) = \text{RD}_o(1) \)
\( \text{RD}_o(3) = \text{RD}_o(2) \cup \text{RD}_o(5) \)
\( \text{RD}_o(4) = \text{RD}_o(3) \)
\( \text{RD}_o(5) = \text{RD}_o(4) \)
\( \text{RD}_o(6) = \text{RD}_o(3) \)

6 equations in \( \text{RD}_o(1), \ldots, \text{RD}_o(6) \)
Summary of Equations

\[
\begin{align*}
\text{RD}_1(1) &= \text{RD}_1(1) \setminus \{(y, \ell) \mid \ell \in \text{Lab}\} \cup \{(y, 1)\} \\
\text{RD}_1(2) &= \text{RD}_1(2) \setminus \{(z, \ell) \mid \ell \in \text{Lab}\} \cup \{(z, 2)\} \\
\text{RD}_1(3) &= \text{RD}_1(3) \\
\text{RD}_1(4) &= \text{RD}_1(4) \setminus \{(z, \ell) \mid \ell \in \text{Lab}\} \cup \{(z, 4)\} \\
\text{RD}_1(5) &= \text{RD}_1(5) \setminus \{(y, \ell) \mid \ell \in \text{Lab}\} \cup \{(y, 5)\} \\
\text{RD}_1(6) &= \text{RD}_1(6) \setminus \{(y, \ell) \mid \ell \in \text{Lab}\} \cup \{(y, 6)\} \\
\end{align*}
\]

- **12 sets**: \(\text{RD}_1(1), \cdots, \text{RD}_1(6)\) all being subsets of \(\text{Var} \times \text{Lab}\)
- **12 equations**:
  \[\text{RD}_j = F_j(\text{RD}_1(1), \cdots, \text{RD}_1(6))\]
- **one function**:
  \[F: \mathcal{P}(\text{Var} \times \text{Lab})^{12} \to \mathcal{P}(\text{Var} \times \text{Lab})^{12}\]
- we want the least fixed point of \(F\) — this is the best solution to the equation system
Solving the Equations

A simple iterative algorithm

- **Initialisation**
  \[ \text{RD}_1 := \emptyset; \cdots; \text{RD}_{12} := \emptyset; \]

- **Iteration**
  while \( \text{RD}_j \neq F_j(\text{RD}_1, \cdots, \text{RD}_{12}) \) for some \( j \)
  do
    \[ \text{RD}_j := F_j(\text{RD}_1, \cdots, \text{RD}_{12}) \]
## Iterative Process

<table>
<thead>
<tr>
<th>RD_0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>x?, y?, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>x?, y?, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>x?, y?, z?</td>
<td>x?, y1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>x?, y?, z?</td>
<td>x?, y1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>x?, y?, z?</td>
<td>x?, y1, z?</td>
<td>x?, y1, z2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>x?, y?, z?</td>
<td>x?, y1, z?</td>
<td>x?, y1, z2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RD_1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>x?, y1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>x?, y1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>x?, y1, z?</td>
<td>x?, y1, z2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>x?, y1, z?</td>
<td>x?, y1, z2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>x?, y1, z?</td>
<td>x?, y1, z2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

The equations:

RD_0(1) = RD_0(1) \ \{(y, \ell) | \cdots\} \cup \{(y, 1)\}

RD_0(2) = RD_0(2) \ \{(z, \ell) | \cdots\} \cup \{(z, 2)\}

RD_0(3) = RD_0(3)

RD_0(4) = RD_0(4) \ \{(z, \ell) | \cdots\} \cup \{(z, 4)\}

RD_0(5) = RD_0(5) \ \{(y, \ell) | \cdots\} \cup \{(y, 5)\}

RD_0(6) = RD_0(6) \ \{(y, \ell) | \cdots\} \cup \{(y, 6)\}

RD_1(1) = \{(x, ?), (y, ?), (z, ?)\}

RD_1(2) = RD_1(1)

RD_1(3) = RD_1(2) \cup RD_1(5)

RD_1(4) = RD_1(3)

RD_1(5) = RD_1(4)

RD_1(6) = RD_1(3)