Principles of Program Analysis:

A Sampler of Approaches

Compiler Optimisation

The classical use of program analysis is to facilitate the construction of compilers generating “optimal” code.

We begin by outlining the structure of optimising compilers.

We then prepare the setting for a worked example where we “optimise” a naive implementation of Algol-like arrays in a C-like language by performing a series of analyses and transformations.
The structure of a simple compiler

Characteristics of a simple compiler:

- many phases – one or more passes
- the compiler is fast – but the code is not very efficient
The structure of an optimising compiler

Characteristics of the optimising compiler:

- high-level optimisations: easy to adapt to new architectures
- low-level optimisations: less likely to port to new architectures
The structure of the optimisation phase

Avoid redundant computations: reuse available results, move loop invariant computations out of loops, ...

Avoid superfluous computations: results known not to be needed, results known already at compile time, ...
Example: Array Optimisation

Diagram:

- program with Algol-like arrays
- sequence of analysis and transformation steps
- optimised program with C-like arrays
Array representation: Algol vs. C

A: array \([0:n, 0:m]\) of integer

Accessing the \((i,j)\)'th element of \(A\):

in Algol:

\[ A[i,j] \]

in C:

\[ \text{Cont(Base}(A) + i \times (m+1) + j) \]
An example program and its naive realisation

Algol-like arrays:

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

C-like arrays:

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

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

\[
t1 := i \times (m+1) + j;
\text{temp} := \text{Base}(A) + t1;
\text{Cont}(\text{temp}) := \text{Cont}([\text{Base}(B)+t1] \\
&\quad + \text{Cont}([\text{Base}(C)+t1]);
\]

first computation
re-computations
Detection of Loop Invariants and Invariant Code Motion

\[
i := 0; \\
\text{while } i \leq n \text{ do} \\
\quad j := 0; \\
\quad \text{while } j \leq m \text{ do} \\
\quad \quad t1 := i \times (m+1) \text{ + } j; \\
\quad \quad \text{temp := Base(A) + t1; } \\
\quad \quad \text{Cont(temp) := Cont(Base(B) + t1) } \\
\quad \quad \quad + \text{ Cont(Base(C) + t1);} \\
\quad \quad j := j + 1 \\
\quad \text{od; } \\
\quad i := i + 1 \\
\text{od}
\]

loop invariant

\[
\text{loop invariant } \quad t2 := i \times (m+1); \\
\text{while } j \leq m \text{ do} \\
\quad t1 := t2 \text{ + } j; \\
\quad \text{temp := ...} \\
\quad \text{Cont(temp) := ...} \\
\quad j := ... \\
\text{od}
\]
Detection of Induction Variables and Reduction of Strength

\[ i := 0; \]
\[ \text{while } i \leq n \text{ do} \]
\[ j := 0; \]
\[ t2 := i \times (m+1); \]
\[ \text{while } j \leq m \text{ do} \]
\[ t1 := t2 + j; \]
\[ \text{temp} := \text{Base}(A) + t1; \]
\[ \text{Cont(temp)} := \text{Cont(Base}(B) + t1) \]
\[ + \text{Cont(Base}(C) + t1); \]
\[ j := j+1 \]
\[ \text{od}; \]
\[ i := i+1 \]
\[ \text{od} \]

\[ i := 0; \]
\[ t3 := 0; \]
\[ \text{while } i \leq n \text{ do} \]
\[ j := 0; \]
\[ t2 := t3; \]
\[ \text{while } j \leq m \text{ do ... od} \]
\[ i := i + 1; \]
\[ t3 := t3 + (m+1) \]
\[ \text{od} \]
Equivalent Expressions analysis and Copy Propagation

```plaintext
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)
od

while j <= m do
  t1 := t3 + j;
  temp := ...;
  Cont(temp) := ...;
  j := ...
  od
```
Live Variables analysis and Dead Code Elimination

```plaintext
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
```
## 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>
The Essence of Program Analysis

Program analysis offers techniques for predicting statical at compile-time
safe & efficient approximations to the set of configurations or behaviours arising dynamically at run-time

Safe: faithful to the semantics

Efficient: implementation with
– good time performance and
– low space consumption

we cannot expect exact answers!
The Nature of Approximation

The exact world

Over-approximation

Under-approximation

Slogans: Err on the safe side!
Trade precision for efficiency!
Approaches to Program Analysis

A family of techniques . . .

- data flow analysis
- constraint based analysis
- abstract interpretation
- type and effect systems
- . . .
- flow logic:
  a unifying framework

. . . that differ in their focus:

- algorithmic methods
- semantic foundations
- language paradigms
  — imperative/procedural
  — object oriented
  — logical
  — functional
  — concurrent/distributive
  — mobile
  — . . .
Data Flow Analysis

- **Technique:** Data Flow Analysis

- **Example:** Reaching Definitions analysis
  - idea
  - constructing an equation system
  - solving the equations
  - theoretical underpinnings
Example program

Program with labels for elementary blocks:

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

The assignment \([x := a]^{\ell}\) reaches \(\ell'\) if there is an execution where \(x\) was last assigned at \(\ell\).
Reaching Definitions analysis (1)

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

\[ y := x; \]
\[ z := 1; \]
\[ \text{while } y > 0 \text{ do} \]
\[ z := z \ast y; \]
\[ y := y - 1; \]
\[ \text{od}; \]
\[ y := 0; \]
Reaching Definitions analysis (3)

\[
\begin{align*}
[y := x]^1; & \quad \{(x,?), (y,?), (z,?)\} \\
[z := 1]^2; & \quad \{(x,?), (y,1), (z,?)\} \\
\text{while } [y > 0]^3 \text{ do} & \quad \{(x,?), (y,1), (y,5), (z,2), (z,4)\} \cup \{(y,5), (z,4)\} \\
[z := z \times y]^4; & \quad \{(x,?), (y,1), (y,5), (z,2), (z,4)\} \\
[y := y - 1]^5 & \quad \{(x,?), (y,1), (y,5), (z,4)\} \\
\text{od}; & \quad \{(x,?), (y,5), (z,4)\} \\
[y := 0]^6 & \quad \{(x,?), (y,1), (y,5), (z,2), (z,4)\}
\end{align*}
\]
The best solution

\[ y := x \]
\[ z := 1 \]

\textbf{while} \( y > 0 \) \textbf{do}
\[ z := z \times y \]
\[ y := y - 1 \]
\textbf{od;}
\[ y := 0 \]

\[ \{(x, ?), (y, ?), (z, ?)\} \]
\[ \{(x, ?), (y, 1), (z, ?)\} \]
\[ \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \]
\[ \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \]
\[ \{(x, ?), (y, 1), (y, 5), (z, 4)\} \]
\[ \{(x, ?), (y, 6), (z, 2), (z, 4)\} \]
A safe solution — but not the best

\[
y := x
\]

\[
z := 1
\]

while \(y > 0\)

\[
z := z \times y
\]

\[
y := y - 1
\]

od;

\[
y := 0
\]
An unsafe solution

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

\[ [y := 0]; \]
How to automate the analysis

extract equations from the program

compute the least solution to the equations

Analysis information:

- $RD_\circ(\ell)$: information available at the entry of block $\ell$
- $RD_\bullet(\ell)$: information available at the exit of block $\ell$
Two kinds of equations

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

\[
\text{RD}_\circ(\ell) \downarrow \text{RD}_\bullet(\ell)
\]

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

\[
\text{RD}_\bullet(\ell_1) \cup \text{RD}_\bullet(\ell_2) = \text{RD}_\circ(\ell)
\]
Flow through assignments and tests

[y := x]¹;

[z := 1]²;

while [y > 0]³ do

[z := z * y]⁴;

[y := y - 1]⁵

od;

[y := 0]⁶

Lab = {1, 2, 3, 4, 5, 6}

RD•(1) = RD◦(1) \ {((y, ℓ) | ℓ ∈ Lab) ∪ {(y, 1)}}

RD•(2) = RD◦(2) \ {((z, ℓ) | ℓ ∈ Lab) ∪ {(z, 2)}}

RD•(3) = RD◦(3)

RD•(4) = RD◦(4) \ {((z, ℓ) | ℓ ∈ Lab) ∪ {(z, 4)}}

RD•(5) = RD◦(5) \ {((y, ℓ) | ℓ ∈ Lab) ∪ {(y, 5)}}

RD•(6) = RD◦(6) \ {((y, ℓ) | ℓ ∈ Lab) ∪ {(y, 6)}}

Lab = {1, 2, 3, 4, 5, 6}
Flow along the control

\[ y := x; \]
\[ z := 1; \]
while \( y > 0 \)
do
\[ z := z \times y; \]
\[ y := y - 1; \]
od;
\[ y := 0; \]
\[ Lab = \{1,2,3,4,5,6\} \]

\[ \text{RD}_\circ(1) = \{(x,?), (y,?), (z,?)\} \]
\[ \text{RD}_\circ(2) = \text{RD}_\bullet(1) \]
\[ \text{RD}_\circ(3) = \text{RD}_\bullet(2) \cup \text{RD}_\bullet(5) \]
\[ \text{RD}_\circ(4) = \text{RD}_\bullet(3) \]
\[ \text{RD}_\circ(5) = \text{RD}_\bullet(4) \]
\[ \text{RD}_\circ(6) = \text{RD}_\bullet(3) \]

6 equations in \( \text{RD}_\circ(1), \ldots, \text{RD}_\bullet(6) \)
Summary of equation system

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

- 12 sets: \text{RD}_\circ(1), \ldots, \text{RD}_\bullet(6)
  all being subsets of \text{Var} \times \text{Lab}
- 12 equations:
  \[
  \text{RD}_j = F_j(\text{RD}_\circ(1), \ldots, \text{RD}_\bullet(6))
  \]
- 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
  \[
  F \text{ — this is the best solution to the equation system}
  \]
How to solve the equations

A simple iterative algorithm

- **Initialisation**
  \[
  RD_1 := \emptyset; \cdots; RD_{12} := \emptyset;
  \]

- **Iteration**
  \[
  \text{while } RD_j \neq F_j(RD_1, \cdots, RD_{12}) \text{ for some } j \\
  \text{do}
  \]
  \[
  RD_j := F_j(RD_1, \cdots, RD_{12})
  \]

The algorithm terminates and computes the least fixed point of F.
The example equations

<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?, y_1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>x?, y?, z?</td>
<td>x?, y_1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>x?, y?, z?</td>
<td>x?, y_1, z?</td>
<td>x?, y_1, z_2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>x?, y?, z?</td>
<td>x?, y_1, z?</td>
<td>x?, y_1, z_2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RD_5</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?, y_1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>x?, y_1, z?</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>x?, y_1, z?</td>
<td>x?, y_1, z_2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>x?, y_1, z?</td>
<td>x?, y_1, z_2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>x?, y_1, z?</td>
<td>x?, y_1, z_2</td>
<td>x?, y_1, z_2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The equations:

RD_5(1) = RD_0(1) \ \{ (y, \ell) | \cdots \} \cup \{ (y, 1) \}
RD_5(2) = RD_0(2) \ \{ (z, \ell) | \cdots \} \cup \{ (z, 2) \}
RD_5(3) = RD_0(3)
RD_5(4) = RD_0(4) \ \{ (z, \ell) | \cdots \} \cup \{ (z, 4) \}
RD_5(5) = RD_0(5) \ \{ (y, \ell) | \cdots \} \cup \{ (y, 5) \}
RD_5(6) = RD_0(6) \ \{ (y, \ell) | \cdots \} \cup \{ (y, 6) \}

RD_5(1) = \{ (x,?), (y,?), (z,?) \}
RD_5(2) = RD_5(1)
RD_5(3) = RD_5(2) \cup RD_5(5)
RD_5(4) = RD_5(3)
RD_5(5) = RD_5(4)
RD_5(6) = RD_5(3)
Why does it work? (1)

A function \( f : \mathcal{P}(S) \to \mathcal{P}(S) \) is a \textit{monotone function} if

\[
V \subseteq V' \implies f(V) \subseteq f(V')
\]

(the larger the argument – the larger the result)
Why does it work? (2)

A set $L$ equipped with an ordering $\subseteq$ satisfies the Ascending Chain Condition if all chains

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \cdots$$

stabilise, that is, if there exists some $n$ such that $V_n = V_{n+1} = V_{n+2} = \cdots$

If $S$ is a finite set then $\mathcal{P}(S)$ equipped with the subset ordering $\subseteq$ satisfies the Ascending Chain Condition — the chains cannot grow forever since each element is a subset of a finite set.

**Fact**

For a given program $\text{Var} \times \text{Lab}$ will be a finite set so $\mathcal{P}(\text{Var} \times \text{Lab})$ with the subset ordering satisfies the Ascending Chain Condition.
Why does it work? (3)

Let $f : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ be a monotone function. Then

$$\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq f^3(\emptyset) \subseteq \cdots$$

Assume that $S$ is a finite set; then the Ascending Chain Condition is satisfied. This means that the chain cannot be growing infinitely so there exists $n$ such that $f^n(\emptyset) = f^{n+1}(\emptyset) = \cdots$

$f^n(\emptyset)$ is the least fixed point of $f$
Correctness of the algorithm

- **Initialisation**
  
  \[ \text{RD}_1 := \emptyset; \cdots; \text{RD}_{12} := \emptyset; \]
  
  Invariant: \( \vec{\text{RD}} \subseteq F^n(\vec{\emptyset}) \) since \( \vec{\text{RD}} = \vec{\emptyset} \) is the least element

- **Iteration**
  
  while \( \text{RD}_j \neq F_j(\text{RD}_1, \cdots, \text{RD}_{12}) \) for some \( j \)
  
  do 
  
  assume \( \vec{\text{RD}} \) is \( \vec{\text{RD}}' \) and \( \vec{\text{RD}}' \subseteq F^n(\vec{\emptyset}) \)
  
  \[ \text{RD}_j := F_j(\text{RD}_1, \cdots, \text{RD}_{12}) \]
  
  then \( \vec{\text{RD}} \subseteq F(\vec{\text{RD}}') \subseteq F^{n+1}(\vec{\emptyset}) = F^n(\vec{\emptyset}) \) when \( \text{Ifp}(F) = F^n(\vec{\emptyset}) \)

If the algorithm terminates then it computes the least fixed point of \( F \).

The algorithm terminates because \( \text{RD}_j \subseteq F_j(\text{RD}_1, \cdots, \text{RD}_{12}) \) is only possible finitely many times since \( \mathcal{P}(\text{Var} \times \text{Lab})^{12} \) satisfies the Ascending Chain Condition.
Contraint Based Analysis

- **Technique:** Constraint Based Analysis

- **Example:** Control Flow Analysis
  - idea
  - constructing a constraint system
  - solving the constraints
  - theoretical underpinnings
Example: Control Flow Analysis

```plaintext
let f = fn x => x 7
g = fn y => y
h = fn z => 3

in f g + f (g h)

↓  ↓
g 7  f h

↓
h 7
```

Aim: For each function application, which function abstractions may be applied?

<table>
<thead>
<tr>
<th>function applications</th>
<th>function abstractions that may be applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 7</td>
<td>g, h</td>
</tr>
<tr>
<td>f g</td>
<td>f</td>
</tr>
<tr>
<td>g h</td>
<td>g</td>
</tr>
<tr>
<td>f (g h)</td>
<td>f</td>
</tr>
</tbody>
</table>
Solutions

let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)

The best solution:

A safe solution – but not the best:

<table>
<thead>
<tr>
<th>function applications</th>
<th>function abstractions that may be applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 7</td>
<td>g, h, f</td>
</tr>
<tr>
<td>f g</td>
<td>f</td>
</tr>
<tr>
<td>g h</td>
<td>g</td>
</tr>
<tr>
<td>f (g h)</td>
<td>f</td>
</tr>
</tbody>
</table>

An unsafe solution:

<table>
<thead>
<tr>
<th>function applications</th>
<th>function abstractions that may be applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 7</td>
<td>g, h</td>
</tr>
<tr>
<td>f g</td>
<td>f</td>
</tr>
<tr>
<td>g h</td>
<td>g</td>
</tr>
<tr>
<td>f (g h)</td>
<td>f</td>
</tr>
</tbody>
</table>
An application of control flow analysis

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

**Aim:** For each function application, which function abstractions may be applied?

Partial evaluation of function call:

```
let f = fn x => case x of
g: 7
h: 3
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

<table>
<thead>
<tr>
<th>function applications</th>
<th>function abstractions that may be applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 7</td>
<td>g, h</td>
</tr>
<tr>
<td>f g</td>
<td>f</td>
</tr>
<tr>
<td>g h</td>
<td>g</td>
</tr>
<tr>
<td>f (g h)</td>
<td>f</td>
</tr>
</tbody>
</table>
The underlying analysis problem

let f = fn x => x + 7
  g = fn y => y
  h = fn z => 3
in f g + f (g h)

Aim: for each function application, which function abstractions may be applied?

The analysis will compute:

- for each subexpression, which function abstractions may it denote?
  — e.g. \((g \ h)\) may evaluate to \(h\)

- introduce abstract cache \(C\)

- for each variable, which function abstractions may it denote?
  — e.g. \(x\) may be \(g\) or \(h\)

- introduce abstract environment \(R\)
The best solution to the analysis problem

Add labels to subexpressions:

\[
\begin{align*}
\text{let } f &= \text{fn } x \Rightarrow (x^1 7^2)^3 \\
g &= \text{fn } y \Rightarrow y^4 \\
h &= \text{fn } z \Rightarrow 3^5 \\
\text{in } (f^6 g^7)^8 + (f^9 (g^{10} h^{11})^{12})^{13}
\end{align*}
\]

<table>
<thead>
<tr>
<th>R</th>
<th>variable may be bound to</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{fn y =&gt; y^4, fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>y</td>
<td>{fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>z</td>
<td>∅</td>
</tr>
<tr>
<td>f</td>
<td>{fn x =&gt; (x^1 7^2)^3}</td>
</tr>
<tr>
<td>g</td>
<td>{fn y =&gt; y^4}</td>
</tr>
<tr>
<td>h</td>
<td>{fn z =&gt; 3^5}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>subexpression may evaluate to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{fn y =&gt; y^4, fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>{fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>{fn x =&gt; (x^1 7^2)^3}</td>
</tr>
<tr>
<td>7</td>
<td>{fn y =&gt; y^4}</td>
</tr>
<tr>
<td>8</td>
<td>∅</td>
</tr>
<tr>
<td>9</td>
<td>{fn x =&gt; (x^1 7^2)^3}</td>
</tr>
<tr>
<td>10</td>
<td>{fn y =&gt; y^4}</td>
</tr>
<tr>
<td>11</td>
<td>{fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>12</td>
<td>{fn z =&gt; 3^5}</td>
</tr>
<tr>
<td>13</td>
<td>∅</td>
</tr>
</tbody>
</table>
How to automate the analysis

extract constraints from the program

compute the least solution to the constraints

Analysis information:

• $R(x)$: information available for the variable $x$

• $C(\ell)$: information available at the subexpression labelled $\ell$
Three kinds of constraints

• \texttt{let}-bound variables evaluate to their abstraction

• variables evaluate to their (abstract) values

• \texttt{if} a function abstraction is applied to an argument \texttt{then}
  
  – the argument is a possible value of the formal parameter
  
  – the value of the body of the abstraction is a possible value of the application
let-bound variables

let-bound variables evaluate to their abstractions

let \( f = \text{fn} \; x \Rightarrow e \) gives rise to the constraint \( \{ \text{fn} \; x \Rightarrow e \} \subseteq R(f) \)

\[
\begin{align*}
\text{let } f &= \text{fn} \; x \Rightarrow (x^1 \; 7^2)^3 & \{ \text{fn} \; x \Rightarrow (x^1 \; 7^2)^3 \} \subseteq R(f) \\
g &= \text{fn} \; y \Rightarrow y^4 & \{ \text{fn} \; y \Rightarrow y^4 \} \subseteq R(g) \\
in \; (f^5 \; g^6)^7
\end{align*}
\]
Variables

Variables evaluate to their abstract values

\[ x^\ell \] gives rise to the constraint \( R(x) \subseteq C(\ell) \)

\[
\begin{align*}
\text{let } & f = \text{fn } x \Rightarrow (x^1 \cdot 7^2)^3 & \text{R(x) \subseteq C(1)} \\
\text{g = fn y \Rightarrow y}^4 & \text{R(y) \subseteq C(4)} \\
\text{in (f}^5 \cdot g^6)^7 & \left\{ \begin{array}{l}
\text{R(f) \subseteq C(5)} \\
\text{R(g) \subseteq C(6)}
\end{array} \right.
\end{align*}
\]
Function application (1)

if a function abstraction is applied to an argument then

- the argument is a possible value of the formal parameter
- the value of the body of the abstraction is a possible value of the application

let \( f = \text{fn x} \Rightarrow (x^1 \; 7^2)^3 \)
\( g = \text{fn y} \Rightarrow y^4 \)
in \( (f^5 \; g^6)^7 \)

Conditional constraints

if \( (\text{fn y} \Rightarrow y^4) \in C(1) \)
then \( C(2) \subseteq R(y) \) and \( C(4) \subseteq C(3) \)

if \( (\text{fn x} \Rightarrow (x^1 \; 7^2)^3) \in C(1) \)
then \( C(2) \subseteq R(x) \) and \( C(3) \subseteq C(3) \)
Function application (2)

if a function abstraction is applied to an argument then

• the argument is a possible value of the formal parameter

• the value of the body of the abstraction is a possible value of the application

let \( f = \text{fn } x \Rightarrow (x^1 \ 7^2)^3 \)

\( g = \text{fn } y \Rightarrow y^4 \)

in \((f^5 \ g^6)^7\)

if \((\text{fn } y \Rightarrow y^4) \in C(5)\) then \(C(6) \subseteq R(y)\) and \(C(4) \subseteq C(7)\)

if \((\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(5)\) then \(C(6) \subseteq R(x)\) and \(C(3) \subseteq C(7)\)
Summary of constraint system

\{fn x \mapsto (x^1 7^2)^3\} \subseteq R(f)
\{fn y \mapsto y^4\} \subseteq R(g)
R(x) \subseteq C(1)
R(y) \subseteq C(4)
R(f) \subseteq C(5)
R(g) \subseteq C(6)

(fn y \mapsto y^4) \in C(1) \Rightarrow C(2) \subseteq R(y)
(fn y \mapsto y^4) \in C(1) \Rightarrow C(4) \subseteq C(3)
(fn x \mapsto (x^1 7^2)^3) \in C(1) \Rightarrow C(2) \subseteq R(x)
(fn x \mapsto (x^1 7^2)^3) \in C(1) \Rightarrow C(3) \subseteq C(3)
(fn y \mapsto y^4) \in C(5) \Rightarrow C(6) \subseteq R(y)
(fn y \mapsto y^4) \in C(5) \Rightarrow C(4) \subseteq C(7)
(fn x \mapsto (x^1 7^2)^3) \in C(5) \Rightarrow C(6) \subseteq R(x)
(fn x \mapsto (x^1 7^2)^3) \in C(5) \Rightarrow C(3) \subseteq C(7)

- **11 sets:** \(R(x), R(y), R(f), R(g), C(1), \ldots, C(7)\); all being subsets of the set \textbf{Abstr} of function abstractions
- the constraints can be reformulated as a function:
  \(F : \mathcal{P}(\text{Abstr})^{11} \rightarrow \mathcal{P}(\text{Abstr})^{11}\)
- we want the least fixed point of \(F\) — this is the best solution to the constraint system

\(\mathcal{P}(S)\) is the set of all subsets of the set \(S\); e.g. \(\mathcal{P}\{0, 1\} = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}\).
The constraints define a function

\{ \text{fn } x \Rightarrow (x^1 7^2)^3 \} \subseteq R(f)
\{ \text{fn } y \Rightarrow y^4 \} \subseteq R(g)
R(x) \subseteq C(1)
R(y) \subseteq C(4)
R(f) \subseteq C(5)
R(g) \subseteq C(6)

(FN y \Rightarrow y^4) \in C(1) \Rightarrow C(2) \subseteq R(y)
(FN y \Rightarrow y^4) \in C(1) \Rightarrow C(4) \subseteq C(3)
(FN x \Rightarrow (x^1 7^2)^3) \in C(1) \Rightarrow C(2) \subseteq R(x)
(FN x \Rightarrow (x^1 7^2)^3) \in C(1) \Rightarrow C(3) \subseteq C(3)
(FN y \Rightarrow y^4) \in C(5) \Rightarrow C(6) \subseteq R(y)
(FN y \Rightarrow y^4) \in C(5) \Rightarrow C(4) \subseteq C(7)
(FN x \Rightarrow (x^1 7^2)^3) \in C(5) \Rightarrow C(6) \subseteq R(x)
(FN x \Rightarrow (x^1 7^2)^3) \in C(5) \Rightarrow C(3) \subseteq C(7)

F : \mathcal{P}(\text{Abstr})^{11} \rightarrow \mathcal{P}(\text{Abstr})^{11}
is defined by

\begin{align*}
F_{R(f)}(\cdots, R_f, \cdots) &= R_f \cup \{ \text{fn } x \Rightarrow (x^1 7^2)^3 \} \\
F_{C(1)}(R_x, \cdots, C_1, \cdots) &= C_1 \cup R_x \\
F_{R(y)}(\cdots, R_y, \cdots, C_1, C_2, \cdots, C_5, C_6, \cdots) &= R_y \cup \{ a \in C_2 \mid \text{fn } y \Rightarrow y^4 \in C_1 \} \\
& \quad \cup \{ a \in C_6 \mid \text{fn } y \Rightarrow y^4 \in C_5 \}
\end{align*}
How to solve the constraints

- **Initialisation**
  \[
  X_1 := \emptyset; \ldots; X_{11} := \emptyset;
  \]

- **Iteration**
  \[
  \text{while } X_j \neq F_{X_j}(X_1, \ldots, X_{11}) \text{ for some } j \\
  \text{do} \\
  X_j := F_{X_j}(X_1, \ldots, X_{11})
  \]

  The algorithm terminates and computes the least fixed point of \( F \)

In practice we propagate smaller contributions than \( F_{X_j} \), e.g. a constraint at a time.
The example constraint system

<table>
<thead>
<tr>
<th>R</th>
<th>x</th>
<th>y</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>∅</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>fn y =&gt; .4</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>fn y =&gt; .4</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>fn y =&gt; .4</td>
</tr>
<tr>
<td>5</td>
<td>fn y =&gt; .4</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>fn y =&gt; .4</td>
</tr>
<tr>
<td>6</td>
<td>fn y =&gt; .4</td>
<td>∅</td>
<td>fn x =&gt; .3</td>
<td>fn y =&gt; .4</td>
</tr>
</tbody>
</table>

The constraints:

\[
\begin{align*}
\{ \text{fn x => } .3 \} & \subseteq R(f) \\
\{ \text{fn y => } .4 \} & \subseteq R(g) \\
R(x) & \subseteq C(1) \\
R(y) & \subseteq C(4) \\
R(f) & \subseteq C(5) \\
R(g) & \subseteq C(6) \\
(\text{fn y => } .4) & \in C(1) \Rightarrow C(2) \subseteq R(y) \\
(\text{fn y => } .4) & \in C(1) \Rightarrow C(4) \subseteq C(3) \\
(\text{fn x => } .3) & \in C(1) \Rightarrow C(2) \subseteq R(x) \\
(\text{fn x => } .3) & \in C(1) \Rightarrow C(3) \subseteq C(3) \\
(\text{fn y => } .4) & \in C(5) \Rightarrow C(6) \subseteq R(y) \\
(\text{fn y => } .4) & \in C(5) \Rightarrow C(4) \subseteq C(7) \\
(\text{fn x => } .3) & \in C(5) \Rightarrow C(6) \subseteq R(x) \\
(\text{fn x => } .3) & \in C(5) \Rightarrow C(3) \subseteq C(7)
\end{align*}
\]
Why does it work? (1)

A function $f : \mathcal{P}(S) \to \mathcal{P}(S)$ is a **monotone function** if

$$V \subseteq V' \implies f(V) \subseteq f(V')$$

(the larger the argument — the larger the result)
Why does it work? (2)

A set $L$ equipped with an ordering $\subseteq$ satisfies the Ascending Chain Condition if all chains

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \cdots$$

stabilise, that is, if there exists some $n$ such that $V_n = V_{n+1} = V_{n+2} = \cdots$

If $S$ is a finite set then $\mathcal{P}(S)$ equipped with the subset ordering $\subseteq$ satisfies the Ascending Chain Condition — the chains cannot grow forever since each element is a subset of a finite set.

Fact
For a given program $\text{Abstr}$ will be a finite set so $\mathcal{P}(\text{Abstr})$ with the subset ordering satisfies the Ascending Chain Condition.
Why does it work? (3)

Let \( f : \mathcal{P}(S) \rightarrow \mathcal{P}(S) \) be a monotone function. Then

\[
\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq f^3(\emptyset) \subseteq \ldots
\]

Assume that \( S \) is a finite set; then the Ascending Chain Condition is satisfied. This means that the chain cannot grow infinitely so there exists \( n \) such that \( f^n(\emptyset) = f^{n+1}(\emptyset) = \ldots \)

\( f^n(\emptyset) \) is the least fixed point of \( f \)

\[
lfp(f) = f^n(\emptyset) = f^{n+1}(\emptyset) \text{ for some } n
\]
Correctness of the algorithm

- **Initialisation**
  \[ X_1 := \emptyset; \cdots; X_{11} := \emptyset; \]
  Invariant: \( \vec{X} \subseteq F^n(\emptyset) \) since \( \vec{X} = \emptyset \) is the least element

- **Iteration**
  while \( X_j \neq F_{X_j}(X_1, \cdots, X_{11}) \) for some \( j \)
  do assume \( \vec{X} \) is \( \vec{X}' \) and \( \vec{X}' \subseteq F^n(\emptyset) \)
  \[ X_j := F_{X_j}(X_1, \cdots, X_{11}) \]
  then \( \vec{X} \subseteq F(\vec{X}') \subseteq F^{n+1}(\emptyset) = F^n(\emptyset) \) when \( \text{lfp}(F) = F^n(\emptyset) \)

If the algorithm terminates then it computes the least fixed point of \( F \)

The algorithm terminates because \( X_j \subseteq F_{X_j}(X_1, \cdots, X_{11}) \) is only possible finitely many times since \( \mathcal{P}(\text{AbsExp})^{11} \) satisfies the Ascending Chain Condition
Abstract Interpretation

- **Technique:** Abstract Interpretation

- **Example:** Reaching Definitions analysis
  - idea
  - collecting semantics
  - Galois connections
  - Inducing the analysis
Abstract Interpretation

• We have the analysis old: it has already been proved correct but it is inefficient, or maybe even uncomputable.

• We want the analysis new: it has to be correct as well as efficient!

• Can we develop new from old? abstract interpretation!
Example: Collecting Semantics and Reaching Definitions

The **collecting semantics** CS

- collects the set of traces that can reach a given program point
- has an easy correctness proof
- is uncomputable

The **reaching definitions analysis** RD is as before
Example: Collecting Semantics

Collect the set of traces that reach a given program point \( \ell \)

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

\[
[y := x]^1
\]

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

\[
[z := 1]^2
\]

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

\[
[y > 0]^3
\]

\[
[y := 0]^6
\]

\[
[z := z * y]^4
\]

\[
[y := y - 1]^5
\]

\[
\{ (x,?):(y,?):(z,?): (y,1):(z,2):(z,4):(y,5), (x,?):(y,?):(z,?): (y,1):(z,2):(z,4):(y,5):(z,4):(y,5), \ldots \}
\]
How to proceed

As before:

- extract a set of equations defining the possible sets of traces
- compute the least fixed point of the set of equations

And furthermore:

- prove the correctness: the set of traces computed by the analysis is a superset of the possible traces
Two kinds of equations

\[ x \leftarrow a \]

\[ \ell \quad \text{CS} \circ (\ell) \]

\[ \ell \quad \text{CS} \bullet (\ell) \]

\[ \{ \text{trace} : (x, \ell) \mid \text{trace} \in \text{CS} \circ (\ell) \} = \text{CS} \bullet (\ell) \]

\[ \text{CS} \bullet (\ell_1) \cup \text{CS} \bullet (\ell_2) = \text{CS} \circ (\ell) \]
Flow through assignments and tests

\[
\begin{align*}
[y & := x]^1; \\
[z & := 1]^2; & \quad \text{CS}_\bullet(1) = \{\text{trace} : (y, 1) \mid \text{trace} \in \text{CS}_\circ(1)\} \\
\text{while } [y & > 0]^3 \text{ do} & \quad \text{CS}_\bullet(2) = \{\text{trace} : (z, 2) \mid \text{trace} \in \text{CS}_\circ(2)\} \\
[z & := z \ast y]^4; & \quad \text{CS}_\bullet(3) = \text{CS}_\circ(3) \\
[y & := y - 1]^5 & \quad \text{CS}_\bullet(4) = \{\text{trace} : (z, 4) \mid \text{trace} \in \text{CS}_\circ(4)\} \\
\od & \quad \text{CS}_\bullet(5) = \{\text{trace} : (y, 5) \mid \text{trace} \in \text{CS}_\circ(5)\} \\
[y & := 0]^6 & \quad \text{CS}_\bullet(6) = \{\text{trace} : (y, 6) \mid \text{trace} \in \text{CS}_\circ(6)\} \\
\end{align*}
\]

6 equations in \(\text{CS}_\circ(1), \cdots, \text{CS}_\circ(6)\)
Flow along the control

\[
\begin{align*}
[y := x]^1; & \quad \text{CS}_o(1) = \{(x, ?) : (y, ?) : (z, ?)\} \\
[z := 1]^2; & \quad \text{CS}_o(2) = \text{CS}_i(1) \\
\text{while } [y > 0]^3 \text{ do} & \quad \text{CS}_o(3) = \text{CS}_i(2) \cup \text{CS}_i(5) \\
[z := z \ast y]^4; & \quad \text{CS}_o(4) = \text{CS}_i(3) \\
[y := y - 1]^5 & \quad \text{CS}_o(5) = \text{CS}_i(4) \\
\text{od; } & \quad \text{CS}_o(6) = \text{CS}_i(3) \\
[y := 0]^6 & \quad \text{6 equations in CS}_o(1), \ldots, \text{CS}_i(6) \end{align*}
\]
Summary of Collecting Semantics

\[ \text{CS}_\bullet(1) = \{ \text{trace} : (y, 1) \mid \text{trace} \in \text{CS}_\circ(1) \} \]
\[ \text{CS}_\bullet(2) = \{ \text{trace} : (z, 2) \mid \text{trace} \in \text{CS}_\circ(2) \} \]
\[ \text{CS}_\bullet(3) = \text{CS}_\circ(3) \]
\[ \text{CS}_\bullet(4) = \{ \text{trace} : (z, 4) \mid \text{trace} \in \text{CS}_\circ(4) \} \]
\[ \text{CS}_\bullet(5) = \{ \text{trace} : (y, 5) \mid \text{trace} \in \text{CS}_\circ(5) \} \]
\[ \text{CS}_\bullet(6) = \{ \text{trace} : (y, 6) \mid \text{trace} \in \text{CS}_\circ(6) \} \]

\[ \text{CS}_\circ(1) = \{(x, ?) : (y, ?) : (z, ?)\} \]
\[ \text{CS}_\circ(2) = \text{CS}_\bullet(1) \]
\[ \text{CS}_\circ(3) = \text{CS}_\bullet(2) \cup \text{CS}_\bullet(5) \]
\[ \text{CS}_\circ(4) = \text{CS}_\bullet(3) \]
\[ \text{CS}_\circ(5) = \text{CS}_\bullet(4) \]
\[ \text{CS}_\circ(6) = \text{CS}_\bullet(3) \]

- **12 sets:** \( \text{CS}_\circ(1), \ldots, \text{CS}_\circ(6) \) all being subsets of \( \text{Trace} \)
- **12 equations:**
  \[ \text{CS}_j = G_j(\text{CS}_\circ(1), \ldots, \text{CS}_\circ(6)) \]
- **one function:**
  \[ G : \mathcal{P}(\text{Trace})^{12} \rightarrow \mathcal{P}(\text{Trace})^{12} \]
- we want the least fixed point of \( G \) — but it is uncomputable!
Example: Inducing an analysis

Galois Connections

A Galois connection between two sets is a pair of \((\alpha, \gamma)\) of functions between the sets satisfying

\[
X \subseteq \gamma(Y) \iff \alpha(X) \subseteq Y
\]

\(\mathcal{P}(\text{Trace})\) collecting semantics \(\mathcal{P}(\text{Var} \times \text{Lab})\) reaching definitions

\(\alpha\): abstraction function
\(\gamma\): concretisation function

PPA Section 1.5 © F.Nielson & H.Riis Nielson & C.Hankin (May 2005)
Semantically Reaching Definitions

For a single trace:

\[ \text{trace: } (x,?)\langle y,?\rangle\langle z,?\rangle\langle y,1\rangle\langle z,2 \rangle \]

\[ \text{SRD}(\text{trace}): \{ (x,?)\langle y,1\rangle\langle z,2 \rangle \} \]

For a set of traces:

\[ X \in \mathcal{P}(\text{Trace}): \{ (x,?)\langle y,?\rangle\langle z,?\rangle\langle y,1\rangle\langle z,2 \rangle, (x,?)\langle y,?\rangle\langle z,?\rangle\langle y,1\rangle\langle z,2 \rangle\langle z,4\rangle\langle y,5 \rangle \} \]

\[ \text{SRD}(X): \{ (x,?)\langle y,1\rangle\langle z,2 \rangle\langle z,4 \rangle\langle y,5 \rangle \} \]
Galois connection for Reaching Definitions analysis

\[ \alpha(X) = \text{SRD}(X) \]
\[ \gamma(Y) = \{ \text{trace} \mid \text{SRD}(\text{trace}) \subseteq Y \} \]

Galois connection:

\[ X \subseteq \gamma(Y) \iff \alpha(X) \subseteq Y \]
Inducing the Reaching Definitions analysis (1)

**Known:**
- $G_\bullet(4)$ defined on $\mathcal{P}(\text{Trace})$
- the Galois connection $(\alpha, \gamma)$

**Calculate:**
- $F_\bullet(4)$ defined on $\mathcal{P}(\text{Var} \times \text{Lab})$
  as $F_\bullet(4) = \alpha \circ G_\bullet(4) \circ \gamma$

\[
\begin{array}{c}
\mathcal{P}(\text{Trace}) \\
\alpha \\
G_\bullet(4) \\
\gamma \\
F_\bullet(4) = \alpha \circ G_\bullet(4) \circ \gamma \\
\mathcal{P}(\text{Var} \times \text{Lab})
\end{array}
\]
Inducing the Reaching Definitions analysis (2)

\[ \text{RD}_\bullet(4) = F_\bullet(4)(\ldots, \text{RD}_\circ(4), \ldots) \]
\[ = \alpha(G_\bullet(4)(\gamma(\ldots, \text{RD}_\circ(4), \ldots)))) \text{ using } F_\bullet(4) = \alpha \circ G_\bullet(4) \circ \gamma \]
\[ = \alpha(\{tr : (z, 4) \mid tr \in \gamma(\text{RD}_\circ(4))\}) \]
\[ \text{using } G_\bullet(4)(\ldots, \text{CS}_\circ(4), \ldots) = \{tr : (z, 4) \mid tr \in \text{CS}_\circ(4)\} \]
\[ = \text{SRD}(\{tr : (z, 4) \mid tr \in \gamma(\text{RD}_\circ(4))\}) \text{ using } \alpha = \text{SRD} \]
\[ = (\text{SRD}(\{tr \mid tr \in \gamma(\text{RD}_\circ(4))\})\setminus\{(z, \ell) \mid \ell \in \text{Lab}\})\cup\{(z, 4)\} \]
\[ = (\alpha(\gamma(\text{RD}_\circ(4)))\setminus\{(z, \ell) \mid \ell \in \text{Lab}\})\cup\{(z, 4)\} \text{ using } \alpha = \text{SRD} \]
\[ = (\text{RD}_\circ(4)\setminus\{(z, \ell) \mid \ell \in \text{Lab}\})\cup\{(z, 4)\} \text{ using } \alpha \circ \gamma = \text{id} \]

just as before!
Type and Effect Systems

- **Technique:** Annotated Type Systems

- **Example:** Reaching Definitions analysis
  - idea
  - annotated base types
  - annotated type constructors
The while language

• syntax of statements:

\[ S ::= [x:=a]^{\ell} | S_1; S_2 \]
\[ | \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \]
\[ | \text{while } [b]^\ell \text{ do } S \text{ od} \]

• semantics:

statements map states to states

• types:

\( \Sigma \) is the type of states;
all statements \( S \) have type \( \Sigma \rightarrow \Sigma \) written \( \vdash S : \Sigma \rightarrow \Sigma \)
Annotated base types

```
<table>
<thead>
<tr>
<th>type of initial state</th>
<th>type of final state(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ S : Σ → Σ</td>
<td></td>
</tr>
</tbody>
</table>
```

### Analysis:

```
<table>
<thead>
<tr>
<th>analysis information for initial state</th>
<th>analysis information for final state(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ S : RD₁ → RD₂</td>
<td></td>
</tr>
</tbody>
</table>
```

### Idea:

```
RD₁ ⊆ RDₒ(init(S))
↓
∀ℓ ∈ final(S) : RDₗ(ℓ) ⊆ RD₂
```
Annotated type system (1)

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

before \quad \text{after}

\[ \vdash S_1 : \text{RD}_1 \rightarrow \text{RD}_2 \quad \vdash S_2 : \text{RD}_2 \rightarrow \text{RD}_3 \]

assumptions \quad \text{conclusion}

\[ \vdash S_1; S_2 : \text{RD}_1 \rightarrow \text{RD}_3 \]

before \quad \text{after}

Implicit: the analysis information at the exit of \(S_1\)
equals the analysis information at the entry of \(S_2\)
Annotated type system (2)

\[ \vdash S_1 : RD_1 \rightarrow RD_2 \quad \vdash S_2 : RD_1 \rightarrow RD_2 \]
\[ \vdash \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} : RD_1 \rightarrow RD_2 \]

Implicit: the two branches have the same analysis information at their respective entry and exit points

\[ \vdash S : RD \rightarrow RD \]
\[ \vdash \text{while } [b]^\ell \text{ do } S \text{ od} : RD \rightarrow RD \]

Implicit: the occurrences of RD express an invariance i.e. a fixed point property!
Annotated type system (3)

The subsumption rule:

\[ \vdash S : RD_1 \rightarrow RD_2 \quad \text{if } RD_1 \subseteq RD'_1 \text{ and } RD'_2 \subseteq RD_2 \]

The rule is essential for the rules for conditional and iteration to work

- \( RD_1 \subseteq RD'_1 \): strengthen the analysis information for the initial state
- \( RD'_2 \subseteq RD_2 \): weaken the analysis information for the final states
Example inference in the annotated type system

Abbreviation: \( RD = \{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \)

\[ \vdash [z:=z*y]^4: RD \rightarrow \{(x, ?), (y, 1), (y, 5), (z, 4)\} \]
\[ \vdash [y:=y-1]^5: \{(x, ?), (y, 1), (y, 5), (z, 4)\} \rightarrow \{(x, ?), (y, 5), (z, 4)\} \]

\[ \vdash [z:=z*y]^4; [y:=y-1]^5: RD \rightarrow \{(x, ?), (y, 5), (z, 4)\} \]

\[ \vdash [z:=z*y]^4; [y:=y-1]^5: RD \rightarrow RD \]
using \( \{(x, ?), (y, 5), (z, 4)\} \subseteq RD \)

\[ \vdash \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od: } RD \rightarrow RD \]

\[ \vdots \]
\[ \vdash [y:=x]^1; [z:=1]^2; \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od; } [y:=0]^6: \]
\[ \{(x, ?), (y, ?), (z, ?)\} \rightarrow \{(x, ?), (y, 6), (z, 2), (z, 4)\} \]
How to automate the analysis

Specification
- annotated type system (axioms and rules)

Implementation
- extract constraints from the program
- compute the least solution to the constraints
Change of abstraction level: annotated type constructors

- **Until now:**
  given a statement and a *specific* entry information RD\(\circ\) we determine the *specific* exit information RD\(\bullet\).

- **Now:**
  given a statement we determine how entry information is transformed into exit information.
Annotated type constructors

- Type of initial state: $\Sigma$
- Type of final state(s): $\Sigma$

```
⊢ S : Σ → Σ
```

**Analysis:**

- Reaching definitions that might be produced by $S$
- Set of variables definitely assigned by $S$
Annotated type constructors (1)

\[
\vdash [x:=a]^{\ell} : \Sigma \quad \begin{array}{c} \{x\} \\ \{(x,\ell)\} \end{array} \rightarrow \Sigma
\]

\{x\} : \text{variables definitely assigned}
\{(x,\ell)\} : \text{potential reaching definitions}

\[
\vdash S_1 : \Sigma \quad \begin{array}{c} X_1 \\ RD_1 \end{array} \rightarrow \Sigma \quad \vdash S_2 : \Sigma \quad \begin{array}{c} X_2 \\ RD_2 \end{array} \rightarrow \Sigma
\]

\begin{array}{c}
\vdash S_1; S_2 : \Sigma \\
\begin{array}{c} X_1 \cup X_2 \\ (RD_1 \setminus X_2) \cup RD_2 \end{array} \rightarrow \Sigma
\end{array}

X_1 \cup X_2 : \text{variables definitely assigned}
(RD_1 \setminus X_2) \cup RD_2 : \text{potential reaching definitions}
Annotated type constructors (2)

\[\vdash S_1 : \Sigma \xrightarrow{X_1} \Sigma \quad \vdash S_2 : \Sigma \xrightarrow{X_2} \Sigma\]

\[\vdash \text{if } [b]^{\ell} \text{ then } S_1 \text{ else } S_2 \text{ fi} : \Sigma \xrightarrow{X_1 \cap X_2} \Sigma\]

\[X_1 \cap X_2 : \text{variables definitely assigned}\]

\[\text{RD}_1 \cup \text{RD}_2 : \text{potential reaching definitions}\]

\[\vdash S : \Sigma \xrightarrow{X} \Sigma\]

\[\vdash \text{while } [b]^{\ell} \text{ do } S \text{ od} : \Sigma \xrightarrow{\emptyset} \Sigma\]

\[\emptyset : \text{variables definitely assigned}\]

\[\text{RD} : \text{potential reaching definitions}\]
Annotated type constructors (3)

Subsumption rule:

\[
\frac{\vdash S : \Sigma}{\vdash S : \Sigma} \quad \frac{X \rightarrow \Sigma}{X' \rightarrow \Sigma}
\]

if \( X' \subseteq X \) (variables definite assigned)
and \( RD \subseteq RD' \) (potential reaching definitions)

the rule can be omitted!
Example inference in the annotated type system

\[ \vdash [z:=z*y]^4 : \Sigma \quad \vdash [y:=y-1]^5 : \Sigma \]

\[ \vdash [z:=z*y] ; [y:=y-1] : \Sigma \quad \vdash \text{while } [y>1]^3 \text{ do } [z:=z*y] ; [y:=y-1] \text{ od} : \Sigma \]

\[ \vdash [y:=x]^1 ; [z:=1]^2 ; \text{while } [y>1]^3 \text{ do } [z:=z*y] ; [y:=y-1] \text{ od} ; [y:=0]^6 : \]

\[ \Sigma \quad \vdash [\ldots] \quad \Sigma \]

PPA Section 1.6 © F.Nielson & H.Riis Nielson & C.Hankin (May 2005)
How to automate the analysis

**Specification**
- annotated type system (axioms and rules)

**Implementation**
- extract constraints from the program
  - compute the least solution to the constraints
Type and Effect Systems

- **Technique:** Effect systems

- **Example:** Call Tracking analysis
  - idea
  - simple type system
  - effect system
The fun language

- syntax of expressions
  \[ e ::= x \mid \text{fn}\pi x \Rightarrow e \mid e_1 \ e_2 \mid \cdots \]

- types
  \[ \tau ::= \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2 \]

\(\pi\) names the function abstraction

\(f\) has type \(\tau_1 \rightarrow \tau_2\) means that
- \(f\) expects a parameter of type \(\tau_1\)
- \(f\) returns a value of type \(\tau_2\)
Call Tracking analysis

let \( f = \text{fn}_F \ x \rightarrow x \ 7 \)
\( g = \text{fn}_G \ y \rightarrow y \)
\( h = \text{fn}_H \ z \rightarrow z \)

in \( f \ g + f \ (h \ g) \)

\[ \downarrow \quad \downarrow \]
\( g \ 7 \quad f \ g \)
\[ \downarrow \]
\( g \ 7 \)

**Aim:** For each function application, which function abstractions might be applied during its execution?

<table>
<thead>
<tr>
<th>function applications</th>
<th>function abstractions that might be applied during its execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \ 7 )</td>
<td>( G, H )</td>
</tr>
<tr>
<td>( f \ g )</td>
<td>( F, G )</td>
</tr>
<tr>
<td>( h \ g )</td>
<td>( H, G )</td>
</tr>
<tr>
<td>( f \ (h \ g) )</td>
<td>( F, H, G )</td>
</tr>
</tbody>
</table>
Simple types

\[
\begin{align*}
\text{let } & f = \text{fn}_F \ x \Rightarrow x \ 7 & f: (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \\
& g = \text{fn}_G \ y \Rightarrow y & g: \text{int} \rightarrow \text{int} \\
& h = \text{fn}_H \ z \Rightarrow z & h: (\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int}) \\
in \ & f \ g \ + \ f \ (h \ g) & \text{int}
\end{align*}
\]

Simple type system

- type environment: \( \Gamma \) gives types to the variables (like R)

- an expression \( e \) has type \( \tau \) relative to type environment \( \Gamma \) (like C)

\[ \Gamma \vdash e : \tau \]
A simple type system

\[ \Gamma \vdash x : \tau_x \quad \text{if } \Gamma(x) = \tau_x \]

\[ \Gamma \vdash \text{fn}_{\pi} \ x \Rightarrow e : \tau \]

\[ \Gamma \vdash e_1 : \tau_2 \rightarrow \tau, \quad \Gamma \vdash e_2 : \tau_2 \]

\[ \Gamma \vdash e_1 \ e_2 : \tau \]

guess: \( \tau_x \) is the type of the argument \( x \)

the type of the formal parameter equals that of the actual parameter
Effect systems

- **Call Tracking analysis:**
  For each function application, which function abstractions might be applied during its execution?

- **Idea:**
  Annotate the function arrows with sets of names of function abstractions that might be applied when calling the function.

\[
\tau_1 \xrightarrow{\phi} \tau_2
\]

the type of functions from \(\tau_1\) to \(\tau_2\) that might call functions with names in \(\phi\).
Example: Types and Effects

```plaintext
let f = fn_F x => x 7 ←→ f: (int ↪{G}→ int) ↪{F,G}→ int
g = fn_G y => y ←→ g: int ↪{G}→ int
h = fn_H z => z ←→ h: (int ↪{G}→ int) ↪{H}→ (int ↪{G}→ int)
in f g + f (h g) ←→ int & {F, G, H}

the effect of executing
f g + f (g h)
```

PPA Section 1.6  © F.Nielson & H.Riis Nielson & C.Hankin (May 2005) 93
The effect system

\[ \Gamma \vdash x : \tau_x \land \emptyset \quad \text{if } \Gamma(x) = \tau_x \]

variables have no effect

\[ \Gamma[x \mapsto \tau_x] \vdash e : \tau \land \varphi \]

the latent effect consists of
\begin{align*}
- & \text{that of the function body} \\
- & \text{the function itself} \\
- & \text{the function abstraction itself has no effect}
\end{align*}

\[ \Gamma \vdash \text{fn}_{\pi} x \Rightarrow e : \tau_x \quad \text{\(\varphi \cup \{\pi\}\)} \rightarrow \tau \land \emptyset \]

the overall effect comes from
\begin{align*}
- & \text{evaluating the function} \\
- & \text{evaluating the argument} \\
- & \text{evaluating the function application: the latent effect!}
\end{align*}
The effect system

The subsumption rule:

\[
\frac{\Gamma \vdash e : \tau \& \varphi}{\Gamma \vdash e : \tau \& \varphi \cup \varphi'}
\]

the names of functions that may be applied
Example (1)

let f = fn\(\text{F} x \mapsto x\) 7

g = fn\(\text{G} y \mapsto y\)

h = fn\(\text{H} z \mapsto z\)

\text{in } f \ g + f (h \ g)

\[\begin{array}{c}
\Gamma[x \mapsto \tau_x] \vdash e : \tau & \& \varphi \\
\Gamma \vdash \text{fn}_\pi x \mapsto e : \tau_x \quad \varphi \cup \{\pi\} \mapsto \tau & \& \emptyset \\
\Gamma \vdash e_1 : \tau_2 \quad \varphi \mapsto \tau \& \varphi_1 \\
\Gamma \vdash e_2 : \tau_2 \quad \varphi \mapsto \tau \& \varphi_2 \\
\Gamma \vdash e_1 \ e_2 : \tau \& \varphi_1 \cup \varphi_2 \cup \varphi
\end{array}\]

\[\begin{array}{c}
x \mapsto \text{int } \{\{\text{G}\}\} \mapsto \text{int} \vdash x : \text{int } \{\{\text{G}\}\} \mapsto \text{int} & \& \emptyset \\
x \mapsto \text{int } \{\{\text{G}\}\} \mapsto \text{int} \vdash 7 : \text{int} & \& \emptyset \\
x \mapsto \text{int } \{\{\text{G}\}\} \mapsto \text{int} \vdash x \ 7 : \text{int} & \& \{\{\text{G}\}\}
\end{array}\]

\[\begin{array}{c}
[] \vdash \text{fn}_\text{F} x \mapsto x : (\text{int } \{\{\text{G}\}\} \mapsto \text{int}) \quad \{\text{F,G}\} \mapsto \text{int} & \& \emptyset
\end{array}\]
Example (2)

let f = fn_F x => x
    g = fn_G y => y
    h = fn_H z => z
in f g + f (h g)

\[ \begin{array}{c}
\Gamma[x \mapsto \tau_x] \vdash e : \tau \land \varphi \\
\Gamma \vdash \text{fn}_\pi x => e : \tau_x \quad \varphi \cup \{\pi\} \rightarrow \tau \land \emptyset \\
\Gamma \vdash e_1 : \tau_2 \land \varphi_1 \\
\Gamma \vdash e_2 : \tau_2 \land \varphi_2 \\
\Gamma \vdash e_1 \ e_2 : \tau \land \varphi_1 \cup \varphi_2 \cup \varphi
\end{array} \]

\[ \begin{array}{c}
\Gamma \vdash h : (\text{int} \to^{G} \text{int}) \to^{H} (\text{int} \to^{G} \text{int}) \land \emptyset \\
\Gamma \vdash g : \text{int} \to^{G} \text{int} \land \emptyset \\
\Gamma \vdash h \ g : \text{int} \to^{G} \text{int} \land \{H\}
\end{array} \]

\[ \Gamma \vdash f \ (h \ g) : \text{int} \land \{F, G, H\} \]
How to automate the analysis

Specification
- annotated type system (axioms and rules)

Implementation
- extract constraints from the program
- compute the least solution to the constraints