Type-Based Flow Analysis: From Polymorphic Subtyping to CFL-Reachability

Jakob Rehof and Manuel Fähndrich

Microsoft Research

Presentation by Polyvios Pratikakis
some slides from the authors’ presentation.
Flow analysis

```plaintext
f(a) { return a ;
  x = 5;
  y = f(x);
  z = f(2);

  Which values flow in a?
  Does the value of x flow to z?
```
Types are annotated with labels $l$. 
Type-Based Flow analyses

- Types are annotated with labels $l$.
- Typecheck the program using an inference system, generating constraint sets on labels $l$. 

Solve constraints.

The question "is there ow from $e_1$ to $e_2$" is answered:

Find the types of expressions $e_1$, $e_2$:

$t_1$, $t_2$.

Find labels $l_1$ and $l_2$ of types.

"Is there ow from $l_1$ to $l_2$ given the solution of the constraints?"
Type-Based Flow analyses

- Types are annotated with labels $l$.
- Typecheck the program using an inference system, generating constraint sets on labels $l$.
- Solve constraints.
- Types are annotated with labels $l$.
- Typecheck the program using an inference system, generating constraint sets on labels $l$.
- Solve constraints.
- The question “is there flow from $e_1$ to $e_2$” is answered:
Types are annotated with labels $l$.

Typecheck the program using an inference system, generating constraint sets on labels $l$.

Solve constraints.

The question “is there flow from $e_1$ to $e_2$” is answered:
- Find the types of expressions $e_1, e_2: \tau_1, \tau_2$. 
Type-Based Flow analyses

- Types are annotated with labels $l$.
- Typecheck the program using an inference system, generating constraint sets on labels $l$.
- Solve constraints.
- The question “is there flow from $e_1$ to $e_2$” is answered:
  - Find the types of expressions $e_1, e_2: \tau_1, \tau_2$.
  - Find labels $l_1$ and $l_2$ of types.
Type-Based Flow analyses

- Types are annotated with labels $l$.
- Typecheck the program using an inference system, generating constraint sets on labels $l$.
- Solve constraints.
- The question “is there flow from $e_1$ to $e_2$” is answered:
  - Find the types of expressions $e_1, e_2: \tau_1, \tau_2$.
  - Find labels $l_1$ and $l_2$ of types.
  - “Is there flow from $l_1$ to $l_2$ given the solution of the constraints?”
Flow analysis properties

Ways to increase precision:

- Directional:
  “$l_1$ flows to $l_2$” $\neq$ “$l_2$ flows to $l_1$.”

- Context sensitive:
  Differenciate between different call sites to the same function.
Flow analysis properties

Ways to increase precision:

- Directional:
  “$l_1$ flows to $l_2$” $\not\equiv$ “$l_2$ flows to $l_1$.”

- Context sensitive:
  Differenciate between different call sites to the same function.

Approaches:

- Directional is subtyping ($\leq$) on $l$:
  $\text{int}^{l_1} \leq \text{int}^{l_2}$ if $f$ $l_1$ flows to $l_2$.

- Context sensitivity is polymorphism on labels $l$:
  $\forall l_1, l_2. \text{int}^{l_1} \rightarrow \text{int}^{l_2}$
Flow analyses performance

- Context Sensitive (CS)
- Directional (DI)

![Diagram showing relationships between different flow analysis types](image)

- +CS, +DI
- +CS, -DI
- -CS, -DI
- -CS, +DI

Precision, Cost

\( (\forall, \leq) \)
\( (\forall, =) \)
\( (=, \leq) \)
\( (=, =) \)
Goal

- Fast Context Sensitive, Directional analysis.
- Previously $O(n^8)$.
- This approach $O(n^3)$.

$n$ is the size of the program, annotated with types.
Example code:
\[
\text{max}(s, t) = \text{if } s \leq t \text{ then } t \text{ else } s
\]

Standard type:
\[
\text{real} \times \text{real} \rightarrow \text{real}
\]
Monomorphic analysis

Example code:

\[
\text{max}(s, t) = \begin{cases} t & \text{if } s \leq t \\ s & \text{else} \end{cases}
\]

Monomorphic analysis type:

\[\text{real}^{l_1} \times \text{real}^{l_2} \rightarrow \text{real}^{l_3}\]

Two call sites:

- \[\text{max}(x_1^{l_x}, y_1^{l_y})^{l_{call1}}\] produces \(\{l_x \leq l_1, l_y \leq l_2, l_3 \leq l_{call1}\}\).

- \[\text{max}(x_2^{l_x}, y_2^{l_y})^{l_{call2}}\] produces \(\{l_x \leq l_1, l_y \leq l_2, l_3 \leq l_{call2}\}\).

Solution indicates \(l_{x1}\) flows to \(l_{call2}\)!
Previous approach - $(\forall, \leq)$ analysis

Example code:
\[
\max(s^{l_1}, t^{l_2}) = \text{if } s \leq t \text{ then } t \text{ else } s^{l_3}
\]

Standard type:
\[
\text{real} \times \text{real} \rightarrow \text{real}
\]

Analysis type:
\[
\forall l_1, l_2, l_3. \{l_1 \leq l_3, l_2 \leq l_3\} \Rightarrow \text{real}^{l_1} \times \text{real}^{l_2} \rightarrow \text{real}^{l_3}
\]
Previous approach - \((\forall, \leq)\) analysis

Example code:
\[
\max(s^{l_1}, t^{l_2}) = \text{if } s \leq t \text{ then } t \text{ else } s^{l_3}
\]

Analysis type:
\[
\forall l_1, l_2, l_3. \{ l_1 \leq l_3, l_2 \leq l_3 \} \Rightarrow real^{l_1} \times real^{l_2} \rightarrow real^{l_3}
\]

For two call sites:
- \[
\max(x^{l_{x_1}}, y^{l_{y_1}}) \begin{array}{c}\text{call} \end{array} \text{ produces } \{ l_{x_1} \leq l_{\text{call}1}, l_{y_1} \leq l_{\text{call}1} \}.
\]
- \[
\max(x^{l_{x_2}}, y^{l_{y_2}}) \begin{array}{c}\text{call} \end{array} \text{ produces } \{ l_{x_2} \leq l_{\text{call}2}, l_{y_2} \leq l_{\text{call}2} \}.
\]
- A lot of copying, producing a lot of constraints.
- Not scalable.
New method

- Label all call sites of functions with “instantiation” labels $i$.
- Add “instantiation” edges from actual to formal arguments at call sites, instead of copying everything.
- Name the instantiation edges:
  - *Positive* ($+$) for flow from actual arguments to formal,
  - *negative* ($\div$) for flow from return statement to calling context.
- Add the instantiation label $i$ to these edges.
Create a graph with these two kinds of edges. Observations:

- Valid data flow only along paths that “match” call and return edges.
- Looks like “balancing parentheses”...
- We can add labels to edges:
  - \( (i \) for passing arguments in call site \( i \)
  - \( )_i \) for returning from call site \( i \)
- Only allow \( (i \) to match with \( )_i \) in flow paths.
- In essence, we have the graph size of monomorphic analysis, and the expressiveness of polymorphism!
Gain

- General CFL reachability is solvable in $O(m^3 \cdot n^3)$.
- This problem is a special case.
- Adapting the CFL reachability algorithm allows $O(n^3)$.
- Can be done without even creating the graph.
- Allows “demand driven” flow analysis (query/answer).
Formalism

\[\begin{align*}
\tau & ::= \text{int} \mid \tau \rightarrow \tau \mid \tau \times \tau \mid \ldots \\
\sigma & ::= \text{int}^l \mid \tau \rightarrow^l \tau \mid \tau \times^l \tau \mid \ldots
\end{align*}\]
The copy-polymorphism graph $G_{copy}$ is created by applying a substitution $\phi$ at every instantiation point.
For call sites $i$ and $j$:

$$\phi_i = \{l_1 \mapsto l'_1, \ldots\}$$

$$\phi_j = \{l''_1 \mapsto l'_1, \ldots\}$$

$$\vdash \sigma \preceq_i \sigma' : \phi_i$$

$$dom(\phi_i) = \vec{l}$$

$$C \vdash \phi_i(C')$$

$$\text{Inst-Copy} \quad \frac{}{C, A, f : \forall \vec{l}.C' \Rightarrow \sigma \vdash_{cp} f^i : \sigma'}$$
\[
\frac{\vdash \text{int}^{l_1} \rightarrow^l \text{int}^{l_2} \leq_i \text{int}_{x}^{l} \rightarrow^l \text{int}_{y}^{l} : \phi_i}{\text{dom}(\phi_i) = \overrightarrow{l}}
\]

\[
\frac{C \vdash \phi_i(C')}{C, A, \text{id} : \forall l_1, l_2. \{l_1 \leq l_2\} \Rightarrow \text{int}^{l_1} \rightarrow^l \text{int}^{l_2} \vdash_{cp} \text{id}^i : \text{int}_{x}^{l} \rightarrow^l \text{int}_{y}^{l}}
\]

\[
\phi_i = \{l_1 \mapsto l_x, l_2 \mapsto l_y\}
\]
Instead of applying $\phi_i$ to $C'$, “save” $\phi_i$ in the form of instantiation edges.

\[
l_1 \leq^i l_x
\]

\[
l_2 \leq^i l_y
\]

\[
I \vdash \sigma \leq^i \sigma' : \phi_i
\]

\[
dom(\phi_i) = \vec{l}
\]

\[
I \vdash \vec{l}' \leq^i \vec{l}'
\]

\[
I \vdash \vec{l}' \leq^i \vec{l}'
\]

\[
\text{Inst-CFL}
\]

\[
I, C, A, f : (\forall \vec{l}. \sigma, \vec{l}') \vdash_{cp} f^i : \sigma'
\]
Summary

- Program flow in $O(n^3)$.
- Context sensitive
- Directional
- Can be demand driven