Efficient Alias Set Analysis
Using SSA Form

Ondřej Lhoták     Nomair Naeem
How can **aliases** be represented in **SSA** form?

Not this talk.

How can **SSA** form help **alias** analysis?

This talk.

Observe some interesting SSA properties along the way...
Range of Pointer Analyses

Legend:
- Green circle: Concrete (run-time) object
- Blue square: Abstract (static) object

Points-to Analysis

Shape Analysis

Efficient analysis

Precise analysis

Efficient Alias Set Analysis Using SSA Form | Ondrej Lhotak, Nomair Naeem | University of Waterloo
Range of Pointer Analyses

Legend:
- Green: Concrete (run-time) object
- Blue: Abstract (static) object

Points-to Analysis
- p
- q
- r

Alias Set Analysis
- p
- q
- r

Shape Analysis
- p
- q

Efficient analysis
- Precise analysis
Why Alias Sets: Object Tracking

Efficient Alias Set Analysis Using SSA Form | Ondrej Lhotak, Nomair Naeem | University of Waterloo
Each element of the abstract domain is a set of abstract objects. Each abstract object is a set of pointer variables.

$p,q$ represents the object (if any) pointed by $p$ and $q$ and no other local variables.
Each element of the abstract domain is a set of abstract objects. Each abstract object is a set of pointer variables.

\( p, q \) represents the object (if any) pointed by \( p \) and \( q \) and no other local variables.
Transfer Functions

\[
p = \text{new}
\]

\[
r = q
\]

\[
q = s.f
\]

\[
\{p\}, o^\# \setminus \{p\}
\]

\[
\begin{cases}
  o^\# \setminus \{r\} & \text{if } q \not\in o^\#
  
  o^\# \cup \{r\} & \text{if } q \in o^#
\end{cases}
\]

\[
o^\# \setminus \{q\}, o^\# \cup \{q\}
\]
Benefits of SSA Form for Alias Set Analysis

**IF**

- Convert code to SSA form
- Represent each alias set by sorted list, ordered by dominance of (unique) definitions

**THEN**

😊 All inserts into set are at head of list
😊 All removals from set are at head of list
😊 All removals are at \( \phi \) nodes
😊 Tails of lists can be shared (hash consing)
Since $r$ is not live at $\pi$, it is irrelevant.

\[
[s]_2(o^\#) = [s]_1(o^\#) \cap \text{live-out}(s)
\]
Definition: Given statement $\pi$, dom-vars($\pi$) is the set of all variables whose (unique) definition dominates $\pi$. 
Filtering by Liveness

• Since \( r \) is not live at \( \pi \), it is irrelevant.
• Since \( p \) and \( q \) are live at \( \pi \), their defs must dominate \( \pi \).

\[ [s]^2(o^\#) = [s]^1(o^\#) \cap \text{live-out}(s) \]
\[ [s]^3(o^\#) = [s]^1(o^\#) \cap \text{dom-vars}(s) \]
\[ [s]^1(o^\#) \supseteq [s]^3(o^\#) \supseteq [s]^2(o^\#) \]

**SSA Property 1:**

\( \text{live-out}(s) \subseteq \text{dom-vars}(s) \).
Filtering by Liveness

Since r is not live at π, it is irrelevant.

Since p and q are live at π, their defs must dominate π.

Since defs of p and q dominate π, one must dominate the other.

\[
[s]^2(\#) = [s]^1(\#) \cap \text{live-out}(s)
\]

\[
[s]^3(\#) = [s]^1(\#) \cap \text{dom vars}(s)
\]

\[
[s]^1(\#) \supseteq [s]^3(\#) \supseteq [s]^2(\#)
\]

SSA Property 2:
If \{p_1, p_2, \ldots\} are simultaneously live, then the \(p_i\) are totally ordered by dominance of their definitions.
Fact: $[s]_1(o^\#) \subseteq o^\# \cup \text{def}(s)$ for all $s$.

Therefore, if $o^\# \subseteq \text{dom-vars}(\pi)$, then def of $p$ is dominated by defs of all variables in $o^\#$. Thus, insertion of $p$ occurs at head of list.

**SSA Property 3:**
If a transfer function adds only the variable being defined to a set $S$, it preserves the property that $S \subseteq \text{dom-vars}$.
Removal from Head of List

**Fact:** $[s]^1(o^\#) \supseteq o^\# \setminus \text{def}(s)$ for all $s$.

Thus, if $o^\# \subseteq \text{dom-vars(pred(\pi))}$, then $p \not\in o^\#$. Thus, the $\setminus$ operation in $[s]^1$ is unnecessary.

The only removal necessary is intersection with $\text{dom-vars(\pi)}$.

**SSA Property 4:**
If $S \subseteq \text{dom-vars(pred(\pi))}$, then the variable defined at $\pi$ is not in $S$. 
Removal from Head of List

If \text{pred}(\pi) is the only predecessor of \pi, then \(\text{dom-vars}(\pi) = \text{dom-vars}(\text{pred}(\pi)) \cup \{p\}\).

If \(o^\# \subseteq \text{dom-vars}(\text{pred}(\pi))\), and \([s]^{-1}(o^\#) \subseteq o^\# \cup \{p\}\), then \([s]^{-1}(o^\#) \subseteq \text{dom-vars}(\pi)\).

So no intersection is necessary.
If $\pi$ has multiple predecessors, then $\text{dom-vars}(\pi) = \text{dom-vars}(\text{idom}(\pi)) \cup \{p\}$.

Every var in $o^\# \setminus \text{dom-vars}(\text{idom}(\pi))$ is dominated by every var in $\text{dom-vars}(\text{idom}(\pi))$. Therefore, the variables to be removed are at the head of the list $o^\#$.

Thus, $[\phi]^6(o^#) = [\phi]^1(\text{prune}(o^#))$, where prune removes vars from the head of the list until the def of the head of the list strictly dominates $\pi$.

So intersection is removal from head of list.
SSA Property 5:
To maintain the property that $S \subseteq \text{dom-vars}(\pi)$, it suffices to intersect $S$ with $\text{dom-vars(idom}(\pi))$ only at control flow merge points.

It is convenient to arrange for all control flow merge points to be (possibly vacuous) $\phi$ nodes.
The Alias Set Analysis and the Typestate Analysis are each an instantiation of the IFDS algorithm:
• Interprocedural
• Context-Sensitive
• Precise
• Expensive
Summary of SSA Properties

1. live-out(s) \( \subseteq \) dom-vars(s).

2. If \( \{p_1, p_2, \ldots\} \) are simultaneously live, then the \( p_i \) are totally ordered by dominance of their definitions.

3. If a transfer function adds only the variable being defined to set S, it preserves the property that \( S \subseteq \) dom-vars.

4. If \( S \subseteq \) dom-vars(pred(\( \pi \))), then the variable defined at \( \pi \) is not in S.

5. To maintain the property that \( S \subseteq \) dom-vars(\( \pi \)), it suffices to intersect S with dom-vars(idom(\( \pi \))) only at control flow merge points.