Syntactic and Semantic Soundness of Structural Dataflow Analysis

Patrick Cousot

New York University, Courant Institute of Mathematics, Computer Science
pcousot@cs.nyu.edu  cs.nyu.edu/~pcousot
Soundness of data flow analysis

- In what sense is data flow analysis sound?
  - hide subtleties in the definition of soundness,
  - which may lead to incorrect semantics-based compiler optimizations.
Syntax and trace semantics of programs
Syntax

\[ \begin{align*}
\text{x, y, ...} & \in V \\
A & \in A ::= 1 \mid x \mid A_1 - A_2 \\
B & \in B ::= A_1 < A_2 \mid B_1 \text{ nand } B_2 \\
S & \in S ::= \\
& \quad x = A ; \\
& \quad ; \\
& \quad \text{if}(B) S \mid \text{if}(B) S \text{ else } S \\
& \quad \text{while}(B) S \mid \text{break} ; \\
& \quad \{ Sl \} \\
Sl & \in Sl ::= Sl \ S \mid \epsilon \\
P & \in P ::= Sl \\
S & \in Pc \triangleq S \cup Sl \cup P
\end{align*} \]

variable (\( V \) not empty)
arithmetic expression
boolean expression
statement
assignment
skip
conditionals
iteration and break
compound statement
statement list
program
program component
Program labelling

Unique labelling to designate (sets of) program points:

- **at**$[S]$ the program point at which execution of $S$ starts;
- **after**$[S]$ the program exit point after $S$, at which execution of $S$ is supposed to normally terminate, if ever;
- **escape**$[S]$ a boolean indicating whether or not the program component $S$ contains a `break ;` statement escaping out of that component $S$;
- **break-to**$[S]$ the program point at which execution of the program component $S$ goes to when a `break ;` statement escapes out of that component $S$;
- **breaks-of**$[S]$ the set of labels of all `break ;` statements that can escape out of $S$;
- **in**$[S]$ the set of program points inside $S$ (including at$[S]$ but excluding after$[S]$ and break-to$[S]$);
- **labx**$[S]$ the potentially reachable program points while executing $S$ either at, in, or after the statement, or resulting from a break.
Traces

- Events/action: assignment $x = A = \nu$, true test $B$, false test $\neg(B)$, break, skip
- State: program label $\ell$ (next step to be executed)
- Trace: finite/infinite sequence $\pi \in \mathbb{T}^{+\infty}$ of states separated by events
- Example: $\ell_1 \xrightarrow{x = x + 1 = 1} \ell_2 \xrightarrow{\neg(x < 0)} \ell_4$ (with implicit initialization to 0)
- Trace concatenation: $\cdot$
- Value $\mathcal{q}(\pi)x$ of a variable $x$ at the end of trace $\pi$

$$\begin{align*}
\mathcal{q}(\ell)x & \triangleq 0 & \text{implicit initialization to 0} \\
\mathcal{q}(\pi^\ell x = A = \nu \rightarrow \ell')x & \triangleq \nu \\
\mathcal{q}(\pi^\ell \cdots \rightarrow \ell')x & \triangleq \mathcal{q}(\pi^\ell)x & \text{otherwise}
\end{align*}$$
Prefix trace semantics

- Evaluation of an arithmetic expression
  
  \[ \mathcal{A}[1]_{\rho} \triangleq 1 \]
  
  \[ \mathcal{A}[x]_{\rho} \triangleq \rho(x) \]
  
  \[ \mathcal{A}[A_1 - A_2]_{\rho} \triangleq \mathcal{A}[A_1]_{\rho} - \mathcal{A}[A_2]_{\rho} \]

- Assignment \( s ::= \ell \ x = A \) ; (where at\( [s] = \ell \))
  
  \[ s^*[s] \triangleq \{ \langle \pi_\ell, \ell \rangle, \langle \pi_\ell, \ell \xrightarrow{x=A=v} \text{after}[s] \rangle \mid \pi_\ell \in \mathbb{T}^+ \land v = \mathcal{A}[A]q(\pi_\ell) \} \]
Prefix trace semantics (cont’d)

- Break statement $S ::= \ell \text{ break ; }$ (where $\text{at}[S] = \ell$)

$$S^*[S] \triangleq \{ \langle \pi\ell, \ell \rangle, \langle \pi\ell, \ell \xrightarrow{\text{break}} \text{break-to}[S] \rangle \mid \pi\ell \in \mathbb{T}^+ \}$$  \hspace{1cm} (5)
Prefix trace semantics (cont’d)

- Conditional statement $S ::= \text{if } \ell (B) S_t$ (where $\text{at}[S] = \ell$)

$$S^*[S] \triangleq \{ \langle \pi_1^\ell, \ell \rangle | \pi_1^\ell \in \mathbb{T}^+ \}$$

$$\cup \{ \langle \pi_1^\ell, \ell \xrightarrow{\neg(B)} \text{after}[S] \rangle | \mathcal{B}[B]q(\pi_1^\ell) = \text{ff} \land \pi_1^\ell \in \mathbb{T}^+ \}$$

$$\cup \{ \langle \pi_1^\ell, \ell \xrightarrow{B} \text{at}[S_t] \leadsto \pi_2 \rangle | \mathcal{B}[B]q(\pi_1^\ell) = \text{tt} \land \langle \pi_1^\ell \xrightarrow{B} \text{at}[S_t], \pi_2 \rangle \in S^*[S_t] \}$$
Prefix trace semantics (cont’d)

- Statement list \( S_l ::= S_l' S \) (where \( at[S] = after[S_l'] \))
  \[
  S^*[S_l] \triangleq S^*[S_l'] \\
  \cup \{ \langle \pi_1, \pi_2 \cdot \pi_3 \rangle \mid \langle \pi_1, \pi_2 \rangle \in S^*[S_l'] \land \langle \pi_1 \cdot \pi_2, \pi_3 \rangle \in S^*[S] \}
  \]

- Empty statement list \( S_l ::= \epsilon \) (where \( at[S_l] \triangleq after[S_l] \))
  \[
  S^*[S_l] \triangleq \{ \langle \pi at[S_l], at[S_l] \rangle \mid \pi at[S_l] \in T^+ \}
  \]
Prefix trace semantics (cont’d)

- Iteration statement $S ::= \text{while } \ell (B) S_b$ (where $\text{at}[S] = \ell$)

\[ S^*[S] = \text{lfp}^c F^*[S] \]

\[ F^*[\text{while } \ell (B) S_b](X) \overset{\Delta}{=} \{ \langle \pi_1 \ell, \ell \rangle | \pi_1 \ell \in T^+ \} \]  
(a)

\[ \cup \{ \langle \pi_1 \ell, \ell \pi_2 \ell \overset{\neg(B)}{\rightarrow} \text{after}[S] \rangle | \langle \pi_1 \ell, \ell \pi_2 \ell \rangle \in X \land B[B] q(\pi_1 \ell \pi_2 \ell) = \text{ff} \} \]  
(b)

\[ \cup \{ \langle \pi_1 \ell, \ell \pi_2 \ell \overset{B}{\rightarrow} \text{at}[S_b] \overset{\pi_3}{\cdot} \rangle | \langle \pi_1 \ell, \ell \pi_2 \ell \rangle \in X \land \\
B[B] q(\pi_1 \ell \pi_2 \ell) = \text{tt} \land \langle \pi_1 \ell \pi_2 \ell \overset{B}{\rightarrow} \text{at}[S_b], \pi_3 \rangle \in S^*[S_b] \} \]  
(c)
Maximal trace semantics

- Maximal trace semantics

\[ S^+[S] \triangleq \{ \langle \pi_1, \pi_2\ell \rangle \in S^*[S] \mid (\ell = \text{after}[S]) \lor (\text{escape}[S] \land \ell = \text{break-to}[S]) \} \]  

(11)

\[ S^\infty[S] \triangleq \lim(S^*[S]) \]

(12)

- Limit

\[ \lim \mathcal{T} \triangleq \{ \langle \pi, \pi' \rangle \mid \pi' \in \mathcal{T}^\infty \land \forall n \in \mathbb{N} \cdot \langle \pi, \pi'[0..n] \rangle \in \mathcal{T} \}. \]

(13)
Live variables analysis
[Kennedy, 1975, 1976a,b]
Parameterized live variable abstraction on a trace

\[ \alpha^l_{use, mod}[S] \, L_b, L_e \, \langle \pi_0, \pi \rangle \]

- After initialization \( \pi_0 \), execution of component \( S \) may continue with \( \pi \)
- \( L_b \) live variables if \( S \) escapes with a break
- \( L_e \) live variables if \( S \) terminates
- \( use \) defining the set \( use[a] \, \rho \) of variables which value is used when executing action \( a \) in environment \( \rho \);
- \( mod \) defining the set \( mod[a] \, \rho \) of variables which value is modified when executing action \( a \) in environment \( \rho \).

\[ \alpha^l_{use,mod}[S] \, L_b, L_e \, \langle \pi_0, \pi \rangle \] is the set of live variable at\([S]\) for execution \( \pi \) initialized by \( \pi_0 \)
Parameterized live variable analysis
Parameterized definition of the live variable abstraction of a trace

\[
\alpha^l_{\text{use,mod}} \big[ S \big] \ L_b, L_e \langle \pi_0, \ell \rangle \triangleq \{ x \in V \mid (\ell = \text{after} \big[ S \big] \land x \in L_e) \lor (\text{escape} \big[ S \big] \land \ell = \text{break-to} \big[ S \big] \land x \in L_b) \} 
\]

\[
\alpha^l_{\text{use,mod}} \big[ S \big] \ L_b, L_e \langle \pi_0, \ell \xrightarrow{a} \ell', \pi_1 \rangle \triangleq \{ x \in V \mid x \in \text{use} \big[ a \big] q(\pi_0) \lor (x \notin \text{mod} \big[ a \big] q(\pi_0) \land x \in \alpha^l_{\text{use,mod}} \big[ S \big] \ L_b, L_e \langle \pi_0 \xrightarrow{a} \ell', \ell', \pi_1 \rangle) \} 
\]

A variable is live at some point if it holds a value that may be needed in the future, or equivalently if its value may be read before the next time the variable is written to.

https://en.wikipedia.org/wiki/Live_variable_analysis

- may be → potential liveness, is on one trace
- or equivalently → wrong
Parameterized definition of the live variable abstraction of a trace

Lemma 1 If $\pi_1 = \ell_1 \xrightarrow{a_1} \ell_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} \ell_n$ and $\langle \pi_0, \pi_1 \rangle \in \mathcal{S}^* [S]$ then

$$\alpha^l_{use,mod}[S] L_b, L_e \langle \pi_0, \pi_1 \rangle = \{ x \in V \mid \exists i \in [1, n-1]. \forall j \in [1, i-1].
\}
\cup \left( \ell_n = after[S] ? L_e : \emptyset \right) \cup \left( escape[S] \land \ell_n = break-to[S] ? L_b : \emptyset \right). \qed$$
Parameterized definition of the live variable abstraction of a trace semantics

- **Liveness**

\[
\alpha^l_{use,mod}[S] \triangleq L_b, L_e = \bigcup_{\langle \pi_0, \pi \rangle \in S} \alpha^l_{use,mod}[S] L_b, L_e \langle \pi_0, \pi \rangle
\]

potential liveness \((15)\)

\[
\alpha^l_{use,mod}[S] \triangleq L_b, L_e = \bigcap_{\langle \pi_0, \pi \rangle \in S} \alpha^l_{use,mod}[S] L_b, L_e \langle \pi_0, \pi \rangle
\]

definite liveness \((16)\)

- **Deadness** is defined dually

\[
\alpha^d_{use,mod}[S] \triangleq D_b, D_e = -\alpha^l_{use,mod}[S] \triangleq -D_b, -D_e
\]

potential deadness \((1)\)

\[
\alpha^d_{use,mod}[S] \triangleq D_b, D_e = -\alpha^l_{use,mod}[S] \triangleq -D_b, -D_e
\]

definite deadness \((2)\)
Parameterized definition of the live variable abstraction of a trace semantics

Lemma 2 $\alpha_{use,mod}^{\exists l} [s] (\mathcal{S}^{+\infty} [s]) = \alpha_{use,mod}^{\exists l} [s] (\mathcal{S}^* [s])$. \hfill \square
Instances of the parameterized live variable analysis
Semantic liveness/deadness

- **use**
  
  \[
  \text{use}[	ext{skip}] \rho \triangleq \emptyset \\
  \text{use}[x = A] \rho \triangleq \{ y \mid \exists v \in V . \ A[A] \rho \neq A[A] \rho[y \leftarrow v] \land \rho(x) \neq A[A] \rho \} \\
  \text{use}[a] \rho \triangleq \{ y \mid \exists v \in V . \ B[a] \rho \neq B[a] \rho[y \leftarrow v] \} \quad a \in \{B, \neg(B)\}
  \]

- **mod**
  
  \[
  \text{mod}[a] \rho \triangleq \{ x \mid a = (x = A) \land (\rho(x) \neq A[A] \rho) \}
  \]

- **Semantic potential liveness**
  
  \[
  S^{\exists l}[S] \triangleq \alpha_{\text{use}, \text{mod}}[S] (S^{+\infty}[S])
  \]
Classical syntactic liveness/deadness

- **use**

\[
\text{use}[x = A] \rho \triangleq \text{vars}[A] \\
\text{use}[\text{skip}] \rho \triangleq \emptyset \\
\text{use}[B] \rho \triangleq \text{use}[-(B)] \rho \triangleq \text{vars}[B]
\]

(\(\rho\) is useless)

- **mod**

\[
\text{mod}[x = A] \rho \triangleq \{x\} \\
\text{mod}[\text{skip}] \rho \triangleq \emptyset \\
\text{mod}[B] \rho \triangleq \text{mod}[-(B)] \rho \triangleq \emptyset
\]

- **Classical syntactic potential liveness**

\[
\mathcal{S}^{\exists l}[S] \triangleq \alpha^{\exists l}_{\text{use,mod}}[S] (\mathcal{S}^{\infty}[S])
\]

(22)
Soundness expectation

- Soundness of potential liveness

\[ S^\exists[\mathcal{S}] \subseteq S^\forall[\mathcal{S}] \]
Soundness expectation

- Soundness of potential liveness

\[ \mathcal{S}^\text{liveness}(s) \subseteq \mathcal{S}^\text{potential liveness}(s) \]

- **This is not true!**
Soundness expectation

- Soundness of potential liveness

\[ \mathcal{S}^\exists_l[S] \subseteq \mathcal{S}^\exists_l[S] \]

- **This is not true!**

- Problem

\[ \exists a . \exists \rho \in \text{Ev} . x \in \text{mod}[a] \rho \land x \notin \text{mod}[a] \rho \]
Soundness expectation

- Soundness of potential liveness

\[ \mathcal{S}^{\exists_l}[S] \subseteq \mathcal{S}^{\exists_l}[S] \]

- This is not true!

- Problem

\[ \exists a . \exists \rho \in \text{Ev} . x \in \text{mod}[a] \rho \land x \notin \text{mod}[a] \rho \]

- Counter-example

\[ x = x; \]

If the compiler eliminates that assignment, this changes syntactic liveness (but not semantic liveness). For soundness, the syntactic liveness analysis must be redone after useless assignment elimination.
How to fix the problem?

- Change the live variable algorithm to be sound with respect to the semantic definition
  - this becomes a liveness/eventuality problem, requires variant functions, etc.
  - too complicated for compilers!
- Keep the live variable algorithm, but change the notion of soundness
  - this limits compiler optimizations, or
    requires a recomputation of the live variable information after the program transformation
  - less complicated for compilers (which may even be incorrect if the live variable analysis is not redone after program transformation)
Restating soundness

- Define

\[ \mathcal{S}^\exists [S] \triangleq \alpha^{\exists}_{\text{use,mod}} (\mathcal{S}^{+\infty} [S]) \]  

(24)

**Theorem 1** If \( \alpha^{\exists}_{\text{use,mod}}[S] (\mathcal{S}^{+\infty} [S]) \preceq \mathcal{S}^\exists [S] \) then \( \mathcal{S}^\exists [S] \preceq \mathcal{S}^\exists [S] \).

- Follows from

\[ \exists \rho \in \mathcal{Ev} \cdot y \in \text{use}[a] \rho \Rightarrow \forall \rho \in \mathcal{Ev} \cdot y \in \text{use}[a] \rho \]  

(23)

- Intuition

A variable is live at some point if it holds a value that may be necessarily used before the next time the variable is assigned to.
Calculational design of the structural syntactic potential liveness static analysis
Calculational design

- Based on the soundness definition

\[ \alpha_{\text{use,mod}}^{\exists l}[S] (\mathcal{S}^* [S]) \subseteq \mathcal{S}^{\exists l} [S] \]

- Method
  - by structural induction on program components $S$
  - develop $\alpha_{\text{use,mod}}^{\exists l}[S] (\mathcal{S}^* [S])$ to eliminate the abstraction $\alpha_{\text{use,mod}}^{\exists l}[S]$
  - over-approximate to eliminate all concrete computations (e.g. value of a test with dead branch)
Assignment $S ::= \ell \ x = A$ ;

\[
\begin{align*}
S \trianglerighteq [S] & L_b, L_e \\
= & \alpha_{use, mod} [S] (S^* [S]) L_b, L_e \\
= & \bigcup \{ \alpha'_{use, mod} [S] L_b, L_e \langle \pi_0, \pi_1 \rangle \mid \langle \pi_0, \pi_1 \rangle \in S^* [S] \} \\
= & \bigcup \{ \alpha'_{use, mod} [S] L_b, L_e \langle \pi_0 \text{ at } [S] \rangle \} \cup \bigcup \{ \alpha'_{use, mod} [S] L_b, L_e \langle \pi_0 \text{ at } [S] \rangle \} \\
= & \bigcup \{ \alpha'_{use, mod} [S] L_b, L_e \langle \pi_0 \text{ at } [S] \rangle \} \cup \{ y \in \forall \mid y \in \text{use}[x = A] \}[\ell A](\pi_0 \text{ at } [S]) \lor (y \notin \text{mod}[x = A] \)[\ell A](\pi_0 \text{ at } [S]) \land y \in \alpha'_{use, mod} [S] L_b, L_e \langle \pi_0 \text{ at } [S] \rangle \} \\
= & \{ y \in \forall \mid y \in \text{use}[x = A] \lor (y \notin \text{mod}[x = A] \land y \in L_e) \} \\
= & \text{use}[x = A] \cup (L_e \setminus \text{mod}[x = A]) \\
\triangleq & S \trianglerighteq [x = A] L_b, L_e \\
\trianglerighteq & \ell \ x = A ; \end{align*}
\]
Potentially live variables

**Structural syntactic potential liveness analysis**

\[
\begin{align*}
\hat{\mathcal{S}} \triangleright \triangleright [sl \; \ell] \; L_e & \triangleq \hat{\mathcal{S}} \triangleright \triangleright [sl \; \ell] \; \emptyset, L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [x = A \; ;] \; L_b, L_e & \triangleq \text{use}[x = A] \cup (L_e \setminus \text{mod}[x = A]) \\
\hat{\mathcal{S}} \triangleright \triangleright [\;] \; L_b, L_e & \triangleq L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [sl' \; S] \; L_b, L_e & \triangleq \hat{\mathcal{S}} \triangleright \triangleright [sl'] \; L_b, (\hat{\mathcal{S}} \triangleright \triangleright [S] \; L_b, L_e) \\
\hat{\mathcal{S}} \triangleright \triangleright [\; e] \; L_b, L_e & \triangleq L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [\text{if (B) S}_t] \; L_b, L_e & \triangleq \text{use}[B] \cup L_e \cup \hat{\mathcal{S}} \triangleright \triangleright [S_t] \; L_b, L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [\text{if (B) S}_t \; \text{else S}_f] \; L_b, L_e & \triangleq \text{use}[B] \cup \hat{\mathcal{S}} \triangleright \triangleright [S_t] \; L_b, L_e \cup \hat{\mathcal{S}} \triangleright \triangleright [S_f] \; L_b, L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [\text{while (B) S}_b] \; L_b, L_e & \triangleq \text{use}[B] \cup L_e \cup \hat{\mathcal{S}} \triangleright \triangleright [S_b] \; L_b, L_e \\
\hat{\mathcal{S}} \triangleright \triangleright [\text{break ;}] \; L_b, L_e & \triangleq L_b \\
\hat{\mathcal{S}} \triangleright \triangleright [\{ sl \}] \; L_b, L_e & \triangleq \hat{\mathcal{S}} \triangleright \triangleright [sl] \; L_b, L_e
\end{align*}
\]
A surprise

The fixpoint in the structural syntactic potential liveness analysis of the iteration
while $(B) S_b$ is a constant.
Definitely dead variables

**Structural syntactic definite deadness analysis**

\[
\begin{align*}
\mathcal{S}^{\forall d}[sl \ell] D_e &= \mathcal{S}^{\forall d}[sl \ell] V, D_e \\
\mathcal{S}^{\forall d}[x = A ;] D_b, D_e &= \neg \text{use}[x = A] \cap (D_e \cup \text{mod}[x = A]) \\
\mathcal{S}^{\forall d}[;] D_b, D_e &= D_e \\
\mathcal{S}^{\forall d}[sl' s] D_b, D_e &= \mathcal{S}^{\forall d}[sl'] D_b, (\mathcal{S}^{\forall d}[s] D_b, D_e) \\
\mathcal{S}^{\forall d}[\epsilon] D_b, D_e &= D_e \\
\mathcal{S}^{\forall d}[\text{if (B) } S_t] D_b, D_e &= \neg \text{use}[B] \cap D_e \cap \mathcal{S}^{\forall d}[s_t] D_b, D_e \\
\mathcal{S}^{\forall d}[\text{if (B) } S_t \text{ else } S_f] D_b, D_e &= \neg \text{use}[B] \cap \mathcal{S}^{\forall d}[s_t] D_b, D_e \cap \mathcal{S}^{\forall d}[s_f] D_b, D_e \\
\mathcal{S}^{\forall d}[\text{while (B) } S_b] D_b, D_e &= \neg \text{use}[B] \cap D_e \cap \mathcal{S}^{\forall d}[s_b] D_b, D_e \\
\mathcal{S}^{\forall d}[\text{break ;}] D_b, D_e &= D_b \\
\mathcal{S}^{\forall d}[\{ sl \}] D_b, D_e &= \mathcal{S}^{\forall d}[sl] D_b, D_e
\end{align*}
\]
Conclusion
Conclusion

- Classical definitions of the soundness of data flow analyses [Beyer, Gulwani, and Schmidt, 2018; Kildall, 1973; Schmidt, 1998; Steffen, 1991, 1993] are specified with respect to an abstraction of the semantics not the semantics itself, which is confusing.

- Transition systems forget about the program structure\(^1\) so lead to iterations that may be useless.

---

Conclusion

- Classical definitions of the soundness of data flow analyses [Beyer, Gulwani, and Schmidt, 2018; Kildall, 1973; Schmidt, 1998; Steffen, 1991, 1993] are specified with respect to an abstraction of the semantics not the semantics itself, which is confusing.

- Transition systems forget about the program structure\(^1\) so lead to iterations that may be useless.

- Why CompCert get it right?
  - does simultaneously the liveness analysis and the program transformation based on this analysis
  - returns the result of the liveness analysis valid after the transformation
  - justifies by dependency: a variable is dead if nothing later depends on its value

---

Conclusion

- Anonymous reviewer

“It is an old story that the dataflow analysis framework ("syntactic" dataflow analysis in paper's characterization) is way too weak. For modern programming languages, control flow is not syntactic but a part of semantics. Dataflow analysis assumes the control flow to be available before the analysis hence a stalemate for modern languages with higher order functions, dynamic bindings, or dynamic gotos; dataflow analysis has neither a systematic guide to prove the correctness of an analysis nor systematic approach to manage the precision of the analysis. On the other hand, the semantics-based design theory (abstract interpretation) is general enough to handle any kind of source languages and powerful enough to prove the correctness and to manage its precision.”
Bibliography
References


References II

The End, Thank you