Abstract Semantic Dependency

Patrick Cousot

New York University, Courant Institute of Mathematics, Computer Science
pcousot@cs.nyu.edu    cs.nyu.edu/~pcousot
Objective
Objective

- Design a dependency analysis by abstract interpretation of a trace semantics.
- \( a \) depends on \( b \) iff changing \( b \) into a different \( b' \) will change \( a \) into a different \( a' \)
- This involves 2 execution traces \( a \rightarrow b \) and \( a' \rightarrow b' \) (i.e. it is not a trace abstraction)
Objective

- Design a dependency analysis by abstract interpretation of a trace semantics.
- \( a \) depends on \( b \) iff changing \( b \) into a different \( b' \) will change \( a \) into a different \( a' \)
- This involves 2 execution traces \( a \to b \) and \( a' \to b' \) (i.e. it is not a trace abstraction)
- Recent work (Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel and Caterina Urban and Peter Müller) suggests abstract interpretation theory must be revisited
Objective

- Design a dependency analysis by abstract interpretation of a trace semantics.
- \( a \) depends on \( b \) iff changing \( b \) into a different \( b' \) will change \( a \) into a different \( a' \)
- This involves 2 execution traces \( a \rightarrow b \) and \( a' \rightarrow b' \) (i.e. it is not a trace abstraction)
- Recent work (Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel and Caterina Urban and Peter Müller) suggests abstract interpretation theory must be revisited
- or not?
Syntax and trace semantics
Syntax and trace semantics

- The syntax is a subset of \( C \) (while programs)
- The semantics is a structural prefix (or maximal) trace semantics \( \langle \pi^{\ell}, \ell \pi' \rangle \in S^* \) (where \( \ell = \text{at}[S] \)) means that an execution reaching the entry point \( \ell \) of program component \( S \) may continue as stated by \( \ell \pi' \).
- Example: Assignment \( S ::= \ell x = A ; \) (where \( \text{at}[S] = \ell \))

\[
S^*[S] \triangleq \{ \langle \pi^{\ell}, \ell \rangle, \langle \pi^{\ell}, \ell \xrightarrow{x=A=\nu} \text{after}[S] \rangle \mid \pi^{\ell} \in T^+ \land \nu = A[J][Q](\pi^{\ell}) \} \quad (0)
\]

\[
S^+[S] \triangleq \{ \langle \pi^{\ell}, \ell \xrightarrow{x=A=\nu} \text{after}[S] \rangle \mid \pi^{\ell} \in T^+ \land \nu = A[J][Q](\pi^{\ell}) \}
\]

\[
S^\infty[S] \triangleq \emptyset \quad \text{no infinite trace}
\]
Informal Requirements for a Semantic Definition of Dependency
Informal Requirements for a Semantic Definition of Dependency

- For simplicity, we consider dependency upon initial states.
- The dependency of variables on initial states is local, at each program point (not global as in [D. E. Denning and P. J. Denning, 1977] or on program exit as in [Assaf, Naumann, Signoles, Totel, and Tronel, 2017; Urban and Müller, 2018]).
- We don't want to make a difference between control and data dependency (as in [D. E. Denning and P. J. Denning, 1977] and their followers).
- We ignore timing channels (as usual in compilation).
- We ignore empty observations (observing nothing at a program point is not an observation).
Formal Semantic Definition of Dependency
Sequence of values of a variable at a program point

- $\text{seqval}[y]^{\ell}(\pi_0, \pi)$ is the sequence of values of the variable $y$ at program point $\ell$ along the trace $\pi$ continuing $\pi_0$

$$
\text{seqval}[y]^{\ell}(\pi_0, \ell) \triangleq q(\pi_0)y
$$

$$
\text{seqval}[y]^{\ell}(\pi_0, \ell') \triangleq \emptyset \quad \text{when} \quad \ell' \neq \ell
$$

$$
\text{seqval}[y]^{\ell}(\pi_0, \ell \xrightarrow{a} \ell'' \pi) \triangleq q(\pi_0)y \cdot \text{seqval}[y]^{\ell}(\pi_0 \circ \ell \xrightarrow{a} \ell'', \ell'' \pi)
$$

$$
\text{seqval}[y]^{\ell}(\pi_0, \ell' \xrightarrow{a} \ell'' \pi) \triangleq \text{seqval}[y]^{\ell}(\pi_0 \circ \ell' \xrightarrow{a} \ell'', \ell'' \pi) \quad \text{when} \quad \ell' \neq \ell
$$

- (bi-induction: induction for finite traces, co-induction for infinite ones)
Differences between sequences of values of a variable at a program point

- $\text{diff}(\omega, \omega')$ holds if and only if the sequences of value observations $\omega$ and $\omega'$ at some program point differ by at least one value

\[
\text{diff}(\omega, \omega') \triangleq \exists \omega_0, \omega_1, \nu, \nu'. \omega = \omega_0 \cdot \nu \cdot \omega_1 \land \omega' = \omega_0 \cdot \nu' \cdot \omega_1' \land \nu \neq \nu'
\]  \hspace{1cm} (2)

- $\neg\text{diff}(\omega, \omega')$ implies
  - either that $\omega = \omega'$ (no dependency for same futures)
  - or one is a strict prefix of the other (timing channels are abstracted away).

- Change this definition to get alternative concepts of dependency (e.g. timing channels, empty observation, etc.)
Definition of value dependency

- $\Pi \in \wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty})$ is a trace semantics
- Properties are represented by sets (of individuals with this property)
- $\Pi \in D^\ell(x, y)$ means that $y$ at $\ell$ depends on the initial value of $x$

**Definition 1 (Dependency $D$)**

$$D^\ell(x, y) \triangleq \{\Pi \in \wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty}) \mid \exists \langle \pi_0, \pi_1 \rangle, \langle \pi'_0, \pi'_1 \rangle \in \Pi \cdot$$

$$\forall z \in V \setminus \{x\}. \varrho(\pi_0)z = \varrho(\pi'_0)z \land$$

$$\text{diff(seqval}_\ell y][\pi_0, \pi_1], \text{seqval}_\ell y][\pi'_0, \pi'_1])\}$$

□
Value dependency flow

- $x \succeq_\ell P y$ iff, at program point $\ell$ of program $P$, variable $y$ depends on the initial value of variable $x$ (or the initial value of variable $x$ flows to variable $y$ at program point $\ell$)

**Definition 2 (Value dependency flow)**

$$x \succeq_\ell P y \triangleq (\mathcal{S}^{+\infty}[P] \in \mathcal{D}_\ell(x, y)).$$  \hspace{1cm} (4)

- The use of the prefix trace semantics $\mathcal{S}^*[P]$ is equivalent to that of the maximal trace semantics $\mathcal{S}^{+\infty}[P]$

**Lemma 1 (Value dependency for finite prefix traces)**

$$x \succeq_\ell P y = (\mathcal{S}^*[P] \in \mathcal{D}_\ell(x, y)).$$  \hspace{1cm} □
Value dependency abstraction

- $\alpha^d(S)$ is the value dependency abstraction of a semantic property $S \in \wp(\wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty}))$ is

\[
\alpha^d(S) \triangleq \{\langle x, y \rangle \mid S \subseteq D^\ell \langle x, y \rangle\}
\]  \(5 \)

- This a Galois connection $\langle \wp(\wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty})), \subseteq \rangle \leftrightarrow \langle \wp^d, \supseteq^d \rangle$ where

$\wp^d \triangleq L \rightarrow \wp(\wp(\wp(\mathbb{V} \times \mathbb{V})))$ is ordered pointwise

\[
\ell \mapsto \{\langle x, y \rangle \mid x \leadsto^\ell P y\} = \alpha^d(\{S^+\supseteq [P]\}) = \alpha^d(\{S^*\supseteq [P]\})
\]
Exact, definite, and potential value dependency semantics

\[
\begin{align*}
\mathcal{S}_{\text{diff}}[S] & \triangleq \alpha^d(\{\mathcal{S}^+\infty[S]\}) = \alpha^d(\{\mathcal{S}^*[S]\}) \quad \text{exact dependency} \\
\mathcal{S}_{\text{diff}}^\forall[S] & \triangleq \alpha^d(\{\mathcal{S}^+\infty[S]\}) \quad \text{definite dependency} \\
\alpha^d(\{\mathcal{S}^+\infty[S]\}) & \preceq \mathcal{S}_{\text{diff}}^\exists[S] \quad \text{potential dependency} \quad (6)
\end{align*}
\]
Calculational design of the structural potential dependency analysis
Calculational design

- Based on the soundness definition

\[ \alpha^d(\{S^*\llbracket S \rrbracket\}) \subseteq \widehat{S}_{\text{diff}}^{\exists} \llbracket S \rrbracket \]

- The finite abstract domain is \( L \rightarrow \wp (V \times V) \) ordered pointwise

- Method
  - by structural induction on program components \( S \)
  - develop \( \alpha^d(\{S^*\llbracket S \rrbracket\}) \) to eliminate the abstraction \( \alpha^d \)
  - over-approximate to eliminate all concrete computations (e.g. value of a test with dead branch)

- A bit more complicated than for DFA since for each program component \( S \), we have to consider any two execution traces of \( S \) (only one for DFA)
Structural static potential value dependency analysis

- assignment $S ::= x = A$ ;

\[
\begin{align*}
\widehat{S}_{\text{diff}}^\exists [S] \ell & \triangleq (\ell = \text{at}[S] \not\in \forall \\
& \cup (\ell = \text{after}[S] \not\in \forall \{\langle y, x \rangle \mid y \in \widehat{S}_{\text{diff}}^\exists [A] \} \cup \{\langle y, y \rangle \mid y \neq x\} \\
\& \subseteq \emptyset ) \\
\widehat{S}_{\text{diff}}^\exists [A] & \triangleq \{y \mid \exists \rho \in \mathbb{E}v . \exists \nu \in \mathbb{V} . \mathcal{A}[A] \rho \neq \mathcal{A}[A] \rho[y \leftarrow \nu] \subseteq \text{vars}[A]\}
\end{align*}
\]
Proof of (10) We consider the case $\ell = \text{after}[S]$. (The cases $\ell = \text{at}[S]$ and $\ell \notin \text{labx}[S]$ are simpler.)

$$\alpha^d(\{S^+ \in \text{at}[S]\}) \text{ after}[S]$$

$$= \alpha^d(\{S^* \in \text{at}[S]\}) \text{ after}[S]$$

$$= \{ \langle x', y \rangle | S^* \in D(\text{after}[S]) \langle x', y \rangle \}$$

$$\text{proof of (5) of } \alpha^d \text{ and def. } \subseteq$$

$$= \{ \langle x', y \rangle | \exists \langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle \in S^* \text{ after}[S] \exists z \in \forall \setminus \{x'\} . q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) = q(\pi'_0)z \land \text{diff}(\text{seqval}[y](\text{after}[S]))(\pi_0, \pi_1)$$

$$\text{def. of prefix finite trace semantics}$$

$$= \{ \langle x', y \rangle | \exists \langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle \in \langle \text{at}[S], \text{at}[S] \rangle \} \xrightarrow{x = A[q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) \text{ after}[S]]} \text{ after}[S] \} . \forall z \in \forall \setminus \{x'\} . q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) = q(\pi'_0)z \land \text{seqval}[y](\text{after}[S])(\pi_0, \pi_1)$$

$$\text{def. of prefix finite trace semantics}$$

$$= \{ \langle x', y \rangle | \exists \langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle \in \langle \text{at}[S], \text{at}[S] \rangle \} \xrightarrow{x = A[q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) \text{ after}[S]]} \text{ after}[S] \} . \forall z \in \forall \setminus \{x'\} . q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) = q(\pi'_0)z \land \text{seqval}[y](\text{after}[S])(\pi_0, \pi_1)$$

$$\text{def. of prefix finite trace semantics}$$

$$= \{ \langle x', y \rangle | \exists \langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle \in \langle \text{at}[S], \text{at}[S] \rangle \} \xrightarrow{x = A[q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) \text{ after}[S]]} \text{ after}[S] \} . \forall z \in \forall \setminus \{x'\} . q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) = q(\pi'_0)z \land \text{seqval}[y](\text{after}[S])(\pi_0, \pi_1)$$

$$\text{def. of prefix finite trace semantics}$$

$$= \{ \langle x', y \rangle | \exists \langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle \in \langle \text{at}[S], \text{at}[S] \rangle \} \xrightarrow{x = A[q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) \text{ after}[S]]} \text{ after}[S] \} . \forall z \in \forall \setminus \{x'\} . q(\langle \pi_0, \pi_1, \pi'_0, \pi'_1 \rangle) = q(\pi'_0)z \land \text{seqval}[y](\text{after}[S])(\pi_0, \pi_1)$$

$$\text{def. of prefix finite trace semantics}$$
\[
\begin{align*}
\{ (x', y) \mid & \exists (\pi_0 at[S], at[S]) \quad x = A[\pi_0 at[S]] \rightarrow \text{after}[S]), \langle \pi_0 at[S], at[S] \rangle \quad x = A[\pi_0 at[S]] \rightarrow \text{after}[S]) \cdot (\forall z \in V \setminus \{x'\} \cdot q(\pi_0 at[S])z = q(\pi_0 at[S])z) \wedge ((q(\pi_0 at[S]))y \neq q(\pi_0 at[S]))y \vee (q(\pi_0 at[S]))y = q(\pi_0 at[S]))y \wedge q(\pi_0 at[S]) \rightarrow \text{after}[S])y \neq q(\pi_0 at[S]) \rightarrow \text{after}[S])(y) \}\end{align*}
\]
\[\text{def. } q\]
\[
\{ (x', y) \mid ((y = x') \vee (y = x \wedge x \in \mathcal{A}[\pi_0 at[S]] \neq \mathcal{A}[\pi_0 at[S]]))\}
\]
\[\text{letting } \rho = q(\pi_0 at[S])\text{ and } \nu = q(\pi_0 at[S])(x') \text{ so that } \forall z \in V \setminus \{x'\} \cdot q(\pi_0 at[S])z = q(\pi_0 at[S])z \text{ implies that } q(\pi_0 at[S]) = \rho[x' \leftarrow \nu].\]
\[
\{ (x', x') \mid x' \neq x \} \cup \{ (x', x) \mid \exists \rho, \nu \cdot \mathcal{A}[\pi_0 at[S]] \neq \mathcal{A}[\pi_0 at[S]]\rho[x' \leftarrow \nu]\}
\]
\[\text{case analysis}\]
\[
\{ (x', x') \mid x' \neq x \} \cup \{ (x', x) \mid x' \in \mathcal{S}_{\text{diff}}^3[\mathcal{A}]\}
\]
\[\text{by defining the functional dependency of an expression } A \text{ as } \mathcal{S}_{\text{diff}}^3[\mathcal{A}] \triangleq \{ x' \mid \exists \rho, \nu \cdot \mathcal{A}[\pi_0 at[S]] \neq \mathcal{A}[\pi_0 at[S]]\rho[x' \leftarrow \nu]\} \text{ in (10)}\]
\[\square\]
Determinacy

- If variables in \( x \in \text{det}(B_1, B_2) \) have different values then \( B_1 \) and \( B_2 \) cannot both be true. 
  
i.e. if \( B_1 \) and \( B_2 \) are both true then the values of variables \( x \in \text{det}(B_1, B_2) \) are the same

\[
\text{det}(B_1, B_2) \subseteq \{ x | \forall \rho, \rho'. (B_1^\rho \land B_2^{\rho'}) \Rightarrow (\rho(x) = \rho'(x)) \} \tag{13}
\]

e.g. \( \text{det}(x=1, x=1 \land y=42) = \{x\} \)

- The values of variables in \( \text{det}(B, B) \) are determined by the veracity of \( B \)

\[
\text{det}(B, B) \subseteq \{ x | \forall \rho, \rho'. (B^\rho \land B^{\rho'}) \Rightarrow (\rho(x) = \rho'(x)) \}
\]

e.g. \( \text{det}(x=y \land z=42, x=y \land z=42) = \{z\} \)
Non-determinacy:

- Variables in $x \in \text{nondet}(B_1, B_2)$ do not change the veracity of $B_1$ and $B_2$

  $$\text{nondet}(B_1, B_2) \supseteq \forall \setminus \text{det}(B_1, B_2)$$
  $$\supseteq \{x \mid \exists \rho, \rho'. \mathcal{B}[B_1] \rho \land \mathcal{B}[B_2] \rho' \land \rho(x) \neq \rho'(x)\}$$

  e.g. $\text{nondet}(x=1, x=1 \land y=42) = \{y\}$

- The values of variables in $x \in \text{nondet}(B, B)$ are not determined by the veracity of $B$

  $$\text{nondet}(B, B) \supseteq \{x \mid \exists \rho, \rho'. \mathcal{B}[B] \rho \land \mathcal{B}[B] \rho' \land \rho(x) \neq \rho'(x)\}$$

  e.g. $\text{det}(x=y \land z=42, x=y \land z=42) = \{x, y\}$
Structural static potential value dependency analysis (cont’d)

- conditional $S ::= \text{if } (B) \ S_t$

$$\widehat{\mathcal{S}}_{\text{diff}}^3 [S] \ell \triangleq (\ell = \text{at}[S] \ ? 1_¥$$

$$\ | \ell \in \text{in}[S_t] \ ? \widehat{\mathcal{S}}_{\text{diff}}^3 [S_t] \ell | \ \text{nondet}(B, B)$$

$$\ | \ell = \text{after}[S] \ ? \widehat{\mathcal{S}}_{\text{diff}}^3 [S_t] \ \text{after}[S_t] \ | \ \text{nondet}(B, B)$$

$$\bigcup \ 1_¥ | \ \text{nondet}(\neg B, \neg B)$$

$$\bigcup \ \text{nondet}(\neg B, \neg B) \times \text{mod}[S_t]$$

$$\vdots \emptyset$$

$mod[S_t]$ is the set of variables that may be modified by $S_t$

---

1) is left restriction

"Abstract Semantic Dependency" – 22/40 –
Example

- \( S ::= \ell \; L = H \; \ell' \)
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S] \; \ell = \{ \langle L, L \rangle, \langle H, H \rangle \} \]
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S] \; \ell' = \{ \langle H, L \rangle \} \cup \{ \langle H, H \rangle \}. \]

- \( S' ::= \{ \text{if} \; \ell_1 (H) \; \ell_2 \; L = H \; \ell_3 \; \text{else} \; \ell_4 \; L = H \; \ell_5 \} \}\ell_6 \)
  \[ \text{nondet}(H, H) = \text{nondet}(\neg H, \neg H) = \{ L \} \]
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_1 = \{ \langle L, L \rangle, \langle H, H \rangle \} \]
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_2 = \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_4 = \{ \langle L, L \rangle \} \]
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_3 = \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_5 = \{ \langle H, H \rangle \} \]
  \[ \hat{\mathcal{S}}_{\text{diff}}^3 [S'] \; \ell_6 = \{ \langle H, L \rangle \} \cup \{ \langle H, H \rangle \} \]
Structural static potential value dependency analysis (cont’d)

- statement list $Sl ::= Sl' S$

\[
\tilde{S}_{\text{diff}}^{\exists}[Sl] \; \ell \triangleq (\ell \in \text{labx}[Sl'] ? \tilde{S}_{\text{diff}}^{\exists}[Sl'] \; \ell
\]
\[
\quad \mid \ell \in \text{labx}[S] \setminus \{\text{at}[S]\} ? \tilde{S}_{\text{diff}}^{\exists}[Sl'] \; \text{at}[S] \; \ell
\]
\[
\quad \triangledown \emptyset
\]

(16.a)

where $r_1 \triangledown r_2 \triangleq \{\langle x, y \rangle \mid \exists z. \langle x, z \rangle \in r_1 \land \langle z, y \rangle \in r_2\}$. 

(16.b)
Structural static potential value dependency analysis (cont’d)

- iteration $S ::= \text{while } \ell \ (B) \ S_b$

\[
\widehat{S}_{\text{diff}}^\exists \cdot [S] \ \ell' = (\text{lfp}^\exists F_{\text{diff}}^\exists [\text{while } \ell \ (B) \ S_b]) \ \ell'
\]

\[
F_{\exists}^\text{diff} [\text{while } \ell \ (B) \ S_b] \ X \ \ell' =
\begin{cases}
\ell' = \ell \ ? \ 1 \lor (X(\ell) \ ? (\widehat{S}_{\text{diff}}^\exists [S_b] \ \ell' \ ] \ \text{nondet}(B, B))) \\
\ell' \in \text{in}[S_b] \ ? X(\ell) \ ? (\widehat{S}_{\text{diff}}^\exists [S_b] \ \ell' \ ] \ \text{nondet}(B, B)) \\
\ell' = \text{after}[S] \ ? X(\ell) \lor (X(\ell) \ ? (V \times \text{mod}[S_b])) \lor \\
X(\ell) \ ? \left(\left(\bigcup_{\ell'' \in \text{breaks-of}[S_b]} \widehat{S}_{\text{diff}}^\exists [S_b] \ \ell''\right)\ \text{nondet}(B, B)\right)
\end{cases}
\]

\[
: \emptyset
\]
Reduced product with a relational value analysis
Structural compositionality

In the following statement, \( x \) and \( y \) at \( \ell_1 \) depend on \( x \) at \( \ell_0 \)

\[ \ell_0 \ y = x \ ; \ \\
\ell_1 \]

In the following statement, \( x \) and \( y \) at \( \ell_2 \) depend on \( x \) at \( \ell_1 \)

\[ \ell_1 \ y = y-x \ ; \ \\
\ell_2 \]

In the sequential composition of the two statements

\[
/* \ x = x_0, y = y_0 */
\]

\[ \ell_0 \ y = x \ ; \\
\ell_1 \ y = y-x \ ; \\
\ell_2 \]

\( y \) at \( \ell_2 \) depends on \( x \) at \( \ell_1 \) which depends on \( x \) at \( \ell_0 \).

By composition, \( y \) at \( \ell_2 \) depends on \( x \) at \( \ell_0 \).

However, \( y = 0 \) at \( \ell_2 \) so \( y \) at \( \ell_2 \) does not depend on \( x \) at \( \ell_0 \).
Structural compositionality (cont’d)

In the following statement, \( x \) and \( y \) at \( \ell_1 \) depend on \( x \) at \( \ell_0 \)
\[
\ell_0 \ y = x ; \\
\ell_1 \ y = y-x ; \\
\ell_2
\]

In the following statement, \( x \) and \( y \) at \( \ell_2 \) depend on \( x \) at \( \ell_1 \)
\[
\ell_1 \ y = y-x ; \\
\ell_2 \ y = y-x ; \\
\ell_2
\]

In the sequential composition of the two statements
\[
\ell_0 \ y = x ; \\
\ell_1 \ y = y-x ; \\
\ell_2
\]

\( y \) at \( \ell_2 \) depends on \( x \) at \( \ell_1 \) which depends on \( x \) at \( \ell_0 \).

By composition, \( y \) at \( \ell_2 \) depends on \( x \) at \( \ell_0 \).

However, \( y = 0 \) at \( \ell_2 \) so \( y \) at \( \ell_2 \) does not depend on \( x \) at \( \ell_0 \).

\( \implies \) reduced product with a value analysis (here Karr linear equalities)
Dye instrumented semantics
Dye analysis in hydrology

When a river is lost in the ground (e.g. la perte du Gour de Champlive in France)

a dye analysis with fluorescein can be used to discover its resurgences
Dye instrumented semantics

- The initial values of the variables are colored with different colors
- The initial color of a variable can be the variable name
- The dye instrumented semantics is sound iff it associates to each variable $y$ and program point $\ell$ the set of colors/variables $x$ upon which is depends

$$\{x \mid S^{+\infty}[p] \in D\ell\langle x, y \rangle\}$$

- Better approach than postulating the dye instrumented semantics [Cheney, Ahmed, and Acar, 2011] (e.g. the mix of colors at tests and assignments can be postulated arbitrarily)
Tracking analysis

- Partition the variables $V$ into racked $T$ and untracked $U$ variables ($V = T \cup U$ and $T \cap U = \emptyset$)

- Tracking abstraction $\alpha^T(D)$ of a dependency property $D \in \mathcal{L} \rightarrow \wp(V \times V)$

\[
\alpha^T(D)^{\ell} \triangleq \{y \mid \exists x \in T. \langle x, y \rangle \in D^{(\ell)}\}
\]

- Sound tracking analysis

\[
S^T[S] \supseteq \alpha^T(\alpha^d(\{S^{+\infty}[S]\}))
\]

- Examples: taint analysis in privacy/security checks [Ferrara, Olivieri, and Spoto, 2018; Spoto, Burato, Ernst, Ferrara, Lovato, Macedonio, and Spiridon, 2019] (tracked is tainted, untracked is untainted); binding time analysis in offline partial evaluation [Hatcliff, 1998] (tracked is dynamic, untracked is static) and absence of interference [Bowman and Ahmed, 2015; Goguen and Meseguer, 1984; Heinze and Turker, 2018; Lourenço and Caires, 2015; Volpano, Irvine, and Smith, 1996] (tracked is high (private/untrusted), untracked is low (public/trusted)).
Conclusion
Conclusion

• The dependency analysis is not postulated but derived formally by abstract interpretation of the trace semantics.

• No need for extra notions like (hyper)"properties [Assaf, Naumann, Signoles, Totel, and Tronel, 2017], non-standard abstract interpretation [Urban and Müller, 2018], postulated instrumented semantics [Ørbæk, 1995, Sect. 4], multisemantics [Cabon and Schmitt, 2017], monadic reification [Grimm, Maillard, Fournet, Hritcu, Maffei, Protzenko, Ramananandro, Rastogi, Swamy, and Béguelin, 2018], etc.


References II


References III


The End, Thank you