Calcutational design of a static dependency analysis

Patrick Cousot
New York University, Courant Institute of Mathematics, Computer Science

Dependency

Dependency is prevalent in computer science:
- Non-interference (confidentiality, integrity)
- Security, privacy
- Slicing
- Temporal dependencies in synchronous languages (Lustre, Signal, etc.)
- etc.

The existing definitions
- without semantics justifications (except Assaf, Naumann, Signoles, Totel, and Tronel, 2017 ("hyper-collecting semantics"), Urban and Müller, 2018 on program exit uniquely)

We are interested in principles, in soundness proofs, not so much in a new more powerful dependency analysis.
Program syntax

- C statements limited to integers, assignments, statement lists, conditionals, iterations
- Programs are labelled to designate program points
  - at[S]: entry program point of S starts;
  - after[S]: normal exit program point of S;
  - in[S]: reachable program points of S (excluding after[S]);
  - break-to[S]: breaking point when S contains a break; to exit a loop (then escape[S] = tt);

Value of a variable (and an expression)

- The value of a variable x along a trace π is the last assigned value (or 0 at initialization).

\[
q(\pi \cdot x = E \cdot \rho, v) = v
\]

\[
q(\pi \cdot \text{if } \text{guard} \cdot \rho, \{
q(\pi \cdot \text{then} \cdot \rho, v) \}
q(\pi \cdot \text{else} \cdot \rho, v)
\}) = q(\pi \cdot \text{guard} \cdot \rho, \{v\})
\]

- otherwise

\[
q(\pi \cdot x = \text{expression} \cdot \rho, v) = q(\pi \cdot \text{expression} \cdot \rho, v)
\]

- Value of an arithmetic expression

\[
\delta([1])\rho \triangleq 1
\]

\[
\delta([x])\rho \triangleq \rho(x)
\]

\[
\delta([A_1 - A_2])\rho \triangleq \delta([A_1])\rho - \delta([A_2])\rho
\]

- Same for boolean expressions.

Execution traces

- Program:
  \[
t_i \cdot x = 0 ; \text{while } t; (t_i \cdot x \cdot x + 1 \cdot ) \cdot t_i
\]

- Infinite execution trace:
  \[
t_i \cdot x = 0 \rightarrow t_i \cdot t_i \cdot x = x + 1 \rightarrow t_i \cdot t_i \cdot x = x + 2 \rightarrow ...
\]

- Trace: finite or infinite sequence of program points separated by action

  \[
x = A = \text{value}, B, \neg B, \text{et break ;}
\]

Structural fixpoint prefix/maximal trace semantics \(\hat{\mathcal{E}}^*[S]\)

- The prefix trace semantics \(\hat{\mathcal{E}}^*[S]\) is a relation between
  - an initialization trace \(\pi_0[S]\) arriving at[S], and
  - the prefix execution traces at[S]π continuing this initialization by zero or more execution steps

- The maximal trace semantics \(\hat{\mathcal{E}}^{\infty}[S]\) collects the maximal finite traces and the infinite traces obtained as limits of their prefixes.
Structural fixpoint definition of the prefix trace semantics (I)

- Assignment $S ::= t \leftarrow A$ (where at $S = t$)
  
  $S^* [S] \triangleq \{(\pi, t) \mid \pi \in T^* \uplus \{\langle \pi', t \leftarrow A = v \text{ after}[S] \mid \pi' \in T^* \land v = b[b][q(\pi')]\} \}

- Iteration $S ::= \text{while} \ (B) S_b$ (where at $[S] = t$):
  
  $S^* [S] = \text{fp} \ F^* [S]$

  $F^* [\text{while} \ (B) S_b](X) \triangleq \{(\pi_1, t', \pi_2) \mid \pi_1, t' \in T^* \land \pi_2 = X \land$

  $\text{at}[S_b] = \pi_1 \land \text{at}[S_b] = \pi_2 \land (X, \pi_1) \land t' = \pi_2 \land t' = t \}$

- A definition of the form $d(S) = \{f(\vec{x}) \mid P(\vec{x}, \vec{y})\}$ has the variables $\vec{z}$ in $P(\vec{x}, \vec{z})$ bound to those of $f(\vec{x})$ whereas $\vec{z}$ is free in $P(\vec{x}, \vec{z})$ since it appears neither in $f(\vec{x})$ nor (by assumption) under quantifiers in $P(\vec{x}, \vec{z})$. The $S$ of $P(\vec{x}, \vec{z})$ is therefore bound to the $S$ of $d(S)$.

Properties

- A property is represented by a set of elements (those elements which have the property)
  
  - Even integers: $2\mathbb{Z} \triangleq \{2k \mid k \in \mathbb{Z}\}$
  - $x$ has property $P$ is $x \in P$
  - Implication is $P_1 \subseteq P_2$
Semantic property

- The prefix trace semantics belongs to $\rho(\mathbb{T}^* \times \mathbb{T}^{\omega\omega})$
- A semantics property belongs to $\rho(\rho(\mathbb{T}^* \times \mathbb{T}^{\omega\omega}))$
- The abstraction

$$\langle \rho(\rho(\mathbb{T}^* \times \mathbb{T}^{\omega\omega})), \leq \rangle \overset{\lambda Q \cdot \rho(Q)}{\longrightarrow} \langle \rho(\mathbb{T}^* \times \mathbb{T}^{\omega\omega}), \leq \rangle$$

provides trace properties (e.g., safety, liveness, etc.)

Dependency, informally

- At program point $\ell$, the variable $y$ depends upon the initial value $x_0$ of variable $x$ iff changing only $x_0$ will change the non-empty sequences of values $y_0, y_1, \ldots$ of $y$ observed at $\ell$ whenever control reaches $\ell$
- Example: $\ell_0$ if $(x=0) \{ y=x; \ell_1 \} \ell_2$
  - $y$ does not depend on $x$ neither at $\ell_0$ nor at $\ell_1$
  - $y$ depends on $x$ at $\ell_2$
- No need to distinguish between explicit and implicit dependencies
- Absence of observation is not an observation
- No timing channels
Observation of the sequence of values of a variable at a program point

- non-empty initialization trace \( \pi_0 \in T^* \)
- non-empty continuation trace \( \pi \in T^{\omega_1} \)
- \( \text{seqval}[y](\pi_0, \pi) \) is the sequence of values of the variable \( y \) at program point \( t \) along the trace \( \pi \) continuing \( \pi_0 \)

\[ \text{seqval}[y](\pi_0, 0) \triangleq \varrho(\pi_0, y) \]
\[ \text{seqval}[y](\pi_0, t') \triangleq a \]
\[ \text{seqval}[y](\pi_0, t \xrightarrow{a} t') \triangleq \varrho(\pi_0, y) \cdot \text{seqval}[y](\pi_0, t \xrightarrow{a} t', \pi) \]
\[ \text{seqval}[y](\pi_0, t \xrightarrow{a} t') \triangleq \text{seqval}[y](\pi_0, t \xrightarrow{a} t', \pi) \]

- \( \text{seqval}[y](\pi_0, \pi) \) is the empty sequence \( \varnothing \) if \( t \) never appears in \( \pi \)

(co-inductive definition for infinite traces).

\[ \text{Diff}(\omega, \omega') \triangleq \exists \omega_0, \omega_1, \omega'_0, \omega'_1 \cdot \omega = \omega_0 \cdot \omega_1, \omega' = \omega'_0 \cdot \omega'_1 \land \varnothing \neq \omega_1 \neq \omega'_1 \]

\[ \text{Value dependency abstraction} \]

Dependency, formally

- Dependency property:

\[ \mathcal{D}_{\text{diff}}(x, y) \triangleq \{ (\Pi \in \mathcal{T}(T^* \times T^{\omega_1}) | \exists (\pi_0, \pi_1, (\pi'_0, \pi'_1)) \in \Pi . (\forall z \in \mathcal{V} \setminus \{x\}. \varrho(\pi_0, z) = \varrho(\pi'_0, z) \land \varrho(\pi_0, \pi_1), \varrho(\pi'_0, \pi'_1)) \} \]

- \( y \) depends on the initial value of \( x \) at program point \( t \) in program \( P \) is:

\[ \mathcal{S}^{\omega_1}[P] \in \mathcal{D}_{\text{diff}}(x, y) \]

- Lemma

\[ \mathcal{S}^{\omega_1}[P] \in \mathcal{D}_{\text{diff}}(x, y) \Rightarrow \mathcal{S}[P] \in \mathcal{D}_{\text{diff}}(x, y) \]
Abstraction en dépendance de données

- The abstraction of a semantic property $S \in \rho(p(T^* \times T^*))$ into a value dependency property $\alpha^*(S) \in \mathcal{L} \rightarrow \rho(\forall \times \forall)$ is:

$$\alpha^*(S) \triangleq \{ (x, y) \mid S \in \mathcal{D}_{\text{def}}(x, y) \}$$

- This is a Galois connection:

**Lemma 1** $(\rho(p(T^* \times T^*)), \subseteq) \xrightarrow{y^*} (\mathcal{L} \rightarrow \rho(\forall \times \forall), \supseteq)$ where the concretization of a dependency property $D \in \mathcal{L} \rightarrow \rho(\forall \times \forall)$ is:

$$y^*(D) \triangleq \bigcap_{(x, y) \in D(x)} \mathcal{D}_{\text{def}}(x, y)$$

(The more semantics, the less common dependencies)

Potential dependency

- $\alpha^*(\mathcal{S}^* [S]))$ is not computable (Rice theorem)
- We design an over-approximation:

Abstract potential dependency semantics $\mathcal{S}_{\text{eff}}^*$:

$$\alpha^*(\mathcal{S}_{\text{eff}}^* [S])) \subseteq \mathcal{S}_{\text{eff}}^* [S]$$

- The abstraction in D. E. Denning and P. J. Denning, 1977 is purely syntactic;
- We do a little better by taking the semantics is a simple way.

Static dependency analysis

Calculation design

- $\mathcal{S}_{\text{eff}}^* [S]$ is designed by calculus (in principle can be checked in Coq as Jourdan, Laporte, Blazy, Leroy, and Pichardie, 2015);
- By structural induction on the program syntax;
- By fixpoint approximation for iteration:

**Theorem (fixpoint over-approximation)** If $(\mathcal{C}, \subseteq, \rightarrow, \tau, \rho, \mu)$ and $(\mathcal{A}, \times, 0, 1, \gamma, \lambda)$ are complete lattices, $(\mathcal{C}, \subseteq, \rightarrow, \tau, \rho, \mu)$ is a Galois connection, $f \in \mathcal{C} \rightarrow \mathcal{C}$ and $\forall f \in \mathcal{A} \rightarrow \mathcal{A}$ are monotonically increasing and $\alpha \cdot f \subseteq f \cdot \alpha$ (semi-commutation) then $\text{IFP} f \subseteq \alpha (\text{IFP} f)$.

- Finite domain, no need for widening
Abstract potential dependency semantics of assignment $S ::= x = A$;

$$\overrightarrow{D^A}[[S]]^t = \left\{ t = at[S] \land \{(y, y) \mid y \in \mathcal{V}\} \mid t = after[S] \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \right\}^\cup$$

$$\overrightarrow{D^A}[A] = \{ y \mid \exists p \in Ev \cdot \exists v \in \mathcal{V} . \overrightarrow{D^A}[A][p \neq \mathcal{V}[A][p] \land \mathcal{V}[A][p] \neq v]\}$$

Examples:
- after $x = y - y$ ; $x$ does not depend on $y$.
- after $x = y ; x = y - x$ ; $x$ depends on the initial value of $x$ and $y$ (to be more precise information of values of variables must be kept such as $y - x = 0$ by symbolic constant analysis)

Proof I

The case $t = at[S]$ was handled in (44.39). Assume $t = after[S]$.

$$\overrightarrow{D^A}[a][S][t] = \overrightarrow{D^A}[a][S][t] \land \{(y, y) \mid y \in \mathcal{V}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\}$$

$$\overrightarrow{D^A}[A] \subseteq \overrightarrow{D^A}[A]$$

Example:
- $x = y - y$; $x$ does not depend on $y$.
- $x = y - x$; $x$ depends on the initial value of $x$ and $y$ (to be more precise information of values of variables must be kept such as $y - x = 0$ by symbolic constant analysis)

Proof II

$$\overrightarrow{D^A}[a][S][t] = \overrightarrow{D^A}[a][S][t] \land \{(y, y) \mid y \in \mathcal{V}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\}$$

Example:
- $x = y - y$; $x$ does not depend on $y$.
- $x = y - x$; $x$ depends on the initial value of $x$ and $y$ (to be more precise information of values of variables must be kept such as $y - x = 0$ by symbolic constant analysis)

Abstract potential dependency semantics of the iteration $S ::= while\ t \ (B) S_0$

$$\overrightarrow{D^A}[S][t] = \left\{ (l \in [S] \land \overrightarrow{D^A}[while\ t \ (B) S_0])[l] \right\}^\cup$$

$$\overrightarrow{D^A}[while\ t \ (B) S_0] X = \left\{ (C = t \land X \lor \{X(t) \land \overrightarrow{D^A}[S_0][t]\} \right\}^\cup$$

- Can be refined by taking test determinacy into account (e.g. after test $x = 1$, $x$ can only have value 1 so nothing can depend on $x$ afterwards).

Proof

$\overrightarrow{D^A}[a][S][t] = \overrightarrow{D^A}[a][S][t] \land \{(y, y) \mid y \in \mathcal{V}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\} \land \{(x, y) \mid y \in \overrightarrow{D^A}[A] \cup \{(y, y) \mid y \neq x\}\}$$

Example:
- $x = y - y$; $x$ does not depend on $y$.
- $x = y - x$; $x$ depends on the initial value of $x$ and $y$ (to be more precise information of values of variables must be kept such as $y - x = 0$ by symbolic constant analysis)
No structural compositionality

In the following statement, x and y at $t_1$ depend on x at $t_0$.

\[
\begin{align*}
    t_0 & \ y = x \\
    t_1 & \ y = y_0 \\
\end{align*}
\]

In the following statement, x and y at $t_1$ depend on x at $t_1$.

\[
\begin{align*}
    t_1 & \ y = y - x \\
    t_2 & \ y = x_0 - x_0 \\
\end{align*}
\]

In the sequential composition of the two statements

\[
\begin{align*}
    t_0 & \ y = x \\
    t_1 & \ y = y_0 \\
    t_2 & \ y = y - x \\
\end{align*}
\]

y at $t_2$ depends on x at $t_0$ which depends on x at $t_1$, so by composition, y at $t_2$ depends on x at $t_0$.

However, y = 0 at $t_0$ so y at $t_2$ does not depend on x at $t_0$.

Improving precision

- To improve precision one must take values of variables into account;
- Reduced product with a reachability analysis (e.g., Cortesi, Ferrara, Halder, and Zanioli, 2018; Zanioli and Cortesi, 2011)

Dependence analysis is an abstract interpretation

- No need for a generalized theory (as proposed by Assaf, Naumann, Signoles, Totel, and Tronel, 2017; Urban and Müller, 2018)
- This includes further abstractions, dye analysis, taint analysis, etc.
- Many possible variants (e.g., by changing diff to = we get timing channel dependency).
- Data dependence analysis to detect parallelism in sequential codes Padua and Wolfe, 1986 is also an abstract interpretation Tzolovski, 1997, Tzolovski, 2002, Ch. 5.
References IV

