Set-Constraint-Based Analysis by Abstract Interpretation

Jaeho Shin
netj@ropas.snu.ac.kr

Programming Research Lab, Seoul National University

\[a\]This talk is based on Cousot and Cousot’s work [1].
Today’s Goal

- Is to understand that grammar-based and set-based analyses can be done by abstract interpretation.
- And to see the advantages of using the abstract interpretation framework.
1. Program Semantics
   (a) Standard Semantics
   (b) Collecting Semantics
2. Formal-Language-Based Abstraction
3. Grammar-Based Abstraction
4. Abstract Interpretation of Grammars
5. Conclusion
Standard Semantics

- *Standard semantics* of each program $P$
  \[ S_{sd}[P] \in B_{sd} \]
  is a specification of the possible program execution.
- Behavior domain $B_{sd}$ involves symbolic semantic values in $L$ associated to indexes in $I$.
- Formal language $L$ of symbolic values
  \[ t \rightarrow f^0 | f^n(t_1, \ldots, t_n) \]
Example Program

\[ f(N) = \]
\[
\text{if } (N \leq 0) \text{ then } \\
\quad \text{cons}(0, \text{cons}(0, \text{cons}(0, \text{nil}))) \\
\text{else let } X = f(N-1) \text{ in } \\
\quad \text{cons}(a(\text{hd}(X)), \text{cons}(b(\text{hd}(\text{tl}(X))), \\
\quad \quad \text{cons}(c(\text{hd}(\text{tl}(\text{tl}(X)))), \text{nil})))
\]
Example: Standard Semantics

- The function computed by the program

\[ f_{sd} = \lambda n. \text{if } n \leq 0 \text{ then } \text{cons}(0, \text{cons}(0, \text{cons}(c(0, nil)))) \]

else

\[ \text{cons}(a^n(0), \text{cons}(b^n(0), \text{cons}(c^n(0), nil))) \]

- Behavior domain

\[ B_{sd} = \mathbb{Z} \rightarrow (\mathbb{L} \cup \{\bot\}) \]

where \( \mathbb{L} \) is the set of lists, \( \bot \) denotes non-termination.
Collecting Semantics

- Collecting semantics of each program $P$

$$S_{co}[P] = \{S_{sd}[P]\} \in D_{co} = \wp(B_{sd})$$

associates a strongest property,

- so that we can formalize “program $P$ has property $I$” as $f_{co} \subseteq I \iff f_{sd} \in I$.

- $S_{co} = \text{lfp } F_{co}$ where $F_{co} \in D_{co} \rightarrow D_{co}$ is a concrete property transformer.
Example: Collecting Semantics

- The set of possible functions computed by the program (ignoring non-termination)

\[ f_{co} = \{ \lambda n. \text{if } n \leq 0 \text{ then } \]
\[ \text{cons}(0, \text{cons}(0, \text{cons}(c(0, nil)))) \]
\[ \text{else} \]
\[ \text{cons}(a^n(0), \text{cons}(b^n(0), \text{cons}(c^n(0), nil))) \} \]

- So the domain of collecting semantics is

\[ D_{co} = \emptyset(\mathbb{Z} \rightarrow \mathbb{L}) \]
Formal-Language-Based Abstraction

- Provides a basis for grammar-based abstract semantics and a direct connection to set-constraints.
- Formal language abstract domain

\[ D_{fl} = I \rightarrow \wp(L) \]

is a complete boolean lattice

\((D_{fl}, \subseteq, \bot_{fl}, \top_{fl}, \cup, \cap, \setminus)\)

for the point-wise subset ordering \(\subseteq\) with \(\bot_{fl} = \lambda i.\{\}, \top_{fl} = \lambda i.L\).
Language and application dependent abstraction $\langle \alpha_{fl}, \gamma_{fl} \rangle$

$$(D_{co}, \subseteq) \xleftrightarrow{\gamma_{fl}} (D_{fl}, \subseteq)$$
By lifting to higher-order

\[(D_{co} \rightarrow D_{co}, \subseteq) \xrightarrow{\gamma_{fl}} (D_{fl} \rightarrow D_{fl}, \subseteq)\]

where \(\alpha_{fl}(F) = \alpha_{fl} \circ F \circ \gamma_{fl}\) and \(\gamma_{fl}(F^\#) = \gamma_{fl} \circ F^\# \circ \alpha_{fl}\)

Formal language semantics is defined as

\[S_{fl} = \text{lfp } F_{fl}\]

where the formal language transformer

\[F_{fl} = \alpha_{fl}(F_{co}) \in D_{fl} \rightarrow D_{fl}\]
Meta-Language $\mathcal{L}_{fl}$ for Encoding

\[ T \rightarrow x | f^0 | f^n(T_1, \ldots, T_n) \]
\[ e \rightarrow \mathcal{X} | \{ T' : T_1 \in e_1, \ldots, T_n \in e_n \} | e_1 \cup e_2 | \neg e \]

\[
\begin{align*}
\llbracket T \rrbracket & \in (v \rightarrow L) \rightarrow L \\
\llbracket x \rrbracket \kappa & = \kappa(x) \\
\llbracket f^0 \rrbracket \kappa & = f^0 \\
\llbracket f^n(T_1, \ldots, T_n) \rrbracket \kappa & = f^n(\llbracket T_1 \rrbracket \kappa, \ldots, \llbracket T_n \rrbracket \kappa) \\
\llbracket e \rrbracket & \in (V \rightarrow \wp(L)) \rightarrow \wp(L) \\
\llbracket \mathcal{X} \rrbracket \rho & = \rho(\mathcal{X}) \\
\llbracket \{ T' : T_1 \in e_1, \ldots, T_n \in e_n \} \rrbracket \rho & = \{ \llbracket T' \rrbracket \kappa : \kappa \in v \rightarrow L \land \bigwedge_{i=1}^{n} \llbracket T_i \rrbracket \kappa \in \llbracket e_i \rrbracket \rho \} \\
\llbracket e_1 \cup e_2 \rrbracket \rho & = \llbracket e_1 \rrbracket \rho \cup \llbracket e_2 \rrbracket \rho \\
\llbracket \neg e \rrbracket \rho & = L - \llbracket e \rrbracket \rho
\end{align*}
\]
FL Transformer in $\mathcal{L}_{\text{fl}}^{1/2}$

Formal Language Transformer $F_{\text{fl}}$ can be specified as the fixpoint equation,

$$\begin{cases} 
\rho(\mathcal{X}) = F_{\text{fl}}(\rho)(\mathcal{X}) \\
\mathcal{X} \in \Delta
\end{cases} \iff \begin{cases} 
\mathcal{X} = e \mathcal{X} \\
\mathcal{X} \in \Delta
\end{cases}$$

where

$$\Delta \subseteq V = I$$

$$\rho \in D_{\text{fl}} = I \to \wp(L)$$

$$F_{\text{fl}}(\rho)(\mathcal{X}) = \llbracket e \mathcal{X} \rrbracket \rho$$

$$\forall \mathcal{X} \in V - \Delta : \forall \rho : F_{\text{fl}}(\rho)(\mathcal{X}) = \{\}$$
By Tarski’s fixpoint theorem,

$$\text{lfp } F_{fl} = \cap \{ X : F_{fl}(X) \subseteq X \}$$

the previous fixpoint equation has the same least solution as the system of constraints,

$$\begin{cases} 
    e_X \subseteq X \\
    X \in \Delta 
\end{cases}$$
Example: FL Transformer

Fixpoint equations for FL transformer of $f$ is

$$
\begin{align*}
\{ \text{cons}(0, \text{cons}(0, \text{cons}(0, \text{nil}))) \} & \subseteq \mathcal{X} \\
\{ \text{cons}(a(x), \text{cons}(b(y), \text{cons}(c(z), \text{nil}))) \} & \subseteq \mathcal{X} \\
\text{cons}(x, \text{cons}(y, \text{cons}(z, \text{nil}))) & \in \mathcal{X}
\end{align*}
$$

where $\mathcal{X}$ is the index for the function $f$.

The least solution is

$$
\mathcal{X} = \{ \text{cons}(a^n(0), \text{cons}(b^n(0), \text{cons}(c^n(0), \text{nil}))) : n \geq 0 \}
$$

that encodes a language which is not context-free.
In general, elements of $D_{fl}$ aren’t computer-representable. We need grammar.

Context-free grammar $G = \langle T, N, P \rangle \in D_{gr}$

- $T \subseteq A$ set of terminals
- $N \subseteq V$ set of non-terminals $\mathcal{X}$
- $P$ productions of the form $\mathcal{X} \Rightarrow \sigma$
  
  where $\sigma \in (A \cup V)^*$
Recall that set-constraint-based analysis [2] is restricted to *regular tree grammars*, hence

\[ D_{\text{gr}} = \{ G : G \text{ is a regular tree grammar} \} \]

And productions of \( G \) are in the form of \( X \Rightarrow g \) where

\[ g \rightarrow X | f^0 | f^n(g_1, \ldots, g_n) \]
Language generated by the grammar $G$ for $\mathcal{X}$ is

$$L_G(\mathcal{X}) = \{ w \in A^* : \mathcal{X} \Rightarrow^*_G w \}$$

$$L_G(f^0) = \{ f^0 \}$$

$$L_G(f^n(g_1, \ldots, g_n)) = \{ f^n(w_1, \ldots, w_n) : w_i \in L_G(g_i) \}$$

Concretization to $D_{fl}$

$$\gamma_{gr} \in D_{gr} \rightarrow D_{fl}$$

$$\gamma_{gr} = \lambda G. \lambda \mathcal{X}. L_G(\mathcal{X})$$
Languages generated by a grammar $G$ for each $\mathcal{X}$ is the $\subseteq$-least solution to the system of fixpoint equations:

$$\mathcal{L}_G(\mathcal{X}) = \{ f^0 | \mathcal{X} \Rightarrow f^0 \in P \} \cup$$

$$\{ f^n(t_1, \cdots, t_n) | \mathcal{X} \Rightarrow f^n(g_1, \cdots, g_n) \in P \land$$

$$\forall i = 1, \cdots, n : t_i \in \mathcal{L}_G(g_i) \}$$

$\mathcal{X} \in N$
Representing the generated language by $\mathcal{X}$ itself, and using Tarski’s fixpoint theorem, equations can be written in the form of set-constraints:

\[
\begin{align*}
\mathcal{X} & \supseteq f^0, \quad \mathcal{X} \supseteq f^n(g_1, \ldots, g_n) \\
\mathcal{X} & \Rightarrow f^0 \in P, \quad \mathcal{X} \Rightarrow f^n(g_1, \ldots, g_n) \in P
\end{align*}
\]

where $g \rightarrow \mathcal{X} \mid f^0 \mid f^n(g_1, \ldots, g_n)$

\[
\begin{align*}
f^0 & = \{f^0\} \\
f^n(L_1, \ldots, L_n) & = \{f^n(t_1, \ldots, t_n) \mid t_i \in L_i\} \\
g & = \mathcal{L}_G(g)
\end{align*}
\]
Grammar Abstract Domain

Grammar Abstract Domain \( (D_{gr} / \equiv, \subseteq) \) where

\[
G_1 \subseteq G_2 \iff \forall \mathcal{X} \in V : \mathcal{L}_{G_1}(\mathcal{X}) \subseteq \mathcal{L}_{G_2}(\mathcal{X})
\]

\[
G_1 \equiv G_2 \iff G_1 \subseteq G_2 \land G_2 \subseteq G_1
\]

is not a complete partial order.
Example: Grammar Transformer

Grammar transformer $F_{gr} \in D_{gr} \rightarrow D_{gr}$ for the example program is

$$F_{gr} (\langle T, N, P \rangle) = \langle T \cup \{0, a, b, c, cons, nil\}, N \cup \{X\},$$

$$\{X \Rightarrow cons(0, cons(0, cons(0, nil)))\} \cup$$

$$\{X \Rightarrow cons(a(x), cons(b(y), cons(c(z), nil))) :$$

$$X \Rightarrow cons(x, cons(y, cons(z, nil))) \in P\}$$

The iterates $F_{gr}^n (\bot_{gr})$ where $\bot_{gr} = \langle \{\}, \{\}, \{\}, \{\} \rangle$ is infinite and without limit. We need widening.
Widening of Grammars

$G_1 \uplus G_2$ can be defined as repeated transformation that replaces all $m > 1$ productions in $G_1 \cup G_2$ of the form:

\[
\begin{array}{l}
\{ \\
\chi \Rightarrow f^n(T_1^i, \cdots, T_n^i) \\
i = 1, \cdots, m \\
\} \\
y \text{by} \\
\{ \\
\chi \Rightarrow f^n(Z_1, \cdots, Z_n) \\
Z_k \Rightarrow T_k^i \\
i = 1, \cdots, m \\
k = 1, \cdots, n \\
\}
\end{array}
\]

where $Z_k$ are non-terminals neither used in $G_1$ nor $G_2$ and all other occurrences of non-terminal $T_k^i$ are replaced by $Z_k$. 
Grammar Abstract Semantics

With such widening $\nabla \in D_{\text{gr}} \times D_{\text{gr}} \rightarrow D_{\text{gr}}$, we have an ultimately stabilizing iteration,

$$
R^0 = \bot_{\text{gr}} \\
R^{n+1} = R^n \nabla F_{\text{gr}} (R^n)
$$

The grammar abstract semantics is

$$
S_{\text{gr}} = R^n, \quad n \text{ is } k \text{ such that } F_{\text{gr}} (R^k) \subseteq R^k
$$

which is sound $S_{\text{co}} \subseteq \gamma_{\text{fl}} \circ \gamma_{\text{gr}} (S_{\text{gr}})$. 
An Abstract Interpretation

Now, we have an abstract interpretation such that:

- Abstract domain $D_{gr}$ has infinite values $G$ representing infinite concrete sets $\gamma_{gr}(G)(\mathcal{X})$;
- All abstract values in $D_{gr}$ have a finite computer representation;
- $(D_{gr}, \subseteq)$ has infinite strictly increasing chains without limits in $D_{gr}$;
- $F_{gr}$ is sound $F_{fl} \circ \gamma_{gr} \subseteq \gamma_{gr} \circ F_{gr}$ and computable;
- Increasing chain $F_{gr}^n, n \leq 0$ is not convergent, so that a widening is necessary.
Conclusion

- As we have seen, set-constraint-based analysis can be actually done by *iterations* of abstract interpretation.

- Contrast to set-constraint-based analysis which only manipulates symbolic structures, combining grammar-based abstractions with non-grammar-based ones (e.g. arithmetic) will be easier and smoother in the abstract interpretation framework.