Set-Constraint-Based Analysis by Abstract Interpretation [a]

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.

Contents

Standard Semantics

- *Standard semantics* of each program P

$$S_{\mathrm{sd}}[\![P]\!] \in B_{\mathrm{sd}}$$

  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

$$t \longrightarrow f^0 \mid f^n(t_1, ..., t_n)$$

Example Program

```
f(N) =
  if (N <= 0) then
    cons(0, cons(0, cons(0, nil)))
  else let X = f(N-1) in
    cons(a(hd(X)), cons(b(hd(tl(X))),
      cons(c(hd(tl(tl(X)))), nil)))
```

Example: Standard Semantics

- The function computed by the program

$$
\begin{aligned}
f_{\mathrm{sd}} = \quad & \lambda n.\text{if } n \leq 0 \ \text{ then} \\
& \qquad \qquad \mathit{cons}(0, \mathit{cons}(0, \mathit{cons}(c(0, \mathit{nil})))) \\
& \qquad \text{else} \\
& \qquad \qquad \mathit{cons}(a^n(0), \mathit{cons}(b^n(0), \mathit{cons}(c^n(0), \mathit{nil})))
\end{aligned}
$$

- Behavior domain

$$B_{\mathrm{sd}} = \mathbb{Z} \longrightarrow (\mathbb{L} \cup \{\perp\})$$

where $\mathbb{L}$ is the set of lists, $\perp$ denotes non-termination.

Collecting Semantics

- *Collecting semantics* of each program P

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

  associates a strongest property,
- so that we can formalize "program P has property I" as $f_{\mathrm{co}} \subseteq I \iff f_{\mathrm{sd}} \in I$.
- $S_{\mathrm{co}} = \mathrm{lfp}\, F_{\mathrm{co}}$ where $F_{\mathrm{co}} \in D_{\mathrm{co}} \to D_{\mathrm{co}}$ is a concrete property transformer.

Example: Collecting Semantics

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

$$
\begin{aligned}
f_{\mathrm{co}} = \ & \{\lambda n.\text{if } n \leq 0 \ \text{ then} \\
& \qquad \mathit{cons}(0, \mathit{cons}(0, \mathit{cons}(c(0, \mathit{nil})))) \\
& \quad \text{else} \\
& \qquad \mathit{cons}(a^n(0), \mathit{cons}(b^n(0), \mathit{cons}(c^n(0), \mathit{nil})))\}
\end{aligned}
$$

- So the domain of collecting semantics is

$$D_{\mathrm{co}} = \wp(\mathbb{Z} \to \mathbb{L})$$

Formal-Language-Based Abstraction 1/3

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

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

is a complete boolean lattice $(D_{\text{fl}}, \dot{\subseteq}, \dot{\bot}_{\text{fl}}, \dot{\top}_{\text{fl}}, \dot{\cup}, \dot{\cap}, \dot{\neg})$ for the point-wise subset ordering $\dot{\subseteq}$ with $\dot{\bot}_{\text{fl}} = \lambda i.\{\}, \dot{\top}_{\text{fl}} = \lambda i.L$.

Formal-Language-Based Abstraction 2/3

- Language and application dependent abstraction $\langle \alpha_{\mathrm{fl}}, \gamma_{\mathrm{fl}} \rangle$

$$(D_{\mathrm{co}}, \subseteq) \underset{\alpha_{\mathrm{fl}}}{\overset{\gamma_{\mathrm{fl}}}{\leftrightarrows}} (D_{\mathrm{fl}}, \dot{\subseteq})$$

Formal-Language-Based Abstraction 3/3

- By lifting to higher-order

$$(D_{\mathrm{co}} \rightarrow D_{\mathrm{co}}, \subseteq) \underset{\alpha_{\mathrm{fl}}}{\overset{\gamma_{\mathrm{fl}}}{\longleftrightarrow}} (D_{\mathrm{fl}} \rightarrow D_{\mathrm{fl}}, \dot{\subseteq})$$

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

- *Formal language semantics* is defined as

$$S_{\mathrm{fl}} = \mathrm{lfp}\, F_{\mathrm{fl}}$$

where the formal language transformer
$F_{\mathrm{fl}} = \alpha_{\mathrm{fl}}(F_{\mathrm{co}}) \in D_{\mathrm{fl}} \rightarrow D_{\mathrm{fl}}$

Meta-Language $\mathcal{L}_{\mathrm{fl}}$ for Encoding

$$
\begin{array}{rcl}
T & \rightarrow & x \mid f^0 \mid f^n(T_1, \cdots, T_n) \\
e & \rightarrow & \mathcal{X} \mid \{T' : T_1 \in e_1, \cdots, T_n \in e_n\} \mid e_1 \cup e_2 \mid \neg e
\end{array}
$$

$$
\begin{array}{rcl}
[\![T]\!] & \in & (v \rightarrow L) \rightarrow L \\
[\![x]\!]\kappa & = & \kappa(x) \\
[\![f^0]\!]\kappa & = & f^0 \\
[\![f^n(T_1, \cdots, T_n)]\!]\kappa & = & f^n([\![T_1]\!]\kappa, \cdots, [\![T_n]\!]\kappa)
\end{array}
$$

$$
\begin{array}{rcl}
[\![e]\!] & \in & (V \rightarrow \wp(L)) \rightarrow \wp(L) \\
[\![\mathcal{X}]\!]\rho & = & \rho(\mathcal{X}) \\
[\![\{T' : T_1 \in e_1, \cdots, T_n \in e_n\}]\!]\rho & = & \{[\![T']\!]\kappa : \kappa \in v \rightarrow L \wedge \bigwedge_{i=1}^{n} [\![T_i]\!]\kappa \in [\![e_i]\!]\rho\} \\
[\![e_1 \cup e_2]\!]\rho & = & [\![e_1]\!]\rho \cup [\![e_2]\!]\rho \\
[\![\neg e]\!]\rho & = & L - [\![e]\!]\rho
\end{array}
$$

FL Transformer in $\mathcal{L}_{\text{fl}}$ 1/2

Formal Language Transformer F_{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

$$\Delta \subseteq V = I$$

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

$$F_{\text{fl}}(\rho)(\mathcal{X}) = [\![e_{\mathcal{X}}]\!] \rho$$

$$\forall \mathcal{X} \in V - \Delta : \forall \rho : F_{\text{fl}}(\rho)(\mathcal{X}) = \{\}$$

FL Transformer in $\mathcal{L}_{\text{fl}}$ 2/2

By Tarski's fixpoint theorem,

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

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

$$\begin{cases} e_{\mathcal{X}} \subseteq \mathcal{X} \\ \mathcal{X} \in \Delta \end{cases}$$

Example: FL Transformer

Fixpoint equations for FL transformer of `f` is

$$\begin{cases} \{cons(0, cons(0, cons(0, nil)))\} \subseteq \mathcal{X} \\ \{cons(a(x), cons(b(y), cons(c(z), nil))) \mid \\ \qquad cons(x, cons(y, cons(z, nil))) \in \mathcal{X}\} \subseteq \mathcal{X} \end{cases}$$

where $\mathcal{X}$ is the index for the function `f`.
The least solution is

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

that encodes a language which is *not* context-free.

Grammar Abstraction 1/5

- In general, elements of D_{fl} aren't computer-representable. We need *grammar*.
- Context-free grammar $G = \langle T, N, P \rangle \in D_{\mathrm{gr}}$

$T \subseteq A$	set of terminals
$N \subseteq V$	set of non-terminals $\mathcal{X}$
P	productions of the form $\mathcal{X} \Rightarrow \sigma$
	where $\sigma \in (A \cup V)^\star$

Grammar Abstraction 2/5

- Recall that set-constraint-based analysis [2] is restricted to *regular tree grammars*, hence

$$D_{\mathrm{gr}} = \{G : G \text{ is a regular tree grammar}\}$$

- And productions of G are in the form of $\mathcal{X} \Rightarrow g$ where

$$g \to \mathcal{X} \mid f^0 \mid f^n(g_1, \cdots, g_n)$$

Grammar Abstraction 3/5

- *Language generated by the grammar* G for $\mathcal{X}$ is

$$\mathcal{L}_G(\mathcal{X}) = \{w \in A^\star : \mathcal{X} \overset{\star}{\Rightarrow}_G w\}$$

$$\begin{array}{rcl} \mathcal{L}_G(f^0) & = & \{f^0\} \\ \mathcal{L}_G(f^n(g_1, \cdots, g_n)) & = & \{f^n(w_1, \cdots, w_n) : \\ & & \qquad w_i \in \mathcal{L}_G(g_i)\} \end{array}$$

- Concretization to D_{fl}

$$\begin{array}{rcl} \gamma_{\text{gr}} & \in & D_{\text{gr}} \to D_{\text{fl}} \\ \gamma_{\text{gr}} & = & \lambda G.\lambda \mathcal{X}.\mathcal{L}_G(\mathcal{X}) \end{array}$$

Grammar Abstraction 4/5

Languages generated by a grammar G for each $\mathcal{X}$ is the $\subseteq$-least solution to the system of fixpoint equations:

$$
\begin{cases}
\mathcal{L}_G(\mathcal{X}) = & \{f^0 \mid \mathcal{X} \Rightarrow f^0 \in P\} \cup \\
& \{f^n(t_1, \cdots, t_n) \mid \\
& \qquad \mathcal{X} \Rightarrow f^n(g_1, \cdots, g_n) \in P \wedge \\
& \qquad \forall i = 1, \cdots, n : t_i \in \mathcal{L}_G(g_i)\} \\
\mathcal{X} \in N &
\end{cases}
$$

Grammar Abstraction 5/5

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

$$\begin{cases} \mathcal{X} \supseteq \mathsf{f}^0, & \mathcal{X} \supseteq \mathsf{f}^n(\mathsf{g}_1, \cdots, \mathsf{g}_n) \\ \mathcal{X} \Rightarrow f^0 \in P, & \mathcal{X} \Rightarrow f^n(g_1, \cdots, g_n) \in P \end{cases}$$

where $\quad \mathsf{g} \rightarrow \mathcal{X} \mid \mathsf{f}^0 \mid \mathsf{f}^n(\mathsf{g}_1, \cdots, \mathsf{g}_n)$

$$\begin{array}{rcl} \mathsf{f}^0 & = & \{f^0\} \\ \mathsf{f}^n(L_1, \cdots, L_n) & = & \{f^n(t_1, \cdots, t_n) \mid t_i \in L_i\} \\ \mathsf{g} & = & \mathcal{L}_G(g) \end{array}$$

Grammar Abstract Domain

Grammar Abstract Domain $(D_{\mathrm{gr}}/\doteq, \dot\subseteq)$ where

$$
\begin{aligned}
G_1 \dot\subseteq G_2 &\iff \forall \mathcal{X} \in V : \mathcal{L}_{G_1}(\mathcal{X}) \subseteq \mathcal{L}_{G_2}(\mathcal{X}) \\
G_1 \doteq G_2 &\iff G_1 \dot\subseteq G_2 \wedge G_2 \dot\subseteq G_1
\end{aligned}
$$

is not a complete partial order.

Example: Grammar Transformer

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

$$
\begin{aligned}
F_{\mathrm{gr}}(\langle T, N, P\rangle) = \langle & T \cup \{0, a, b, c, cons, nil\}, N \cup \{\mathcal{X}\}, \\
& \{\mathcal{X} \Rightarrow cons(0, cons(0, cons(0, nil)))\} \cup \\
& \quad \{\mathcal{X} \Rightarrow cons(a(x), cons(b(y), cons(c(z), nil))) : \\
& \quad\quad \mathcal{X} \Rightarrow cons(x, cons(y, cons(z, nil))) \in P\}\rangle
\end{aligned}
$$

The iterates ${F_{\mathrm{gr}}}^n(\perp_{\mathrm{gr}})$ where $\perp_{\mathrm{gr}} = \langle \{\}, \{\}, \{\}\rangle$ is infinite and without limit. We need *widening*.

Widening of Grammars

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

$$\left\{\begin{array}{l} \mathcal{X} \Rightarrow f^n(T^i_1, \cdots, T^i_n) \\ i = 1, \cdots, m \end{array}\right. \quad \text{by} \quad \left\{\begin{array}{l} \mathcal{X} \Rightarrow f^n(Z_1, \cdots, Z_n) \\ \left\{\begin{array}{l} Z_k \Rightarrow T^i_k \\ i = 1, \cdots, m \end{array}\right. \\ k = 1, \cdots, n \end{array}\right.$$

where Z_k are non-terminals neither used in G_1 nor G_2 and all other occurrences of non-terminal T^i_k are replaced by Z_k.

Grammar Abstract Semantics

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

$$
\begin{aligned}
R^0 &= \perp_{\mathrm{gr}} \\
R^{n+1} &= R^n \,\nabla\, F_{\mathrm{gr}}(R^n)
\end{aligned}
$$

The *grammar abstract semantics* is

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

which is sound $S_{\mathrm{co}} \subseteq \gamma_{\mathrm{fl}} \circ \gamma_{\mathrm{gr}}(S_{\mathrm{gr}})$.

An Abstract Interpretation

Now, we have an abstract interpretation such that:

- Abstract domain D_{gr} has infinite values G representing infinite concrete sets $\gamma_{\mathrm{gr}}(G)(\mathcal{X})$;
- All abstract values in D_{gr} have a finite computer representation;
- $(D_{\mathrm{gr}}, \dot{\subseteq})$ has infinite strictly increasing chains without limits in D_{gr};
- F_{gr} is sound $F_{\mathrm{fl}} \circ \gamma_{\mathrm{gr}} \dot{\subseteq} \gamma_{\mathrm{gr}} \circ F_{\mathrm{gr}}$ and computable;
- Increasing chain ${F_{\mathrm{gr}}}^n, n \leq 0$ 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.

References

[1] Patrick Cousot and Radhia Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 170–181. ACM Press, 1995.

[2] Nevin Heintze. Set-based analysis of ml programs. In *LFP '94: Proceedings of the 1994 ACM conference on LISP and functional programming*, pages 306–317. ACM Press, 1994.