1. Introduction

Motivation

- Claims:
  - In model-checking the properties to be checked are a user-defined parameter of the model checker;
  - In abstract interpretation, the properties to be discovered are wired in the (generic) static program analyzer;
- Not completely true (see the invariant and intermittent assertions for abstract testing in [1, 2]);
- Can we do better? temporal logic specifications.

References


Objective of the talk

- Not a general presentation;
- Just consider a very simple example:

  Derive a dataflow analysis by abstract interpretation of its temporal logic specification
2. Background

Traditional Dataflow Analysis... [3] 1; 2

- In traditional boolean dataflow analysis, the problem specification for the flowchart program is left informal;
- Or, it is expressed informally along a program path and then there is a merge over all paths 3;
- Correctness by intuition (informal arguments).

Reference


Abstract Interpretation coming in... [4]

- A prefix-closed path semantics of the transition system (program) is expressed in fixpoint form;
- The dataflow problem specification is by an abstraction function describing:
  - The property along one path;
  - How path properties are merged;
- Using abstract interpretation techniques, the boolean dataflow fixpoint equations were formally derived by calculational design from the trace-based semantics. Correctness by construction.
- Only one example (available expressions).

Reference


Model-checking coming in... [5]

- The program is a flowchart (with obvious semantics);
- Abstract interpretation is used to derive the transfer functions at the node level;
- The dataflow problem specification is by a branching time temporal logic formula;
- Classical model-checking algorithms are used to check the program model for the temporal formula;
- Correctness by specification.

Reference

Model-checking with more Abstract Interpretation... [6]

- The abstract flowchart is proved to be an abstract interpretation of a trace-based semantics;
- The dataflow problem specification is by a branching time temporal logic formula;
- It is shown that the algorithms checking the program model for the temporal formula yield the same result as the dataflow equations;
- Correctness by abstract interpretation (flowchart) and by specification (dataflow problem).

Reference

Bad news...

A Bug!

Reference
[7] D.A. Schmidt. Data-flow analysis is model checking of abstract interpretations. 25th ACM POPL, 1998 (see Figure 5 and paragraph 7 "Why Some Analyses are Unsound").

Live Variables Analysis is Unsound

Reference

Liveness analysis claims y to be live before test which is wrong when initially not x == 2! [8]
Questions...

- What should we think of a model-checking based design methodology which let you design unsound analyzes?  
- Who is guilty?  
  - Abstract interpretation?  
  - Data flow analysis?  
  - Model-checking?

7 is everybody as wise as D. Schmidt to find a 25 years old bug? Does B. Steffen tool in Passau effectively signals that bug to the user?  
8 I can’t believe it!  
9 D. Schmidt forgives them by claiming “Of course, data-flow practitioners are well aware of the above problem, and disaster does not arise in practice … But we might not be so fortunate in general.”

My Diagnosis...

- Model-checking is guilty!  
- The lacuna is that the model-checking specification by a temporal logic formula does not take the abstraction process into account;  
- This is common in the model-checking community:  
  - who really cares about how the finite model is obtained?  
  - the model is the truth, the specification is the truth, model-checking is only about their concordance!  
- In the program analysis community we (should) care: the programming language semantics is the referential (or should be).

Which (general) fix cures the problem in the context of program analysis?

Good news...  

Abstract Interpretation!
In the rest of the talk, I will explain...

- How to design a dataflow analysis specified by a temporal formula and an abstraction so that it is correct by construction;
- To do so I just have to show that:

**Model-checking is an abstract interpretation!**

and then instantiate for the dataflow analysis problem specified by a temporal formula;

---

Which Temporal Logic?

- Dataflow analysis people are used to reason on (merge over all) paths so we prefer a linear time logic (one path at the time) to the branching time logics considered by Steffen and Schmidt;
- Dataflow analysis people make no essential distinction between forward and backward analyses so that one should directly derive one from the other\(^{15}\); We introduce a new temporal reversal operator to make past and future completely symmetric;
- Dataflow analysis people make no essential distinction between minimal and maximal flow problem so that one should directly derive one from the other (using duality\(^{16}\)).

---

\(^{15}\) as in P. Cousot & R. Cousot. Systematic design of program analysis frameworks. 4th ACM-PDL, 289–292, 1979 where backward is just forward for the inverse transition system.

\textbf{RTL: Reversible Temporal Logic — Syntax}

\begin{itemize}
  \item \(l \in \mathcal{L}\) \quad \text{locations}
  \item \(\pi \in \mathcal{S}\) \quad \text{state predicates}
  \item \(\pi \ ::= \ \text{tt} \quad \text{true}\)
  \item \(\ \quad \pi \ ::= \ \text{ff} \quad \text{false}\)
  \item \(\ \quad \pi \ ::= \ \text{at}(l) \quad \text{at control predicate}\)
  \item \(\ \quad \ldots\)
  \item \(\tau \in \mathcal{T}\) \quad \text{transition predicates}
  \item \(\tau \ ::= \ \mathbf{1} \quad \text{identity}\)
  \item \(\ \quad \tau \ ::= \ \tau_1^{-1} \quad \text{inverse}\)
  \item \(\ \quad \ldots\)
\end{itemize}

\textbf{RTL: Reversible Temporal Logic — Semantic Domains}

\begin{itemize}
  \item \(\Sigma\) \quad \text{set of states}
  \item \(\triangleleft \Delta \equiv \varphi(\Sigma)\) \quad \text{semantic domain of state predicates}
  \item \(\vdash \triangle \equiv \varphi(\Sigma \times \Sigma)\) \quad \text{semantic domain of transition predicates}
  \item \(\trianglerighteq \Delta \equiv \mathcal{P}(\Sigma)\) \quad \text{paths}
  \item \(\trianglerighteq \Delta \equiv \mathcal{P}(\Sigma \times \Sigma)\) \quad \text{computations}
  \item \(\trianglerighteq \Delta \equiv \varphi(\mathcal{C})\) \quad \text{semantic domain of temporal formulae}
\end{itemize}

\textbf{RTL: Reversible Temporal Logic — Semantic functions}

\begin{itemize}
  \item \(\mathcal{S} \in \mathcal{S} \mapsto \triangleleft\) \quad \text{semantics of state predicates}
  \item \(\in \in \mathcal{S} \mapsto \Sigma \mapsto \Delta\) \quad \text{(isomorphic alternative)}
  \item \(\mathcal{T} \in \mathcal{T} \mapsto \trianglerighteq\) \quad \text{semantics of transition predicates}
  \item \(\in \mathcal{T} \mapsto (\Sigma \times \Sigma) \mapsto \trianglerighteq\) \quad \text{(isomorphic alternative)}
  \item \(\mathcal{F} \in \mathcal{F} \mapsto \trianglerighteq\) \quad \text{semantics of temporal formulae}
\end{itemize}
RTL: Reversible Temporal Logic — Semantics

\[ \mathfrak{G}[\text{init}] \triangleq \Sigma \]
\[ \mathfrak{G}[\text{init}] \triangleq \emptyset \]
\[ \ldots \]
\[ \mathfrak{G}[1] \triangleq \{(s, s) | s \in \Sigma\} \]
\[ \mathfrak{G}[\tau^{-1}] \triangleq (\mathfrak{G}[\tau])^{-1} \]
\[ \ldots \]
\[ \mathfrak{S}\pi \triangleq \{(i, \sigma) | \sigma_1 \in \mathfrak{G}[\pi]\} \]
\[ \mathfrak{S}\pi \triangleq \{(i, \sigma) | \langle \sigma_i, \sigma_{i+1} \rangle \in \mathfrak{G}[\pi]\} \]

RTL: Reversible Temporal Logic — Abbreviations

\[ \varphi_1 \land \varphi_2 \triangleq \neg (\neg \varphi_1 \lor \neg \varphi_2) \quad \text{conjunction} \]
\[ \varphi_1 \rightarrow \varphi_2 \triangleq \neg (\neg \varphi_1) \lor \varphi_2 \quad \text{implication} \]
\[ \diamond \varphi \triangleq \text{tt U } \varphi \quad \text{sometime or eventually} \]
\[ \Box \varphi \triangleq \neg \diamond \neg \varphi \quad \text{always or henceforth} \]
\[ \varphi_1 \Rightarrow \varphi_2 \triangleq \Box (\varphi_1 \rightarrow \varphi_2) \quad \text{entailment} \]
\[ \varphi_1 \Rightarrow \varphi_2 \triangleq (\varphi_1 \lor \varphi_2) \lor \Box \varphi_1 \quad \text{unless or waiting-for} \]
\[ \varphi \wedge \varphi \triangleq (\varphi_1 \lor \varphi_2) \lor \Box \varphi_1 \quad \text{previous} \]
\[ \varphi \triangleq (\Diamond \varphi) \quad \text{since} \]
\[ \varphi \triangleq (\varphi \lor \varphi) \quad \text{has always been} \]
\[ \Box \varphi \triangleq (\Diamond \varphi) \quad \text{once} \]
\[ \varphi_1 \Rightarrow \varphi_2 \triangleq (\varphi_1 \lor \varphi_2) \quad \text{back to} \]

RTL: Reversible Temporal Logic — Semantics (Continued)

\[ \mathfrak{S}[\diamond \varphi] \triangleq \{(i, \sigma) | \langle i+1, \sigma \rangle \in \mathfrak{S}[\varphi]\} \]
\[ \mathfrak{S}[\varphi_1 \lor \varphi_2] \triangleq \{(i, \sigma) | \exists k \geq i : \langle k, \sigma \rangle \in \mathfrak{S}[\varphi_2] \land \forall j : i \leq j < k : \langle j, \sigma \rangle \in \mathfrak{S}[\varphi_1]\} \]
\[ \mathfrak{S}[\neg \varphi] \triangleq \mathfrak{S}[\varphi] \cup \mathfrak{S}[\varphi] \quad \text{(also written } \neg \mathfrak{S}[\varphi]) \]
\[ \mathfrak{S}[\neg \varphi] \triangleq \mathfrak{S}[\varphi] \cup \mathfrak{S}[\varphi] \quad \text{(also written } \neg \mathfrak{S}[\varphi]) \]
\[ \mathfrak{S}[\neg \varphi] \triangleq \{(\neg \sigma, j) \sigma \in \mathfrak{S}[\varphi] \} \]

Implication and Equivalence of Temporal Formulae

\[ \varphi_1 \Leftrightarrow \varphi_2 \triangleq \mathfrak{S}[\varphi_1] \subseteq \mathfrak{S}[\varphi_2] \quad \text{Implication} \]
\[ \varphi_1 \equiv \varphi_2 \triangleq \mathfrak{S}[\varphi_1] = \mathfrak{S}[\varphi_2] \quad \text{Equivalence} \]

(\equiv \text{ is a congruence on } \mathcal{F}.)

Dagstuhl Seminar on Program Analysis, April 11-16, 1999
© P. R. Cousot
RTL: Reversible Temporal Logic — Fixpoint Semantics

\[ \mathfrak{S}(\varphi_1 U \varphi_2) = \text{lfp} \lambda X : \mathfrak{S}[\varphi_1] \cup (\mathfrak{S}[\varphi_1] \cap \text{pre}[X]) \quad (1) \]

where

\[ \text{pre}[X] \triangleq \{ \langle i - 1, \sigma \rangle \mid \langle i, \sigma \rangle \in X \} \]

so that \( \mathfrak{S}[\varphi] = \text{pre}[\mathfrak{S}[\varphi]] \) whence:

\[ \varphi_1 U \varphi_2 \equiv \varphi_2 \lor (\varphi_1 \land \diamond(\varphi_1 U \varphi_2)) \]

(Continued)

All other cases directly follows by:

- **Lattice duality:**
  \[ \text{gfp} \subseteq F = \text{lfp} \supseteq F \]

- **Park negation duality** \(^{17}\):
  \[ \neg(\text{gfp} F) = \text{lfp} \lambda X : \neg F(\neg X) \]

- **Reversal duality:**
  \[ (\text{lfp} F)^\ast = \text{lfp} \lambda X : (F(X^\ast))^\ast \]

---

Small Step Operational Semantics

Transition predicate:

\[ \text{trans} \in \mathcal{T} \]

Final states are the only possible blocking states and they are repeated forever:

\[ \text{final} \triangleq \neg \text{trans} \land \Box 1 \]

and symmetrically for the initial states:

\[ \text{init} \triangleq \Box (\neg \text{trans}) \land \square 1 \]

\(^{17}\) in the complete boolean lattice \( \Box \).
Trace-Based Operational Semantics

- We define the forward and backward transition predicates as:

\[ \text{ftrans, btrans} \in T \]
\[ \text{ftrans} \triangleq \text{trans} \lor \text{final} \]
\[ \text{btrans} \triangleq \ominus \text{trans} \lor \text{init} \]

- The trace-based operational semantics of a program is (including non-termination and symmetrically non initialization):

\[
\begin{align*}
\text{fsem} & \triangleq \bigcup \text{ftrans} & \text{forward operational semantics} \\
\text{bsem} & \triangleq \bigcap \text{btrans} & \text{backward operational semantics} \\
\text{sem} & \triangleq \text{fsem} \land \text{bsem} & \text{operational semantics}
\end{align*}
\]

In one definition...

We use the following notation:

\[
\langle M, \preceq \rangle \xrightarrow{\alpha} \langle L, \sqsubseteq \rangle
\]

for Galois connections:

\[
\forall x \in M, y \in L : \alpha(x) \sqsubseteq y \iff x \preceq \gamma(y)
\]

We write \( \langle M, \preceq \rangle \xleftarrow{\alpha} \langle L, \sqsubseteq \rangle \) when \( \alpha \) is surjective, \( \langle M, \preceq \rangle \xleftarrow{\alpha} \langle L, \sqsubseteq \rangle \) when \( \alpha \) is injective and \( \langle M, \preceq \rangle \xleftrightarrow{\alpha} \langle L, \sqsubseteq \rangle \) when \( \alpha \) is bijective.

... and one theorem

**Theorem 1** If \( \langle M, \preceq, 0, \lor \rangle \) is a cpo, the pair \( \langle \alpha, \gamma \rangle \) is a Galois connection \( \langle M, \preceq \rangle \xrightarrow{\alpha} \langle L, \sqsubseteq \rangle \), \( \mathcal{F} \in M \xrightarrow{\text{mon}} M \) and \( \mathcal{G} \in L \xrightarrow{\text{mon}} L \) are monotonic and satisfy the semi-commutation condition

\[
\forall y \in L : \gamma(y) \preceq \text{lfp} \xrightarrow{\preceq} \mathcal{F} \Rightarrow \alpha \circ \mathcal{F} \circ \gamma(y) \sqsubseteq \mathcal{G}(y)
\]

or equivalently

\[
\forall x \in M : \gamma \circ \alpha(x) \preceq \text{lfp} \xrightarrow{\preceq} \mathcal{F} \Rightarrow \alpha \circ \mathcal{F}(x) \sqsubseteq \mathcal{G} \circ \alpha(x)
\]

or equivalently

\[
\forall y \in L : \gamma(y) \preceq \text{lfp} \xrightarrow{\preceq} \mathcal{F} \Rightarrow \mathcal{F} \circ \gamma(y) \preceq \gamma \circ \mathcal{G}(y)
\]

then

\[
\text{lfp} \xrightarrow{\preceq} \mathcal{F} \preceq \gamma(\text{lfp} \xrightarrow{\preceq} \mathcal{G})
\]

and equivalently \( \alpha(\text{lfp} \xrightarrow{\preceq} \mathcal{F}) \sqsubseteq \text{lfp} \xrightarrow{\preceq} \mathcal{G} \).
6. The Model-Checking Abstractions

Boolean Universal Abstraction

The boolean universal satisfaction abstraction $\alpha^\forall[\varphi](M)$ checks all computations of the model $M$ for the temporal formula $\varphi$:

$$
\begin{align*}
\alpha^\forall & \in \mathcal{F} \mapsto \mathcal{T} \mapsto \mathbb{B} \\
\alpha^\forall[\varphi](M) & \triangleq M \mid \varphi \\
\gamma^\forall & \in \mathcal{F} \mapsto \mathcal{T} \mapsto \mathbb{B} \\
\gamma^\forall[\varphi](b) & \triangleq (b ? \mathcal{F} \varphi] \land C)
\end{align*}
$$

This is a generic Galois connection parameterized by the temporal formula $\varphi$:

$$
(\mathcal{C}, \subseteq) \leq \frac{\gamma^\forall}{\alpha^\forall} \frac{\alpha^\forall}{\gamma^\forall} (\mathcal{C}, \subseteq)
$$

By Dualization: Four Different Abstractions

$$
\begin{align*}
\alpha^\forall[\varphi](M) & \triangleq M \subseteq \mathcal{F} \varphi] \\
\alpha^\exists[\varphi](M) & \triangleq \neg \alpha^\forall[\neg \varphi](M) \\
\alpha^\exists [\varphi](M) & \triangleq \neg \alpha^\exists[\neg \varphi](M) \\
\alpha^\exists [\varphi](M) & \triangleq \neg \alpha^\exists[\neg \varphi](M)
\end{align*}
$$

More on the Existential Satisfaction Abstraction

$$
\begin{align*}
\alpha^\exists[\varphi](M) & \triangleq \neg \alpha^\forall[\neg \varphi](M) \\
& \quad = \ldots \\
& \quad \text{easing calculation} \\
& \quad = (M \cap \mathcal{F} \varphi] \neq \emptyset
\end{align*}
$$

The boolean existential satisfaction abstraction $\alpha^\exists[\varphi](M)$ checks that some computations of the model $M$ do satisfy the temporal formula $\varphi$. 
State Static Partitionning

- Let’s check the model for each state;
- State projection:
  \[ M_{ls} \triangleq \{ (i, \sigma) \in M \mid \sigma_i = s \} \]
- Static state partitioning abstraction:
  \[ \alpha_\Sigma(M) \triangleq \lambda s. M_{ls} \quad \gamma_\Sigma(S) \triangleq \bigcup_{s \in \Sigma} S(s)_{ls} \]
- Galois isomorphism \(^\dagger\):
  \[ (\tau_i, \subseteq) \xleftarrow{\frac{\gamma_\Sigma}{\alpha_\Sigma}} \bigl( \prod_{s \in \Sigma} \tau_{ls}, \subseteq \bigr) \]

\(^\dagger\) no information is lost.

Location Static Partitionning

- Let’s now have a coarser partition according to locations (program points, call strings, contours, ...);
- The locations are assumed to cover all states:
  \[ \forall s \in \Sigma : \exists l \in L : s \in S_{\text{at}(l)} \]  \( (3) \)

State Partitionned Satisfaction Abstractions

\[
\begin{align*}
\alpha_{\Sigma, l}^\triangle & \triangleq \alpha_{\Sigma, l}^\triangle \circ \alpha_{\Sigma} \\
\gamma_{\Sigma, l}^\triangle & \triangleq \gamma_{\Sigma} \circ \gamma_{\Sigma, l}^\triangle \\
\alpha_{\Sigma, l}^\triangledown & \triangleq \alpha_{\Sigma, l}^\triangledown \circ \alpha_{\Sigma} \\
\gamma_{\Sigma, l}^\triangledown & \triangleq \gamma_{\Sigma} \circ \gamma_{\Sigma, l}^\triangledown \\
\alpha_{\Sigma, l}^\triangledown & \triangleq \alpha_{\Sigma, l}^\triangledown \circ \alpha_{\Sigma} \\
\gamma_{\Sigma, l}^\triangledown & \triangleq \gamma_{\Sigma} \circ \gamma_{\Sigma, l}^\triangledown \\
\alpha_{\Sigma, l}^\triangledown & \triangleq \alpha_{\Sigma, l}^\triangledown \circ \alpha_{\Sigma} \\
\gamma_{\Sigma, l}^\triangledown & \triangleq \gamma_{\Sigma} \circ \gamma_{\Sigma, l}^\triangledown \\
\end{align*}
\]

\(^\triangledown\) need not be finite!

Location Static Partitionning Abstraction

- Location partitioning:
  \[ \alpha_L^\triangledown \triangleq \lambda f, l. \bigwedge_{s \in S_{\text{at}(l)}} f(s) \quad \langle \Sigma \mapsto \exists \rangle \xleftarrow{\frac{\gamma_L^\triangledown}{\alpha_L^\triangledown}} \langle L \mapsto \exists \rangle \]
  \[ \gamma_L^\triangledown \triangleq \lambda g, l. \bigvee_{s \in S_{\text{at}(l)}} g(l) \]
- Dually \( \alpha_L^\triangle \) and \( \gamma_L^\triangle \), such that:
  \[ \langle \Sigma \mapsto \exists \rangle \xleftarrow{\frac{\gamma_L^\triangledown}{\alpha_L^\triangledown}} \langle L \mapsto \exists \rangle \]
7. The Calculational Design of the Abstract Model-Checking Algorithms by Abstract Interpretation

Location Partitioned Satisfaction Abstractions

\[\begin{align*}
\alpha_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \alpha_{\mathcal{L}}^{\mathcal{C}} \\
\gamma_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \gamma_{\mathcal{L}}^{\mathcal{C}}
\end{align*}\]

\[\begin{align*}
\langle \mathcal{F}, \subseteq \rangle \xrightarrow{\gamma_{\mathcal{L}}^{\mathcal{C}}} \langle \mathcal{F} \mapsto \mathcal{F}, \iff \rangle
\end{align*}\]

\[\begin{align*}
\alpha_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \alpha_{\mathcal{L}}^{\mathcal{C}} \\
\gamma_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \gamma_{\mathcal{L}}^{\mathcal{C}}
\end{align*}\]

\[\begin{align*}
\langle \mathcal{F}, \subseteq \rangle \xrightarrow{\alpha_{\mathcal{L}}^{\mathcal{C}}} \langle \mathcal{F} \mapsto \mathcal{F}, \implies \rangle
\end{align*}\]

\[\begin{align*}
\alpha_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \alpha_{\mathcal{L}}^{\mathcal{C}} \\
\gamma_{\mathcal{L}}^{\mathcal{C}} & \triangleq \mathcal{C} \gamma_{\mathcal{L}}^{\mathcal{C}}
\end{align*}\]

\[\begin{align*}
\langle \mathcal{F}, \subseteq \rangle \xrightarrow{\alpha_{\mathcal{L}}^{\mathcal{C}}} \langle \mathcal{F} \mapsto \mathcal{F}, \iff \rangle
\end{align*}\]

Location Partitioned Existential Satisfaction Abstractions of the Forward Trace Semantics

- Assume we are interested in checking for any location \( l \in \mathcal{L} \) whether there is a computation from \( l \) such that \( \tau_1 \) will hold until eventually \( \tau_2 \) does hold;
- Formally we want to calculate/compute:

\[\exists_{\mathcal{L}}^3 [\tau_1 \cup \tau_2] [\overline{\mathcal{S}[fsem]}] \tag{5}\]

- Design strategy:
  - Express this as the abstraction of a fixpoint (see (1))
  - Use the fixpoint approximation Theorem 1 with the Galois connection (4) (or dual forms).

Fixpoint Abstraction

\[\exists_{\mathcal{L}}^3 [\tau_1 \cup \tau_2] [\overline{\mathcal{S}[fsem]}] = \ldots \text{ skipping 12 lines of hand computation using (1)}\]

\[= \exists_{\mathcal{L}}^3 [fsem][\{p \in X. \overline{\mathcal{S}[\tau_2]} \cup (\overline{\mathcal{S}[\tau_1]} \cap \text{pre}[X])\}]\]
Calculating the Semi-commuting Abstract Transformer

We assume:
\[
X = \mathcal{S}_2 \cup (\mathcal{S}_1 \cap \text{pre}[X]) \\
\psi = \text{ftrans} \land \bigcirc \psi
\]

and calculate:
\[
\alpha^3_L[\text{fsem}](\mathcal{S}_2 \cup (\mathcal{S}_1 \cap \text{pre}[X])) \\
\implies \ldots \text{ skipping 25 lines of hand computation} \\
= F^\delta(\alpha^3_L[\text{fsem}](X))
\]

so that Theorem 1 with the Galois connection \((4)\), we conclude \ldots\ldots

8. Application to Live-Variables Analysis

“In live-variable analysis we wish to know for variable \(x\) and point \(p\) whether the value of \(x\) at \(p\) could be used along some path in the flow graph starting at \(p\). If so, we say \(x\) is live at \(p\); otherwise \(x\) is dead at \(p\)” [9, p. 631].

References


Live-Variables Analysis is a Sound Location Partitionned Model-Checking Existential Abstraction of the Trace Semantics

- For a single flowchart node:
  \[
  \text{mod}(x) : \text{transitions potentially modifying variable } x \\
  \text{used}(x) : \text{transitions definitively using the value of variable } x
  \]

- Along one path: Variable \(x\) is live at the origin of a computation if it will not be modified until it is used:
  \[
  \text{isLive}(x) \triangleq (\neg \text{mod}(x)) \lor \text{used}(x)
  \]

- Merge over some path: Variable \(x\) is live at location \(l\) if and only if it is live on some computation path starting from that location:
  \[
  \text{Live}(x) \triangleq \alpha^3_L[\text{isLive}(x)](\mathcal{S}_2[\text{fsem}])
  \]
The Classical Live-Variables Dataflow Equations are Definitely **Sound**

\[
\text{Live}(x) \iff \text{Live}^2(x)
\]

\[
\triangleq \ \text{lfp} \Rightarrow \lambda X \cdot \lambda l \cdot (\exists l' : \mathcal{L}^3[\text{used}(x)](l, l')) \lor
\]

\[
\bigvee_{l' \in \text{succ}(l)} \mathcal{L}^3[\neg \text{mod}(x)](l, l') \land X(l')
\]

- Note: \(\Rightarrow\), not \(\Leftrightarrow\)!
- Hence \(\text{Dead}^3(x) \triangleq \neg \text{Live}^2(x) \Rightarrow \neg \text{Live}(x) \triangleq \text{Dead}(x)\);
- So Dave forgives the practitioners!
- And now the practitioners forgive Dave!

---

**So what is Unsound?**

- Live-variables analysis is a location partitioned model-checking existen-
tial/merge over some paths abstraction of the trace semantics;
- It is unsound to reason on live-variables analysis as if it were a location
partitioned model-checking universal/merge over all paths abstraction
of the trace semantics;
- Model-checking is unsound in that it does not make explicit which of
the abstractions is involved (and there are many such as \(\alpha^3_L\), \(\alpha^3_L\), \(\alpha^3_L\)
and \(\alpha^3_L\)).

---

**Conclusion (on Live-Variables Analysis Being Unsound)**

- Data-flow analysis was **not** guilty;
- Model-Checking was **guilty**, by not taking the abstraction process into
account;
- Abstract Interpretation was the **rescuer**:
  - Which abstracts are used is made explicit;
  - Abstraction is used for the formal calculational design of correct al-
gorithms;
  - Abstract Interpretation provides an unambiguous understanding of
what is going on.
Conclusion (on Model-Checking Design by Abstract Interpretation)

- In this talk, we have chosen to handle a striking example rather than present formally the full theory;
- In full generality, we have to handle any $\alpha[\varphi](\delta[\psi])$ for $\alpha$ in the cube of abstractions and all possible RTL formulae $\varphi, \psi \in \mathcal{F}$ (to get interleaved fixpoints);
- A more general and debatable question:
  
  Is model-checking of any practical use in program static analysis?  

---

Abstract Interpretation...

In one single formal framework, abstract interpretation lets you meta-understand the foundational aspects of:

- Data flow analysis;
- Constraint based program analysis;
- Types and effect systems;
- ...  
- Relationships between semantics;
- ...

...and now:

- Model-Checking;
- Including its use in the design of sound program analyzes!
Abstract Interpretation...

Abstract interpretation is a theory of discrete approximation of semantics, not only a peculiar static program analysis method.

More on the Calculational Design of Abstract Interpretations

A complete calculational design of an abstract interpreter:

P. Cousot.
Calculational System Design
chapter "The Calculational Design of a Generic Abstract Interpreter".

and its OCAML implementation:

P. Cousot.
The Marktoberdorf'98 generic abstract interpreter.
http://www.dmi.ens.fr/~cousot/Marktoberdorf98.shtml
November 1998.

THE END