Patterns in Program Executions

There are many interesting patterns within programs executions

- use-def pairs

  `.*; def(x); [- def(x)]*; use(x)`

- api usage

  `[- open(f)]*; (open(f); (read(f)|write(f))*; close(f))*`

where `.` matches any symbol, `;` denotes sequencing, `|` denotes disjunction, `*` denotes zero-closure, and `[- x]` means everything *except* `x`.

Encoding Patterns as Automata

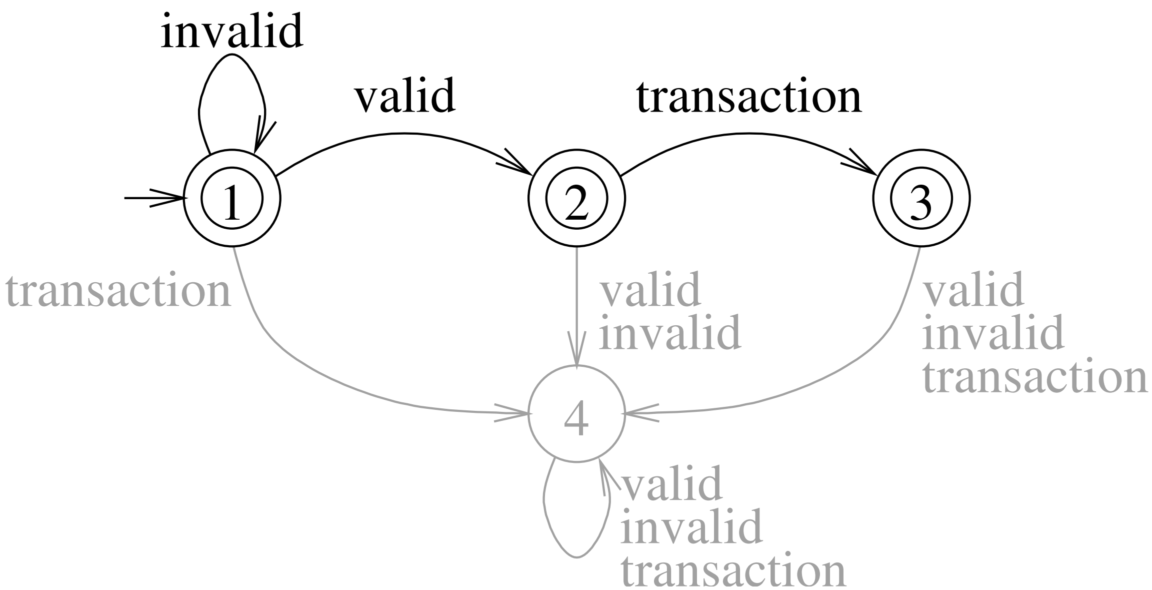
A deterministic finite-state *property* automaton is

$$P = (\Sigma, S, \delta, A, s_0, s_{\textit{trap}})$$

where

- Σ is the alphabet of the property,
- $S = \{s_0, s_1, \ldots, s_k\}$ is the set of property automaton states that represent equivalence classes of strings over Σ,
- $\delta : S \times \Sigma \rightarrow S$ is the total state transition function,
- $A \subseteq S$ is the set of accepting states,
- $s_0 \in S$ is the unique start state, and
- $s_{\textit{trap}} \in S$ is the unique trap state.

An Example Property Automaton



$$S = \{1, 2, 3, 4\}, A = \{1, 2, 3\}, s_0 = 1, s_{\mathit{trap}} = 4$$
$$\delta = \{(1, \texttt{invalid}, 1), (1, \texttt{valid}, 2), (1, \texttt{transaction}, 4), \ldots\}$$

Questions about Patterns

One can formulate a variety of questions about how a pattern corresponds to the set of program paths

- Do all terminating paths correspond to the pattern?
- Does some path reaching a program point correspond to a pattern?
- If a pattern matches a path, what are the points on the path where the match occured?

These can be answered using variations of a *state propagation* flow analysis.

State Propagation - Program to Property Mapping

Let *label*(l) map statements onto $\Sigma \cup \uparrow$, where $\uparrow$ means that there is no label in Σ for l.

- e.g., *label*$($`[x := ...]`$^l) =$`def(x)`
- e.g., *label*$($`[...]`$^l) =$`valid` if l is the first node in the true branch of `[if (isValid())]`.

State Propagation - Summarizing Paths

A state, $s \in S$, of a property automaton symbolically encodes the set of sequences $\sigma \in \Sigma^*$ such that $\Delta(s_0, \sigma) = s$.

- where $\Delta(s_0, \sigma)$ is the composition of δ applied to the symbols in σ.

For the example, if $\sigma = \mathtt{invalid}, \mathtt{valid}, \mathtt{transaction}$

- $\Delta(s_0, \sigma) = \delta(\delta(\delta(s_0, \mathtt{invalid}), \mathtt{valid}), \mathtt{transaction}) = 3$

State Propagation Lattice

$\mathcal{P}(S)$

- $\sqsubseteq = \subseteq$
- $\sqcup = \cup$
- $\perp = \emptyset$

So, we are working with a complete lattice

State Propagation Transfer Functions

Let

$$f(l, s) = \begin{cases} \Delta(s, \textit{label}(l)) & \text{if } \textit{label}(l) \in \Sigma \\ s & \text{if } \textit{label}(l) \notin \Sigma \end{cases}$$

then the transfer function for l is

$$f_l(S) = \bigcup_{s \in S} f(l, s)$$

Why do we use Δ and not δ in f_l?

State Propagation as a Framework Instance

- $L = \mathcal{P}(S)$
- $\sqsubseteq = \subseteq$
- $\bigsqcup = \cup$
- $\bot = \emptyset$
- $\iota = \{s_0\}$
- $E = \{init(G)\}$, where G is the flow graph
- $F = flow(G)$ (forward flow problem)
- $\mathcal{F} = \{f_l : L \rightarrow L \mid \forall l \in \text{Lab}_* \text{as defined above}\}$

Function space properties

Basic requirements

- $\mathcal{F}$ contains the identity function (self-loop in Δ or *label*$(l) = \uparrow$)
- $\mathcal{F}$ is closed under composition ($\Delta(\Delta(s, a), b) = \Delta(s, ab)$)

Distributivity

$$\forall f \in F \forall l_1, l_2 \in L : f(l_1 \cup l_2) = f(l_1) \cup f(l_2)$$

Since f operates component-wise on L

This is a slight reformulation and improvement on the result in the paper.

Recording Selected History - Lattice

Distinguish certain symbols as *label generators* and when those are encountered add the label to propagated state

- $L = \mathcal{P}(S \times (\text{Lab}_* \cup \uparrow))$ (propagate pairs)
- $\sqsubseteq = \subseteq$
- $\bigsqcup = \cup$
- $\bot = \emptyset$
- $\iota = \{(s_0, \uparrow)\}$

Recording Selected History - Transfer Functions

$$f(l,s) = \begin{cases} (\Delta(\pi_1(s), \textit{label}(l), l) & \text{if } \textit{label}(l) \in \Sigma \wedge \\ & \quad \textit{label}(l) \in \text{generators} \\ (\Delta(\pi_1(s), \textit{label}(l), \pi_2(s)) & \text{if } \textit{label}(l) \in \Sigma \\ & \quad \textit{label}(l) \notin \text{generators} \\ s & \text{if } \textit{label}(l) \notin \Sigma \end{cases}$$

where π_i projects the i^{th} component from a tuple.

Multiple Pattern Matching

Consider the *use-def* pair pattern

- Write down the property automaton
- How would you define *generators* to record the label of the definition for each use that is reached?

Can you define a variant of this analysis to calculate *use-def* pairs for all program variables?