Delta-oriented Monitor Specification

Eric Bodden, Kevin Falzon Ka I Pun, Volker Stolz

EC-SPRIDE, Darmstadt

Universitetet i Oslo

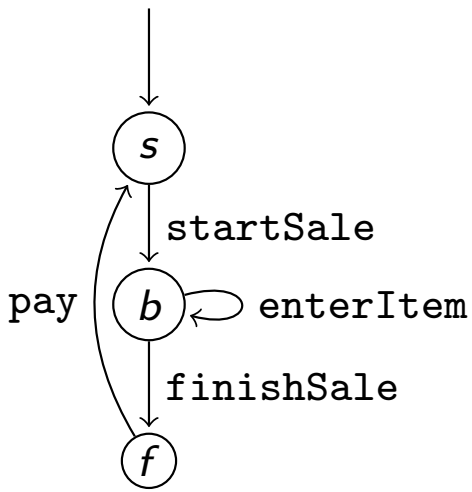
October 2012

Motivation

- What is RV?
- Does ABS support infrastructure for RV?
  (before: aspect-oriented programming)
- How do we define variable protocols for different products?
- How do we make protocols part of the spec/file?
  (machine readable protocols)

Runtime Verification

Use *protocol* to describe permitted API use. Here: CoCoME.

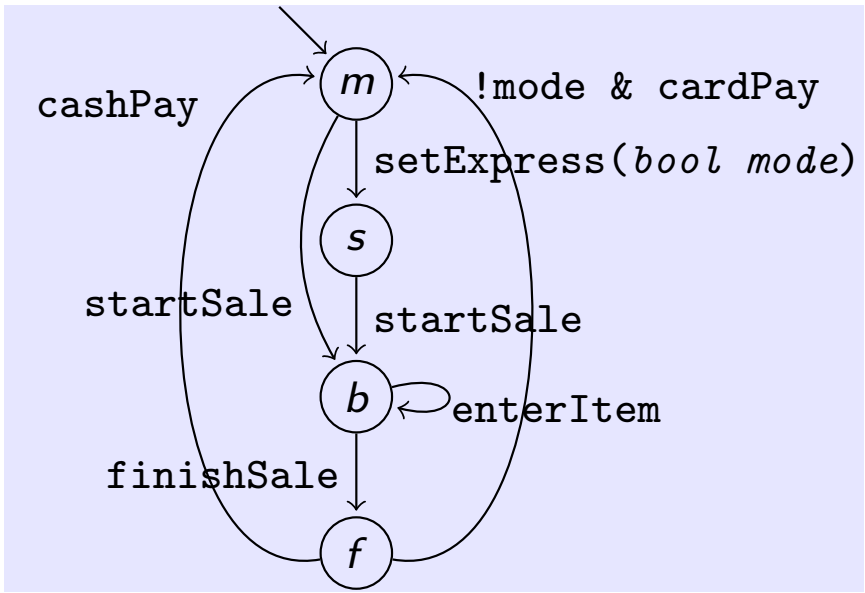



```
interface Cashdesk {
  Unit startSale();
  Unit enterItem(Int code, Int qty);
  Unit finishSale();
  Int pay(Int given);
}
```

Transition labels: method names (of a *single* class)

Guards and Binders

- Transitions of form: $g\&m(x_0, \ldots, x_n)$
- Guard g may refer to *previously* bound variable
- Variables x_i bound to actual parameters during run
- Requires wellformedness of automaton/LTS



Formalization (Automaton)

$\Theta := \mathtt{VAR} \to \mathtt{VAL}$: set of variable bindings over values.

- *Base automaton* $\mathcal{M}$: $\langle Q, \Sigma \times \overrightarrow{\mathtt{VAR}}, q_0, \theta_0, \Gamma \rangle$
- Alphabet *with a list of formal parameters*
- Initial variable binding $\theta_0 \in \mathtt{VAR} \to \mathtt{VAL}$

Transitions $\Gamma : Q \times (\Sigma \times \overrightarrow{\mathtt{VAR}}) \times (\Theta \to \mathbb{B}) \times ((\Theta \times \overrightarrow{\mathtt{VAL}}) \to \Theta) \times Q$

Single transition:

$$(q, \theta) \xrightarrow{e(c_0,\ldots,c_n)}_{\mathcal{M}} (q', \theta') := \quad (q, e(x_0, \ldots, x_n), \mathit{guard}, \mathit{binding}, q') \in \Gamma \\ \wedge\ \mathit{guard}(\theta) \wedge\ \mathit{binding}(\theta, \vec{c}) = \theta'$$

Formalization (Wellformedness)

Need classical *def-before-use* analysis on variables.
Assume $vars : \Gamma \to 2^{\mathtt{VAR}}$.

Transition $\langle S, a, g, b, T \rangle$ is *wellformed*, iff $vars(g) \subseteq defs_{\mathcal{M}}(S)$
where $defs_{\mathcal{M}}(S) : Q \to 2^{\mathtt{VAR}}$

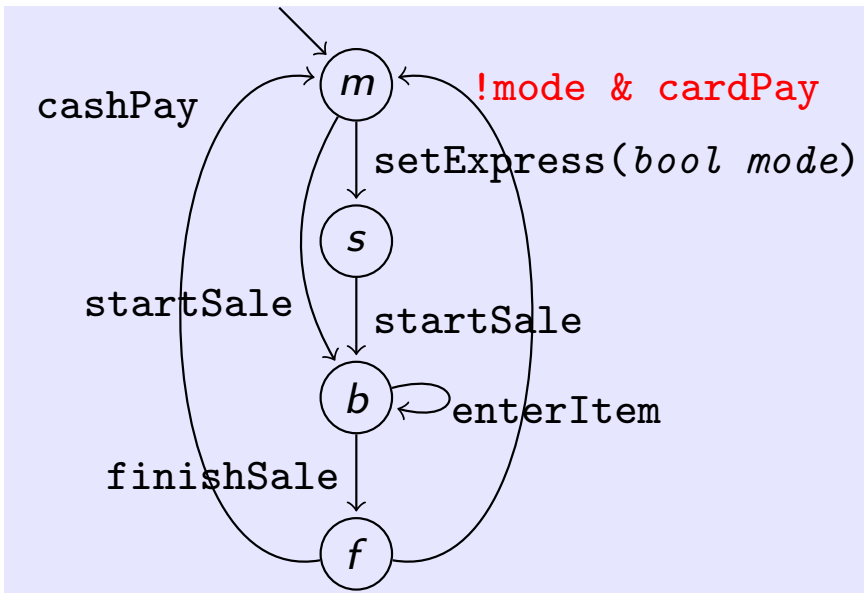
$$defs_{\mathcal{M}}(S) := \begin{cases} dom(\theta_0) & \text{iff } s = q_0; \\ \bigcap_{\gamma \in \Gamma} (defs_{\mathcal{M}}(S^p) \cup \{x_0, \ldots, x_n\}) & \text{otherwise} \end{cases}$$

$$\gamma = (S^p, e(x_0, \ldots, x_n), g, \theta, S)$$

Formalization (Wellformedness)

Need classical *def-before-use* analysis on variables.
Assume $vars : \Gamma \to 2^{\mathtt{VAR}}$.

Transition $\langle S, a, g, b, T \rangle$ is *wellformed*, iff $vars(g) \subseteq defs_{\mathcal{M}}(S)$



So much for RV

Back to Software Engineering!

ABS and Deltas

Our Cashdesk: simple OO. (Actually not even OO.)

```
Unit enterItem(Int code, Int qty) {
    Item item = store.lookup(code);
    total = total + qty*price(item);
    items = Cons(item,items);
}
```

More interesting: variability with Deltas.

Variability

Optional payment with credit card:

```
delta Credit
  modifies class Cashdesk
    adds Bool cardPay(CCData cc)
      { return store.authorize(cc); }
    adds Int cashPay(Int given)
      { return pay(given); }
```

Express mode with restricted functionality:

```
delta Express(Int k)
  modifies class Cashdesk
    adds Bool mode = False;
    adds Unit setExpress(Bool m) { mode = m; }
    modifies Unit enterItem(Int code, Int qty) {
      // You are allowed to buy k items in ExpressMode
      if (mode && length(items) == k) { assert False;}
                                    else { original(code,qty);}
```

Deltas are not oblivious—`Express`-delta must know about *items*.

Our Product Line

Special case from use case:

```
delta ExpressCC
  modifies class Cashdesk
    modifies Bool cardPay(CCData cc) {
      assert ~mode; // Not allowed in express mode
      return original(cc);
    }
```

```
productline CoCoME
  features Express, Credit;
  delta Credit when Credit;
  delta Express(10) when Express;
  delta ExpressCC after Credit
          when Express && Credit;
```

```
product Credit(Credit);
product Ex(Express);
product CCEx(Express,Credit);
```

Back to Protocols!

Observation:
Monitoring becomes a *feature* just like other variabilities.

How to describe *protocols* for different *products*?

Alternatives:

- Give new protocol per product
  Disadvantage: presumably unwieldily large
- Give base protocol,
  attach protocol-change to delta $\surd$

Formalization (Delta automaton)

Delta automaton $\Delta := \left\langle Q^\Delta, \Sigma^\Delta \times \overrightarrow{\texttt{VAR}}, q_0^\Delta, \theta_0^\Delta, \Gamma_+^\Delta, \Gamma_-^\Delta \right\rangle$ with:

- Q^Δ new introduced states,
- $\Sigma^\Delta \times \overrightarrow{\texttt{VAR}}$ new symbols,
- q_0^Δ an optional redefined start state,
- θ_0^Δ new initial bindings,
- Γ_+^Δ and Γ_-^Δ transitions added/removed.

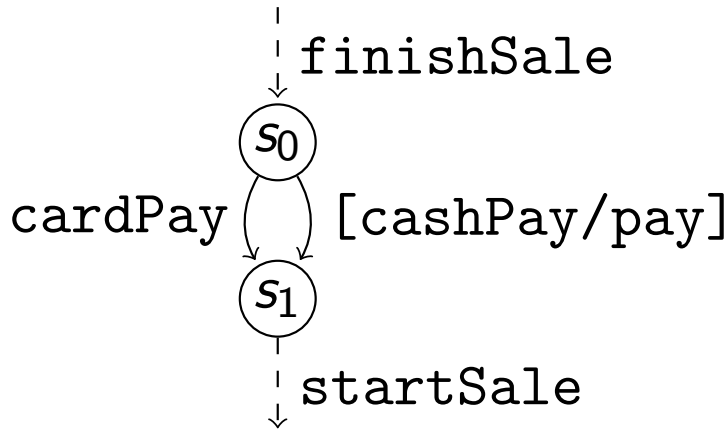
Given base automaton $\mathcal{M} = \left\langle Q^\mathcal{M}, \Sigma^\mathcal{M}, q_0^\mathcal{M}, \Gamma^\mathcal{M} \right\rangle$,
delta automaton $\Delta = \left\langle Q^\Delta, \Sigma^\Delta, q_0^\Delta, \Gamma_+^\Delta, \Gamma_-^\Delta \right\rangle$.

Application of Δ to $\mathcal{M}$: $\mathcal{M}' := \mathcal{M} \downarrow \Delta$

$$
\begin{array}{rcl}
Q' & := & Q^\mathcal{M} \cup Q^\Delta \\
\Sigma' \times \overrightarrow{\texttt{VAR}} & := & \Sigma^\mathcal{M} \times \overrightarrow{\texttt{VAR}} \cup \ \Sigma^\Delta \times \overrightarrow{\texttt{VAR}}, \\
q_0' & := & q_0^\mathcal{M} \text{ if } q_0^\Delta = \bot, q_0^\Delta \text{ otherwise} \\
\theta_0' & := & \theta_0^\mathcal{M} \text{ if } \theta_0^\Delta = \bot, \text{ otherwise:} \\
 & & \quad \lambda c.(\texttt{case } \theta_0^\Delta(c) = \bot \Rightarrow \theta_0^\mathcal{M}(c); \texttt{otherwise}, \theta_0^\Delta(c)) \\
\Gamma' & := & (\Gamma^\mathcal{M} \cup \Gamma_+^\Delta) - \Gamma_-^\Delta
\end{array}
$$

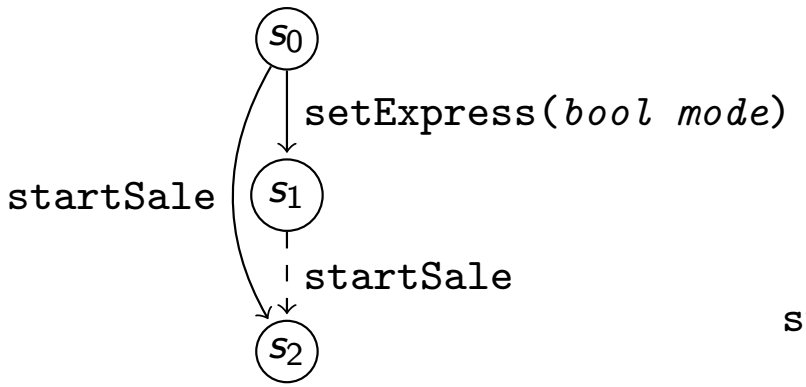
(Check resulting automaton for wellformedness.)

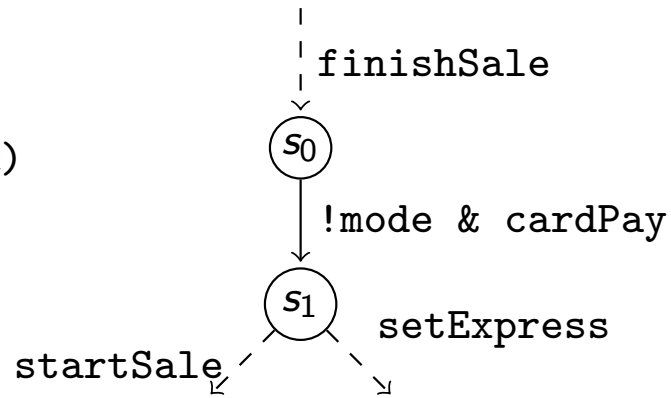
Protocol Deltas (Credit Card Payment)



$$
\begin{aligned}
\Delta_{CC} := \langle\, & \emptyset, && \text{no new state} \\
& \{\texttt{cashPay}, \texttt{cardPay}\}, && \text{new alphabets} \\
& \bot, \bot, && \text{no new initial state/symbols} \\
& \{(f, \texttt{cashPay}, \lambda s.\mathit{true}, \lambda(s, \vec{c}).s, s), && \text{transitions added} \\
& \ (f, \texttt{cardPay}, \lambda s.\mathit{true}, \lambda(s, \vec{c}).s, s)\}, && \\
& \{(f, \texttt{pay}, \lambda s.\mathit{true}, \lambda(s, \vec{c}).s, s)\}\rangle && \text{transition removed}
\end{aligned}
$$

Protocol Deltas (Express Mode/Special Case)





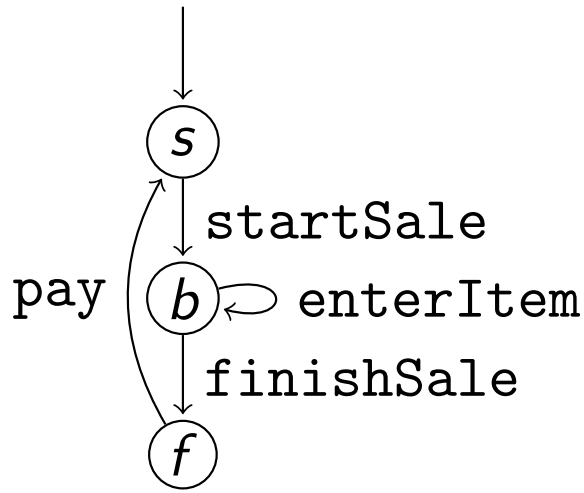
$\langle \{m\}, \{\texttt{setExpress}\}, m,$
$\lambda c.(\texttt{case}\ c = \text{“}mode\text{”} \Rightarrow true),$
$\{(m, \texttt{setExpress}, \lambda s.true, \lambda(s, x).$
$(\lambda y.(\texttt{case}\ y = \text{“}mode\text{”} \Rightarrow x;$
$\text{otherwise} \Rightarrow s(y))), s),$
$(m, \epsilon, \lambda s.true, \lambda(s, \vec{c}).s, b)\},$
$\emptyset\rangle$

$\langle \emptyset, \emptyset, \bot,$
$\bot,$
$\{(f, \texttt{cardPay}, \lambda\ s.(\neg s(\text{“}mode\text{”})), \lambda(s, \vec{c}).s, m)\},$
$\{(f, \texttt{cardPay}, \lambda s.true, \lambda(s, \vec{c}).s, m)\}\rangle$

Note: “before” advice relocates initial state!

Generating Monitor Deltas

Straightforward. Base monitor:



(More interesting if more than one incoming edge. . .)

Generating Monitor Deltas

Straightforward. Base monitor:

```
data State = Init | Buying | Finished;

delta Monitor
 modifies class Cashdesk
  adds State state = Init;
  modifies Unit startSale() {
    if (state == Init) { original(); state = Buying; }
      else { assert False; }}
  modifies Unit enterItem(Int code, Int qty)
    if (state == Buying) { original(code,qty); }
      else {assert False; }}
  modifies Unit finishSale()
    if (state == Buying) {original(); state = Finished;}
      else {assert False;}}
  modifies Int pay(Int given)
    Int res = -1;
    if (state == Finished) {res = original(given); state = Init;}
      else {assert False;} /* endif*/ return res; }
```

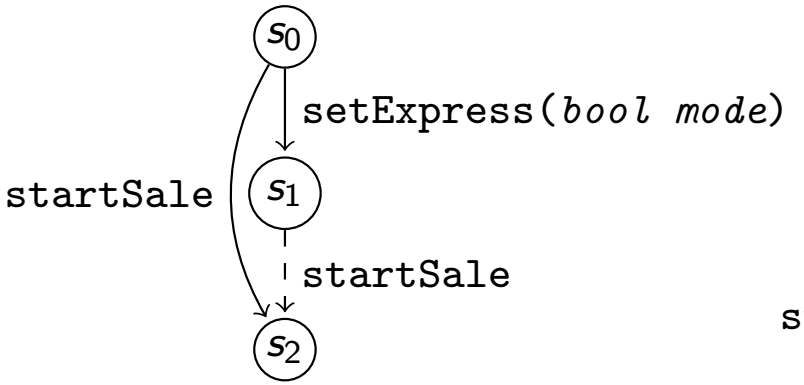
(More interesting if more than one incoming edge...)

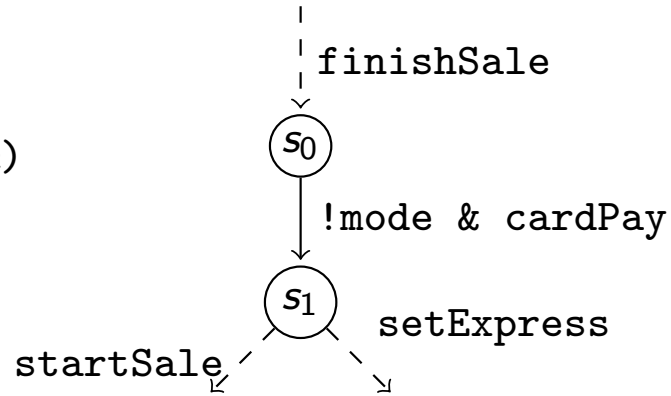
Binding and Testing State

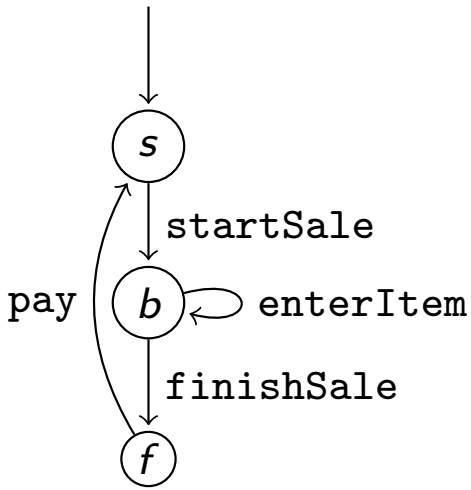
Introduce state variable:

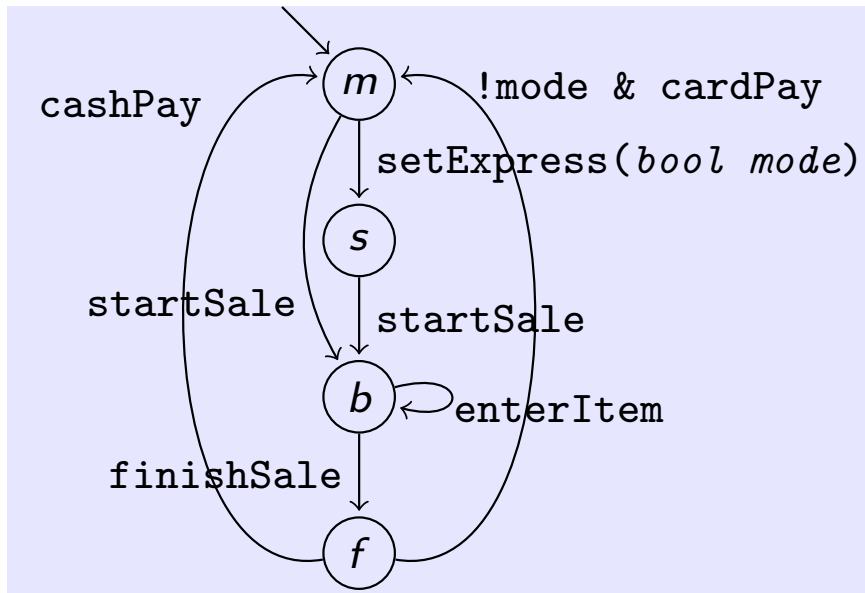
```
delta ExpressMon
  modifies class Cashdesk
    adds Maybe<Bool> monMode = Just(False);
    modifies Unit setExpress(Bool m) {
      monMode = Just(m); // record mode
      // Only allowed between sales
      if (state == Init) { original(m); } else {assert False; }
    }

delta ExpressCCMon {
  modifies class Cashdesk {
    modifies Bool cardPay(CCData cc) {
      assert ~fromJust(monMode); return original(cc);
    }
```

Combined Monitors









More Scenarios & Future Work

- Store protocol in annotations (see Bodden/Stolz 2006)
- Use protocol to implement `QueueManager`
  —resulting monitor no longer depends on single object.
  Problem: ABS doesn't support static factories; no way to connect to global monitor?
- Teach guards about object identities, quantification, allow general function calls
  - Example: `o` $\neq$ `p & o.m(x)` (for LTL: [RV 2007])
  - Danger: $\xrightarrow{\texttt{o.m()}} \cdot \xrightarrow{\texttt{p.n()}}$ ("hidden" monitor)
- In actor setting:
  - role of *callee*?
  - what to do (`skip` instead of `assert False`?)
- Not only enforce protocol—*contribute* behaviour.

Conclusion

- Monitoring is just another feature
- Monitoring for SPLs:
  base monitor + delta monitors (both as deltas)
  instead of defining monitors for products
- Wellformedness checked afterwards
- Similar effects wrt. ordering like deltas for programs,
  monitor delta is applied after corresponding code delta