On Event-B and Control Flow

Alexei Iliasov

Newcastle University, Newcastle Upon Tyne, England

July 17, 2009

Plan

- Event ordering in Event-B
- Event-B Flow viewpoint: extending but not changing
- Flow language
- Example
- Verification
- Future work

The Goals

1. Adding explicit event ordering support to Event-B
2. Not changing Event-B

Event ordering in Event-B (1)

It is best to formulate a problem in the way it suits Event-B

- Many problems do not require explicit ordering
- Certain ordering properties may be captured in refinement
- Auxiliary variables may be introduced to enforce event ordering

Event ordering in Event-B (2)

At later development stages, when implementation concerns are captured, event ordering becomes more constrained

- Events are combined into blocks (informally)
- For some problems, a substantial number of guards/actions handles event ordering
- Some preparatory work towards code generation
- Some specific techniques: constructing loops, merging events

Context (1)

CSP (CCS, pi, ...) $\|$ B (Event-B, Z, ActSys, ...)

- Interpret state part as a communicating process (e.g., CSP process)
- Compose at the process algebraic level
- Process algebraic semantics
- Process algebraic verification

It works, but ...

- Complex semantics
- Not an extension/integration but rather a new formalism
- Alien to the users from both camps
- Poor reasoning support: starting from a scratch

Context (2)

So, when extending a method...

- do not change the existing verification technique: essential part of a formalism
- retain backwards compatibility: projection of an extension back onto the original method
- make it possible to ignore the extension
- when possible, reuse the existing infrastructure
- make the extension blend into the method

Context (3)

Possible directions for intrducing event ordering into for Event-B:

- computing event ordering of a machine: undecidable in general, seems to be difficult even for simpler cases
- checking whether a given ordering describes the possible event orderings is much easier
- it is even easier to check that addition of event ordering does not introduce new deadlocks and divergencies

Event-B Flow viewpoint: extending but changing

The proposal:

- a new *viewpoint* for an Event-B developments
- Event-B machine is treated as another viewpoint
- separation of concerns
- verification based on theorem proving
- does not change the method: a differing presentation style

Event-B Flow viewpoint: extending but changing

There are a number of reasons to consider an extension of Event-B with an event ordering mechanism:

- for some problems the information about event ordering is an essential part of requirements; it comes as a natural expectation to be able to adequately reproduce these in a model;
- explicit control flow may help to prove properties related to event ordering;
- sequential code generation requires some form of control flow information;
- model checking might benefit from explicit event ordering information description;
- there is a potential for a visual editor based on control flow information;
- realizing such a mechanism could help to bridge the gap between high-level workflow languages and Event-B

Event-B Flow

```
system gcd
variables a, b
invariant a ∈ ℕ ∧ b ∈ ℕ
initialisation a := 0||b := 0
flow
    input. ∗ (eucgcd)
events
    input    =  any f, s where
                  a + b = 0 ∧ f + s > 0
                then
                  a := f
                  b := s
                end
    eucgcd   =  when
                  b ≠ 0
                then
                  b := a mod b
                  a := b
                end
```

Event-B + Flow

Event-B Model

Event-B/Flow Consistency Model

Flow Model

Event-B Flow: language

$e_i(p)$	event
$p; q$	sequential composition
$p \| q$	parallel composition
$p \sqcap q$	choice
$*(p)$	terminating loop
$**(p)$	non-terminating loop
$'start$	initialisation event
$'stop$	termination event
$'skip$	stuttering event

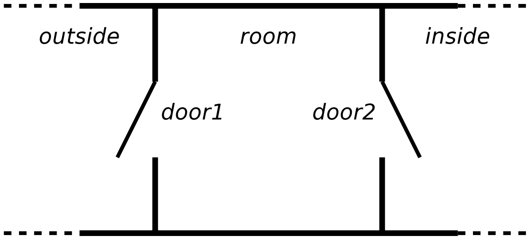
Event-B Flow

Some examples:

- $first.'stop$
- $*(first.second).'stop$
- $(first \| second); (third | fourth)$
- $read.(workA1.workA2 | workB). * *(write)$
- $'start. * (e_1 | e_2 | \ldots | e_k).'stop$

Not all machine events have to be mentioned.

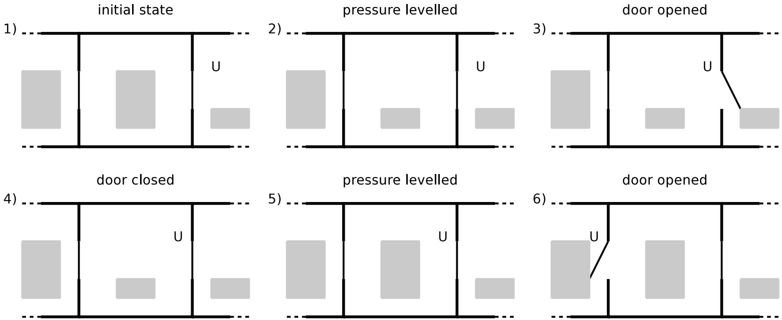
Example



Example: Requirements

1. the system allows a user to get inside or outside by leveling pressure between room and a destination
2. the system has three locations - *outside*, *sluice* and *inside*
3. the system has two doors - *door1*, connecting *outside* and *sluice*, and *door2*, connecting *sluice* and *inside*;
4. there is a device to change pressure in *sluice*;
5. a door may be opened only if the pressures in the locations it connects is equalised;
6. at most one door is open at any moment;
7. the pressure can only be switched on when the doors are closed;
8. *when inside, a user is always able to get outside*;
9. *when outside, a user is always able to get inside*

Example: Use Case



Flow consistency (1)

$$e_1 = \texttt{any}\ p\ \texttt{where}\ G(p,v)\ \texttt{then}\ S(p,v,v')\ \texttt{end}$$
$$e_2 = \texttt{any}\ q\ \texttt{where}\ H(q,v)\ \texttt{then}\ R(q,v,v')\ \texttt{end}$$

Composed event:

$$"e_1; e_2" = \begin{array}{l}\texttt{any}\ p\ \texttt{where} \\ \quad G(p,v) \\ \texttt{then} \\ \quad S(p,v,v'); (\exists q \cdot H(q,v) \wedge R(q,v,v')) \\ \texttt{end}\end{array}$$

Sequential composition of event actions:

$$S_0(p,v,v'); S_1(p,v,v') \mathrel{\widehat{=}} \exists\ v_1\ \cdot\ S_0(p,v,v_1)\ \wedge\ S_1(p,v_1,v')$$

Flow consistency (2)

Proving that $e_1; e_2$ s consistent with the machine:

$$\begin{array}{l} P(c,s) \wedge I(c,s,v) \wedge G(c,s,p,v) \models \\ \exists v' \cdot (S(p,v,v'); (\exists q \cdot H(q,v) \wedge R(q,v,v'))) \models \\ \exists\ v_1\ \cdot\ (S(p,v,v_1)\ \wedge\ \exists q \cdot H(q,v_1) \wedge R(q,v_1,v'))) \end{array}$$

Composed events feasibility is assumed:

$$\begin{array}{l} P(c,s) \wedge I(c,s,v) \wedge G(c,s,p,v) \models \exists v' \cdot S(p,v,v') \\ P(c,s) \wedge I(c,s,v) \wedge H(c,s,q,v) \models \exists v' \cdot R(q,v,v') \end{array}$$

Simplify...

$$\begin{array}{l} P(c,s) \wedge I(c,s,v) \wedge G(c,s,p,v) \wedge S(p,v,v_1) \models \\ \exists\ q, v_1\ \cdot\ H(q,v_1) \wedge R(q,v_1,v') \end{array}$$

A practical proof obligation condition:

$$P(c,s) \wedge I(c,s,v) \wedge G(c,s,p,v) \wedge S(p,v,v_1) \models \exists\ q \cdot\ H(q,v_1)$$

(next event is enabled in the after-states of the previous event)

Abstract/Concrete

Flow may play a number of *roles*:

- The implementability property of a workflow
  - abstract: non-deterministic choice is allowed
  - concrete: only deterministic choice is allowed
    - for each workflow choice show that next event guards are pairwise disjoint
- Determining whether a workflow is overlaid (a driver) or an actual machine workflow
  - overlaid: next events of an event does not have to be all the potentially enabled events
  - equivalence: next events contain all the possible enabled events

Future Work

- So far just a small-scale experiment
- Some ideas on where to take this next

Verification: Workflow Properties

It is possible to compute certain properties of a workflow:

- $e_1 \nrightarrow e_2$ (after e_1 eventually e_2)
  - closure of a relation defined by flow (computable)
- good thing keeps happening: $'all \nrightarrow good$
- system termination: $'start \nrightarrow' stop$
- after an error success may not be reached: $\neg error \nrightarrow success$

Loop Convergence with Workflow

More flexibility in establishing convergence

- one variant per loop
- nested loops ok
- not all loop events must decrement variant
  - in sequential composition, it is enough for only left or right part to decrement variant
  - for a choice, all the branches must be involved
  - example: for $*(a;(b|c;d))$ it is enough to have a and d update variant variables

Verification Process

Now:

- Check Event-B model consistency/refinement
- Check Flow consistency/refinement
- Check Event-B/Flow consistency

Verification Process

Possibly in future:

- Flow is made visible to a machine
- Refer to flow in invariants, event guards and actions (read-only)
- Two new variables added to a model: next function, pointer to a current event
- The variables do not appear anywhere in a model but SC and POG are made aware of them

Verification Process (E)

Alternative future:

- Use flow to generate additional hypothesis
- The state in which event may be enabled is constrained by flow
  - $a; b$ item b is only enabled in after-states of a - may be stronger than guard of b

Event-B0

Event-B + Flow could be seen as a counterpart of B0

- Event-B stylised as an algorithmic language
- Gradual refinement process (needs feasibility study)
- Code generation
- Positioning as an intermediate notation: discourage use in early refinement steps

Event-B0

```
system gcd
variables a, b
invariant a ∈ ℕ ∧ b ∈ ℕ
initialisation a := 0 || b := 0
flow
    input . ∗ (eucgcd)
events
    input    =   any f, s where
                   a + b = 0 ∧ f + s > 0
                 then
                   a := f
                   b := s
                 end
    eucgcd   =   when
                   b ≠ 0
                 then
                   b := a mod b
                   a := b
                 end
```

```
int a = 0;
int b = 0;
void gcd(int f, int s) {
    if (a + b = 0 ∧ f + s > 0) {
        a = f;
        b = s;
    } else {
        return;
    }
    while (b ≠ 0) {
        b = a mod b;
        a = b;
    }
}
```

Code generation

Some notes:

- What is needed: *concrete* flow + deterministic event actions
- Code generation should be a small step: stay for as long as possible within the platform
- Tasking/Scheduling as a separate viewpoint: nothing to do with flows. Possibly several of differing tasking viewpoints
- It is still important to be able to generate code directly form Event-B

(Near) Future Work

- Experiment with tool: feasibility study, esp. scalability issues for proof obligations
- Check POs soundness
- Fix POs to include needed hypothesis
- Machine/flow interaction study: e.g., refining out auxiliary variables using flow
- Tackle some larger examples (SSF?)

The End

Thank You!

Questions?