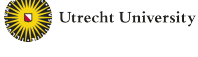
Change Impact Analysis for Rebel Specifications

Jordi Wippert 6303013 May 15, 2020



Title clarification

- Change Impact Analysis:

‘The process of identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change.’

- Rebel, a Formal Specification Language

Questions and motivation

- Why should one use a formal specification language (FSL)?

- Why Rebel?

- Current state (of Rebel)

Questions and motivation

- `Why should one use a formal specification language (FSL)?`

  FSLs are formal methods which have a mathematical basis, e.g. **for proves**

- `Why Rebel?`

  Rebel is a DSL by **ING**, especially for the **financial domain**

- `Current state (of Rebel)`

  Relatively new and not widely incorporated yet due to **open issues** such as **changeability**

Questions and motivation

- `Why should one use a formal specification language (FSL)?`

  FSLs are formal methods which have a mathematical basis, e.g. **for proves**

- `Why Rebel?`

  Rebel is a DSL by **ING**, especially for the **financial domain**

- `Current state (of Rebel)`

  Relatively new and not widely incorporated yet due to **open issues** such as **changeability**

`How can formal methods be used to compute the impact of a change to a Rebel specification?`

Table of contents

Background

preliminaries for Change Impact Analysis

Extended Finite State Machine

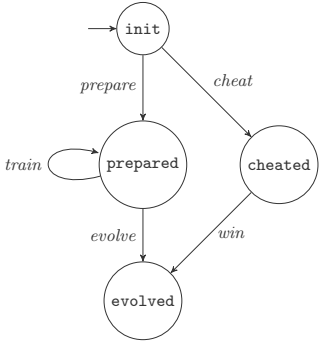


Figure: State diagram of model *Runner*

$M = (Q, \Sigma, I, V, \Lambda)$

- Q is a set of states.
- Σ is a finite set of events: $e\ (p^*)[c^*]/a^*$, where: **e**ventname, **p**arameters, **c**onditions and **a**ctions.
- $I \subseteq Q$ is the set of initial states.
- V is the set of state variables: $x \in Instance \rightarrow Value$
- Λ is a set of transitions: $q \xrightarrow{e} q'$

Figure: EFSM definition

Q = {init, prepared, evolved, cheated}

Σ = {$prepare()[]/hp = hp + 1$,
$train()[hp + 2 \leq 10]/hp = hp + 2$,
$evolve()[hp \geq 5]/pt = pt + 1$,
$cheat()[]/hp = hp + 3$,
$win()[]/pt = pt + 2$}

I = {init}

V = {$hp \in Instance \rightarrow \mathbb{Z}$, $pt \in Instance \rightarrow \mathbb{Z}$}

Λ = { $\xrightarrow{INITIALISATION}$ init,
init $\xrightarrow{prepare}$ prepared,
init $\xrightarrow{cheat}$ cheated,
prepared $\xrightarrow{train}$ prepared,
prepared $\xrightarrow{evolve}$ evolved,
cheated $\xrightarrow{win}$ evolved }

where $INITIALISATION : hp = 0; pt = 0$

Figure: EFSM of model *Runner*

States and State Spaces

- Concrete state:
  $concrete\_state \subseteq V \times Value$
- State space
- Initializations: *default* and *custom*

$$
\begin{array}{l}
/ * \ Variables \ * / \\
\\
hp = \{(Runner1 \mapsto 3), (Runner2 \mapsto 1)\} \\
pt = \{(Runner1 \mapsto 2), (Runner2 \mapsto 0)\} \\
\_state = \{(Runner1 \mapsto \texttt{evolved}), (Runner2 \mapsto \texttt{prepared})\}
\end{array}
$$

Figure: Concrete state in model *Runner*

Impact

- Models: M and M'
- State Spaces: S and S'
- State Space Impact: $S \setminus S'$

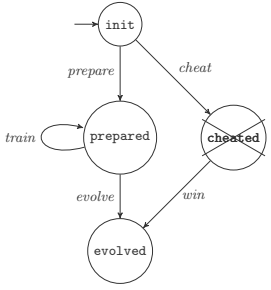


Figure: State diagram of model *Runner*,
with a change applied that prohibits state `cheated`

Changes (as Invariants) and model checkers

"As Stakeholder we want Runners that cheated always to have a hp of 3 as a minimum."

- Changes
- Invariants
- Selected constructs
- Consistency checking
- State space exploration

$$\forall\, r \in Runner : \_state(r) \neq \texttt{cheated}$$

Figure: Change as invariant example (1)

$$\forall\, r \in Runner : \_state(r) = \texttt{evolved} \Rightarrow hp(r) \geq 5 \ \wedge \ pt(r) \geq 1$$

Figure: Change as invariant example (2)

Rebel

- Formal specification language
- For documentation purposes
- To share unambiguously aligned financial knowledge
- Designed for code derivation
- Syntax and semantics

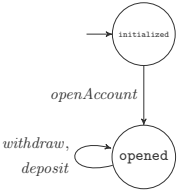


Figure: Account

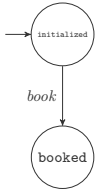


Figure: Transaction

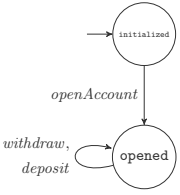


Figure: Account

Figure: Bank state machine

```
module bank.Library
import bank.Account
import bank.Transaction

@doc { Account and Transaction-events }

event openAccount() {}

event withdraw(amount: Money) {
  preconditions {
    amount > EUR 0.00;
  }
  postconditions {
    new this.balance == this.balance - amount;
  }
}

event deposit(amount: Money) {
  preconditions {
    amount > EUR 0.00;
  }
  postconditions {
    new this.balance == this.balance + amount;
  }
}

// No explicit pre- and postconditions
event book() {
  sync {
    Account[this.from].withdraw(this.amount);
    Account[this.to].deposit(this.amount);
  }
}
```

Figure: Rebel Library ↑ and Rebel Specifications →

```
module bank.Account
import bank.Library

specification Account {
  fields {
    accountNumber: IBAN @key
    balance: Money
  }

  events {
    openAccount[]
    withdraw[]
    deposit[]
  }

  lifeCycle {
    initial inititalized -> opened: openAccount
    opened -> opened: withdraw, deposit
  }
}
```

```
module bank.Transaction
import bank.Library
import bank.Account

specification Transaction {
  fields {
    id:     Integer @key
    from:   IBAN    @ref=Account
    to:     IBAN    @ref=Account
    amount: Money
  }

  events {
    book[]
  }

  lifeCycle {
    initial inititalized -> booked: book
  }
}
```

Formal Specification Languages

- Alloy, B-Method, Event-B, TLA+, VDM
- Current state, features and usability
- Best for our needs: B-Method (and ProB)
- Translation/Mapping (to B-Method)
  - Specifications become sets
  - Fields become functions
  - Events in sync-blocks are split

Change Impact Analysis

to identify the potential consequences of a change

Change Impact Analysis

- Collect information as starting point (state space S of model M)
- Apply a change (to M)
- Collect information for comparison (state space S' of model M')
- Obtain impact for our property of interest: state space differences

Change Impact Analysis

- Collect information as starting point (state space S of model M)
- Apply a change (to M)
- Collect information for comparison (state space S' of model M')
- Obtain impact for our property of interest: state space differences

- Changes as invariants do not update models!
- Two methods: *iterative* and *direct*

Iterative method

- Based on invariant violations:
  counterexamples

- Model update:
  weakest precondition calculus

$$\forall\, r \in Runner : \_state(r) = \texttt{evolved} \Rightarrow hp(r) \geq 5 \wedge pt(r) \geq 1$$

Figure: Invariant example (repeated)

$$/* \; Variables \; */$$

$$hp\texttt{: [(}Runner1\texttt{ = 3)]}$$
$$pt\texttt{: [(}Runner1\texttt{ = 2)]}$$
$$\_state\texttt{: [(}Runner1\texttt{ = evolved)]}$$

Figure: Counterexample concrete state

```
extension  = violating_terms(counterexample)
invariant' = invariant ∨ (extension)
```

Figure: Invariant extension definition

$$\forall\, r \in Runner : \_state(r) = \texttt{evolved} \Rightarrow hp(r) \geq 5 \wedge pt(r) \geq 1 \vee (\boldsymbol{hp(r) = 3 \wedge pt(r) = 2})$$

Figure: Extended invariant

Weakest precondition calculus

- Predicate transformer (Dijkstra)
- Computes a condition that guarantees: postcondition R w.r.t. a statement $x := E$

$$wp(x := E, R) = R[E/x]$$

Figure: Weakest precondition assignment rule

```
/ * Variables * /

hp: [(Runner1 = 3)]
pt: [(Runner1 = 2)]
_state: [(Runner1 = evolved)]
```

Figure: Counterexample concrete state (repeated)

```
win =
  p* : r
  c* : r : Runner ∧ _state(r) = cheated
  a* : pt(r) := pt(r) + 2;
     _state(r) := evolved}
```

Figure: Event win

```
   wp (x := E) R  ⟺  R[E/x]
≡  wp (pt := pt + 2) (pt = 2)
≡  (pt = 2)[(pt + 2)/pt]
≡  (pt + 2) = 2
≡  pt = 0
```

Figure: WP for *pt*

Direct method

- Uses the invariant as desired post-condition R in the wp calculus
- All events must be updated, as no counterexample/trace information is available
- Implication: *if-then* construction
- No negation needed

State Space Impact

- After obtaining M' based by updating M
- S' can be computed
- The state space is not dependent on the update method: i.e. the *iterative* and *direct* update method result in the **same** state space

Impact:

- $S \setminus S'$, for constrained state spaces
- $S \triangle S'$, when additions must be shown
- $(|S'| - |S|) \; / |S| * 100$, for relative decrease/increase

S :

`init:`	$[(hp = 0),\ (pt = 0)]$
`prepared:`	$[(hp = 1, 3, 4, 5, 7, 9),\ (pt = 0)]$
`evolved:`	$[(hp = \mathbf{3}, 4, 5, 7, 9),\ (pt = 1, \mathbf{2})]$
`cheated:`	$[(hp = 3),\ (pt = 0)]$

Figure: Simplified state space (values per variable and abstract state modulo instances)

Experiments

to answer our research question

Implementation

- Algorithms
- Models
- Invariants
- Options:
  - Model checker mode
  - Symmetry mode
  - Cardinality of sets (for instances)
  - Minimum/maximum integer

Algorithm 1: Change impact analysis

```
1 Function changeImpactAnalysis(M, I, opts):
2     S  ← probcli (M, opts)
3     M' ← update (M, I, opts)   /* direct or iterative* /
4     S' ← probcli (M', opts)
5     return S \ S'
```

Evaluation of options

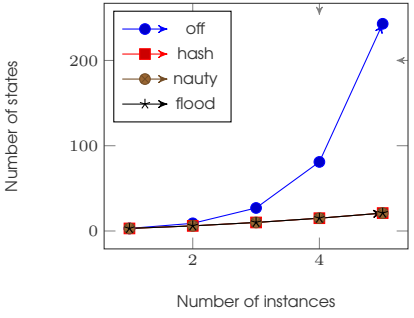


Figure: Number of states in relation to the number of instances using *symmetry* modes and model *ABC*

symmetry	instances	states	transitions	time (ms)
off	1	11	11	43
off	2	121	221	112
off	3	1331	3631	825
off	4	14641	53241	11999
off	5	161051	732051	193107
hash	1	11	11	43
hash	2	66	121	90
hash	3	286	781	246
hash	4	1001	3641	876
hash	5	3003	13651	3014
flood	1	11	11	47
flood	2	66	176	100
flood	3	286	1826	296
flood	4	1001	17281	1405
flood	5	3003	171699	12145
nauty	1	11	11	46
nauty	2	66	121	100
nauty	3	286	781	316
nauty	4	1001	3641	1298
nauty	5	3003	13651	5852

Table: Results of using *symmetry* modes and model *Runner*

Evaluation of update methods

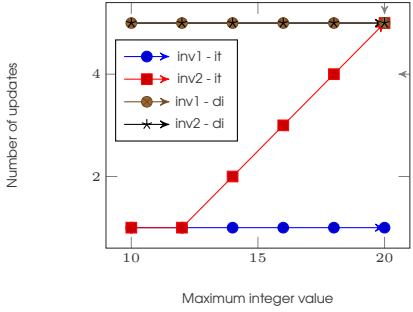


Figure: Number of updates in relation to the maximum integer value using model *Runner*

model	invariant	method	instances	updates	time (s)
ABC	1	*iterative*	1	1	6.732
ABC	2	*iterative*	1	1	7.204
ABC	1	*direct*	1	2	0.160
ABC	2	*direct*	1	2	0.169
Runner	1	*iterative*	1	1	6.934
Runner	2	*iterative*	1	0	2.325
Runner	1	*direct*	1	5	0.355
Runner	2	*direct*	1	5	0.416
Bank	1	*iterative*	2*	10	550.832
Bank	2	*iterative*	2*	6	509.880
Bank	1	*direct*	2*	6	1.287
Bank	2	*direct*	2*	6	1.302

All data is obtained with MININT -5 and MAXINT 10
*2 Accounts, 1 Transaction

Table: Results for model updates

Evaluation of Impact

model	invariant	maxint	$\|S\|$	$\|S'\|$	$S\ impact\ (\%)$	$\|T\|$	$\|T'\|$	$T\ impact\ (\%)$
$Runner$	1	10	11	10	$-9{,}09$	11	10	$-9{,}09$
$Runner$	1	12	13	12	$-7{,}69$	13	12	$-7{,}69$
$Runner$	1	14	15	14	$-6{,}67$	15	14	$-6{,}67$
$Runner$	1	16	17	16	$-5{,}88$	17	16	$-5{,}88$
$Runner$	1	18	19	18	$-5{,}26$	19	18	$-5{,}26$
$Runner$	1	20	21	20	$-4{,}76$	21	20	$-4{,}76$
$Runner$	2	10	11	11	$0{,}00$	11	11	$0{,}00$
$Runner$	2	12	13	11	$-15{,}38$	13	11	$-15{,}38$
$Runner$	2	14	15	11	$-26{,}67$	15	11	$-26{,}67$
$Runner$	2	16	17	11	$-35{,}29$	17	11	$-35{,}29$
$Runner$	2	18	19	11	$-42{,}11$	19	11	$-42{,}11$
$Runner$	2	20	21	11	$-47{,}62$	21	11	$-47{,}62$
$Bank$	1	10	314973	113849	$-63{,}85$	1952018	636439	$-67{,}40$
$Bank$	2	10	314973	139336	$-55{,}76$	1952018	1293715	$-33{,}72$

All data is obtained with MININT -5 and MAXINT 10
$S(')$ is State Space (Prime) and $T(')$ is Transitions (Prime) for $M(')$. Impact is computed by: $(|S'| - |S|)/|S| * 100$. Similar for T.

Table: Results for impact using several models and invariants

Future Work

Future Work

- Changes
- Static approach
- Optimizations
- Solving impacted instances
- Integration

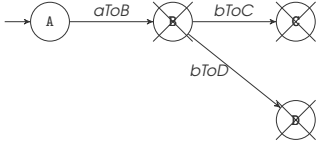


Figure: Example model for reversed transitions

Figure: Example model updated with reverse transition

Conclusion

Conclusion

How can formal methods be used to compute the impact of a change to a Rebel specification?

Conclusion

`How can formal methods be used to compute the impact of a change to a Rebel specification?`

- Definitions for *impact* and *changes*.
- Two methods to update a model with respect to a change encoded as invariant.
- A clear description of the *syntax* and *semantics* for Rebel.
- A comparison of formal specification languages for the translation of Rebel.
- A *change impact analysis* that reveals *state space differences* algorithmically.
- An conceptual approach to solve impacted instances.

Questions?