Temporal Logic Specification and Analysis for Model Transformations

S. Yassipour-Tehrani, K. Lano
Dept of Informatics, King's College London, UK

July 12, 2015

Introduction

- We use Linear Temporal Logic (LTL) to express transformation semantics, and SMV to encode this semantics and to perform model checking.

- LTL operators: $\bigcirc$ (next state), $\diamond$ (some future state), $\Box$ (all future states), $\blacklozenge$ (past).

- Can express fine-grain temporal orderings.

LTL specification of transformation properties

Four generic property patterns:

1. *Achieve*: $G \Rightarrow \diamond Q$ – Q holds in some future state.

2. *Cease*: $G \Rightarrow \diamond \neg\ Q$ – There will be some point in future where Q does not hold

3. *Maintain*: $G \Rightarrow \Box Q$ – Q holds in all future states

4. *Avoid*: $G \Rightarrow \Box \neg\ Q$ – Q will never hold in future.

Transformation semantics in temporal logic

($WithinEnabling$):

$$\Box(\forall\, s : S_i \cdot r_i(s) \;\Rightarrow\; en(r_i(s)))$$

($InvokedRules$):

$$\Box(r(s') \;\Rightarrow\; \vee_i \exists\, s : S_i \cdot r_i(s))$$

($RuleExclusion$):

$$\Box(\forall\, s, s' : S_i \cdot r_i(s) \wedge s' \neq s \;\Rightarrow\; \neg\, r_i(s'))$$

for each top-level r_i, and $\Box(\forall\, s : S_i \cdot r_i(s) \;\Rightarrow\; \neg\, (\exists\, s' : S_j \cdot r_j(s')))$ for each top-level r_j, $i \neq j$.

$(NoReapplication)$:

$$\Box(\forall\, s : S_i \cdot r_i(s) \;\Rightarrow\; \bigcirc \Box\neg\, r_i(s))$$

$(Priority)$:

$$\Box(en(r'(s)) \wedge en(r(s)) \;\Rightarrow\; \neg\, r(s))$$

Encoding transformation semantics in SMV

SMV specification of a class diagram is as follows:

```
MODULE main
VAR
  C : Controller;
  MEntity1 : Entity(C,1);
  .... other object instances ....

MODULE Controller
VAR
  Entityid : 1..n;
  event : { createEntity, killEntity, event1, ..., eventn };

MODULE Entity(C, id)
VAR
  alive : boolean;
  ... attributes of Entity ...
DEFINE
  TcreateEntity := C.event = createEntity & C.Entityid = id;
  TkillEntity := C.event = killEntity & C.Entityid = id;
```

```
  Tevent1 := C.event = event1 & C.Entityid = id & alive = TRUE;
  ... transitions for each event ...
ASSIGN
  init(alive) := FALSE;

  next(alive) :=
    case
      TcreateEntity : TRUE;
      TkillEntity : FALSE;
      TRUE : alive;
    esac;
```

LTL operators denoted in SMV by **G** (for $\Box$), **F** (for $\Diamond$), **X** (for $\bigcirc$) and **O** (for $\blacklozenge$).

Example specification and analysis

We analyse QVT-R case study transformation, UML to RDB:

```
transformation uml2rdb(uml1 : SimpleUML, rdb1 : SimpleRDMS) {
  top relation Package2Schema
  { checkonly domain uml1 p : Package { name = pn }
    enforce domain rdb1 s : Schema { name = pn }
  }

  top relation Class2Table
  { checkonly domain uml1 c : Class
       { namespace = p : Package {}, name = n };
    enforce domain rdb1 t : Table
       { schema = s : Schema {}, name = n };
    when { Package2Schema(p,s); }
    where { Attribute2Column(c,t); }
  }

  relation Attribute2Column
  { checkonly domain uml1 c : Class { attribute =
```

```
        a : Attribute
          { name = an, type = typ : Type { name = tn } } };
    enforce domain rdb1 t : Table { column =
        col : Column {name = an, coltype = tn} };
  }
}
```

QVT-R semantics

$(WhenCond)$:

$$en(Class2Table(c)) \;\Rightarrow\; \blacklozenge Package2Schema(c.namespace)$$

Required invariant of transformation is: $\Box(\forall\, s : Schema \cdot$
$\exists\, p : Package \cdot s.name = p.name)$.

SMV specification for uml2rdb, with source model with one package and two classes, is:

```
MODULE main
VAR
  C : Controller;
  MPackage1 : Package(C,1);
  MClass1 : Class(C,1);
  MClass2 : Class(C,2);
  MSchema1 : Schema(C,MPackage1,1);
  MTable1 : Table(C,MClass1,1);
  MTable2 : Table(C,MClass2,2);

MODULE Controller
VAR
  Packageid : 1..1;
  Classid : 1..2;
  Schemaid : 1..1;
  Tableid : 1..2;
  event : { Package2Schema, Class2Table };

MODULE Package(C, id)
```

```
FROZENVAR
  name : { string1, string2, string3 };

MODULE Schema(C, P, id)
VAR
  alive : boolean;
  name : { empty, string1, string2, string3 };
DEFINE
  TPackage2Schema := C.event = Package2Schema &
          C.Packageid = id;
ASSIGN
  init(alive) := FALSE;

  next(alive) :=
    case
      TPackage2Schema : TRUE;
      TRUE : alive;
    esac;

  init(name) := empty;

  next(name) :=
```

```
case
  TPackage2Schema : P.name;
  TRUE : name;
esac;
```

This represents effect of $Package2Schema(p)$: creation of a Schema instance and setting of its name.

Invariant:

```
LTLSPEC G(MSchema1.alive = TRUE  ->
                          MSchema1.name = MPackage1.name)
```

Counter-examples are presented as sequences of states – directly correspond to traces of transformation execution.

Validated properties are reported as true, eg.:

```
NuSMV > read_model -i volt.smv
NuSMV > go
-- specification  G (MSchema1.alive = TRUE ->
             MSchema1.name = MPackage1.name)  is true
NuSMV >?
```

Results of invariant verification

Source model size	*Formula size*	*Time taken*
3	2 conjuncts	less than 1s
6	5 conjuncts	1s
11	10 conjuncts	5 minutes
21	20 conjuncts	Out of memory

Conclusions

- An approach for specification and analysis of model transformations using linear temporal logic and nuSMV model checker.

- Implementing an example.

  Our contribution is to define general temporal logic semantics for MT languages.