Automatic Determination Statecharts Based on Guard of Events

A. Chouarfia, H. Bouziane and B. Messabih
Institut d’Informatique de l’Université USTO, BP 1505 Oran M’Naouer, Algérie

Abstract: In this study, we are attempting to show how to calculate automatically, through a GeneEtat tool, a set of significative states, hence a description of states, starting from a given abstract system. The basic method of generation, widely used, is based on the guard of events: an event is released in a given state, whenever its guard is valid. Owing to the complexity of the abstract system, the method of generation based on guard may not be sufficient, other methods are proposed.

Key words: B-method, abstract system, transition system, transition state, interpretation, guard, events, state chart

INTRODUCTION

Since software systems are present in all area of Computer Science: control process, chipped systems, etc. it is necessary to have methods allowing their specification, design, coding and verification. Unlike the classical operations of a module, specified by the relation between input and output, systems must be described by their behaviour.

The most usual techniques are based on transition diagrams that give the next states of the system. It is also important to be able check whether these systems are of deadlock or livelock type. The B formalism and the tools inherent to the B method are used. The aim was to develop a software system by writing its specifications and properties at the high level of abstraction and refining it keeping, keeping along the same properties up to some level of implementation, which is translated into an object oriented language.

Testing the software coherence and the preservation can be achieved by means of Proofs Requirements (PR) generated by a tool of B Ateiler. PR are proved by the interactive and automatic demonstration tools.

GeneSyst, a useful, allows the generation of transition systems from an abstract system. The fact that the states must be supplied by the user himself, makes GeneSyst difficult to handle. The GeneEtat tool would allow to palliate this insufficiency by calculating automatically a set of significative states starting from a given abstract system. The automatic generation is based on guards of events, which play the key role in the problem of event release.

ABSTRACT MACHINE

The basic mechanism of B-Method is the Abstract Machine (AM), closer to the class notion in UML. It consists of three parts:

Identification part: AM specifies the name of the Abstract Machine, eventually followed by parameters being either sets or scalars. The latter are specified by a Predicate under the section Constraints. The AM specifies the name of the AM, eventually followed by parameters being either sets or scalars.

The latter are specified by a Predicate under the section Constraints.

Declarative part: It describes data that could be:

Sets
Constants
Properties
Variables
Invariant
Assertion

Dynamical part: It describes the machine behaviour, it contains the clause of initialisation (Generalized substitution) and a list of operation under the form:

List_Id→Op_Name(List_Id)=
PRE P THEN S END

P: Predicate
S: Generalized Substitution

Corresponding Author: Dr. A. Chouarfia, Institut d’Informatique de l’Université USTO, BP 1505 Oran M’Naouer, Algérie
E-mail: Chouarfia@mail.univ-usto.dz
Predicates used are predicates of first order, Generalized Substitutions, which are mathematical notations allowing the modelisation of Predicate transformations, are used for the description of operation behaviour and as well for the systematic setting up of PR, starting from components B (AM, refinement, implementation).

One distinguishes the following Generalised Substitutions:

- Substitutions becomes equal, becomes like, becomes element of
- Precondition condition for operation call
- SELECT
- ANY, LET introducing data verifying some properties
- VAR for local variable
- If and Case
- Operation call
- Loop
- ASSERT introducing properties easing the proof.

**Operation_Declaration**

```
Operation_Signature @ Substitutions
```

**Operation_Signature**

```
List_Id ~ Identifier(List_Id)
Identifier (List_Id)
List_Id ~ Identifier
Identifier
```

**List_Id**

```
List_Id: Identifier
Identifier
```

Abstract Machine description

**Coherence:** The most important feature that distinguishes B-Method from others is the coherence proof.

Let AM be an Abstract Machine

```
MACHINE AM
INCOMPLETE I
INITIALISATION U
OPERATIONS V₁, V₂, ..., Vₙ
END
```

AM is said to be coherent if an initialization U set up an invariant I, which is preserved by operations V₁,...,Vₙ.

Initialization can be seen as a particular operation called once before all the others.

It has to preserve the invariant I of the abstract machine AM. This is expressed by the following PR:

```
I → [U][I]
```

Also the operation must be coherent. An operation is representend under the following form.

```
V = PREP THEN S END
```

It is said to be coherent if it preserves the invariant I by the expression:

```
PA I → [S][I]
```

**ABSTRACT SYSTEM**

An abstract system has states like the Abstract Machine, the difference between them is that operations are replaced by events.

Unlike the former, the latter are uncalleable but can be released if their guard are valide.

The space of the system states is the set of the values taken by the variables that satisfy the invariant.
The generalised substitution $U$ of initialization determines an under set of values that constitute the initial states of abstract system. The dynamical part is a set of events of the form event = $E$, $E$ being the body of the event.

The latter has one of the following forms

BEGIN S END
SELECT P THEN S END
ANY z WHERE P THEN S END

P: Predicate,
S: Substitution,
z: Variable

A guard (noted $grd$) is the condition for feasibility of a generalized substitution. Guards are calculated as follows.

$$
grd(\text{BEGIN S END}) = \text{true}$$
$$grd(\text{SELECT P THEN S END}) = P \land grd(S) = P$$
$$grd(\text{ANY z THEN S END}) = \exists z (P \land \text{grd}(S)) = \exists z (P)$$

**Labelled Transition System (LTS):** A LTS is an oriented graph whose edges called transitions, are labelled with letters taken in an Alphabet of events.

The nodes of this graph are called states. Formally a LTS is a quadruplet $M = (Q, A, T, Q_{init})$ with:

- **Q:** Set of states,
- **Q_{init}** Initial state,
- **A:** Alphabet of events,
- **T ⊆ Q × A × Q:** Transition relation.

A LTS whose state and event set are finite is usually called a finite state automaton, which could be seen as the simplest model of a machine.

LTS are generally used in problems related to the modelization of system behaviour.

In the special case of abstract systems in B, we can generate a LTS that modelize its behaviour. In this framework, the states are the abstract system states, the transitions are the events of the system and the initial states are the states releasable after initialization.

**Principles of LTS Construction:** For illustration of the different approaches, one considers the $B$ specification of the following systems: SCSI-2 system, Drink distributor system.

Let a $B$ specification of the SCSI-2 system

**MACHINE SCSI-2**
**SETS**
$\text{DSK} = \{\text{d1, d2}\}$
**CONSTANTS**
$\text{maxi}$
**PROPERTIES**
$\text{maxi} : \text{NAT} \text{ and maxi} \geq 0 \text{ and maxi} < 3$
**VARIABLES**
$\text{buf}$
**INVARIA\text{NT}**
$\text{buf} : \text{DSK} \rightarrow 0, \text{maxi}$
**INITIALISATION**
$\text{buf} := \text{DSK} \ast \{0\}$
**EVENTS**
$\text{ctr}_{-}\text{cmd} = \text{ANY jj WHERE}$
$\text{jj} : \text{DSK and buf(jj)} < \text{maxi}$
$\text{THEN}$
$\text{buf(jj)} := \text{buf(jj)} + 1$
$\text{END}$
$\text{dsk}_{-}\text{rec} = \text{ANY jj WHERE}$
$\text{jj} : \text{DSK and buf(jj)} > 0$
$\text{THEN}$
$\text{buf(jj)} := \text{buf(jj)} - 1$
$\text{END}$

$\text{ctr}_{-}\text{cmd}$: the controller sends a command to disk $jj$
$\text{disk}_{-}\text{rec}$: the disk $jj$ uses a request from his buffer

The simplest method for building a transition system from an abstract system is to enumerate the states and calculate the transitions between them.

Let $S$ be an abstract system with a list of variables $X = (x_1, ..., x_k)$, an invariant $I$, an initialization $U$, a list of events $(\xi_i = \varepsilon_i)$ with $i \in 1..m$ then:

the set of initial states is defined by:

$$\sigma_1 = \{ \wedge_i (x = v_i) \} \text{ for all value } v_i \text{ such as } (U) (x = v_i)$$

the set of system states is calculable by means of the successor function. For each state $\sigma_p$ that satisfy $I$ and for each event $\xi$ that can be released in the state $\sigma_p$.

The successors $\sigma_{pi}$ in the transition labelled by event are the states whose values can be obtained after generalized substitution $E_i$:

$$\sigma_{pi} = \{ \wedge_i (x = v_i) \} \Rightarrow (E_i \wedge (x = v_i')) \forall \sigma \in \sigma_p \quad (1)$$

199
For the SCSI-2 system (Fig. 1), the initial state is specified by the predicate
\[ \text{buf}(d1) = 0 \land \text{buf}(d2) = 0 \]

Applying the definitions (1), yields the following states:
\[
\begin{align*}
\Psi_0 &= \text{buf}(d1) = 0 \land \text{buf}(d2) = 0 \\
\Psi_1 &= \text{buf}(d1) = 1 \land \text{buf}(d2) = 0 \\
\Psi_2 &= \text{buf}(d1) = 2 \land \text{buf}(d2) = 0 \\
\Psi_3 &= \text{buf}(d1) = 0 \land \text{buf}(d2) = 1 \\
\Psi_4 &= \text{buf}(d1) = 1 \land \text{buf}(d2) = 1 \\
\Psi_5 &= \text{buf}(d1) = 2 \land \text{buf}(d2) = 1 \\
\Psi_6 &= \text{buf}(d1) = 0 \land \text{buf}(d2) = 2 \\
\Psi_7 &= \text{buf}(d1) = 1 \land \text{buf}(d2) = 2 \\
\Psi_8 &= \text{buf}(d1) = 2 \land \text{buf}(d2) = 2
\end{align*}
\]

**Abstract interpretation with guards:** The aim of the abstract interpretation is to have information about the global behaviour of a system, in order to verify some properties that are valid on all paths and in all attainable states.

In this case, abstraction represents more paths and more states than the concrete system and if one property is valid in the abstract system it is also valid in the concrete system.

Let \( S \) be a system with invariant \( I \) and a set of events \( E \), the guard of event \( E \) represent the space of release.

The set of abstract states \( \Psi \) is the set of states specified by the no empty predicates \( \text{grd}(E) \land I \)
\[
\Psi = \{ \text{grd}(E) \land I \mid \text{grd}(E) \land I \text{ est satisfiable} \}
\]

In the abstract behaviour, each state contains a set of states whose event \( E \) can be released in the concrete behavior.

**Fig. 2:** Transition system corresponding to the SCSI-2 system

**Fig. 3:** Transition system of drink distributor based on Abstract Interpretation with guards

For the SCSI-2 system, we distinguish two states \( \rho_0 \) and \( \rho_1 \), corresponding to the guard of event \( \text{ctr \_cmd} \) and the guard of event \( \text{disk \_rec} \), respectively (Fig. 2).
\[
\begin{align*}
\rho_0 &= \exists j \ (jj : \text{DSK} \land \text{buf}(jj) < \text{maxi}) \\
\rho_1 &= \exists j \ (jj : \text{DSK} \land \text{buf}(jj) > 0)
\end{align*}
\]

If there exists a disk whose buffer is not full, the state \( \rho_1 \) contains all the states of the system. The state \( \rho_0 \) plays the same role if the disk has buffer value not empty.

**Abstract interpretation with abstraction on guards:** Owing to the complexity of systems, some of the interpretations based on the methods cited previously, are not sufficient.

To go further, it is necessary to have the most abstract behaviour in order to palliate some details specified by variables.
Going from an abstract system to a more abstract system is called an abstraction.

It is achievable by means of different techniques such as grouping of states or suppression of variables. In this study is considered, solely, the abstraction on variables, it can be performed in many ways.

The simplest is to delete purely and simply variables on which the abstraction must be realized.

This approach is illustrated by the drink distributor example (Fig. 3).

Let a B specification of the drink distributor system

\[
\text{MACHINE DISTRI3}
\]

\[
\text{SETS DRINKS = \{nothing, coffee, tea, choc\};}
\]

\[
\text{PIECES = \{HalfEuro, OneEuro, TwoEuro, FiveEuro\};}
\]

\[
\text{ETM = \{off, ready, select, distri, moreperceive, repay\}}
\]

\[
\text{CONSTANTS price, value}
\]

\[
\text{PROPERTIES price: DRINKS --> NAT &}
\]

\[
\text{value: PIECES --> NAT &}
\]

\[
\text{price(nothing) = 0 &}
\]

\[
\text{price(coffee)=300 &}
\]

\[
\text{price(tea)=200 &}
\]

\[
\text{price(choc)=250 &}
\]

\[
\text{value(HalfEuro)=50 &}
\]

\[
\text{value(OneEuro)=100 &}
\]

\[
\text{value(TwoEuro)=200 &}
\]

\[
\text{value(FiveEuro)=500}
\]

\[
\text{VARIABLES drink, sum, display, etm}
\]

\[
\text{INVARIANT drink: DRINKS&}
\]

\[
\text{sum : 0..max(ran(price))+}
\]

\[
\text{max(ran(value)) &}
\]

\[
\text{display = price(drink)-sum&}
\]

\[
\text{etm : ETM}
\]

\[
\text{INITIALISATION drink, sum, display, etm := nothing, 0, 0, off}
\]

\[
\text{OPERATIONS Start_machine=}
\]

\[
\text{SELECT etm = off}
\]

\[
\text{THEN etm := ready}
\]

\[
\text{END}
\]

\[
\text{;}
\]

\[
\text{stop_machine =}
\]

\[
\text{SELECT etm := off}
\]

\[
\text{THEN etm := ready}
\]

\[
\text{END}
\]

\[
\text{;}
\]

\[
\text{select_drink =}
\]

\[
\text{SELECT etm := ready}
\]

\[
\text{THEN ANY bb WHERE bb : DRINKS &}
\]

\[
\text{bb /= nothing &}
\]

\[
\text{THEN drink := bb || sum := 0 ||}
\]

\[
\text{Display := price(bb)|| etm := select}
\]

\[
\text{END}
\]

\[
\text{END}
\]

\[
\text{;}
\]

\[
\text{insert_money =}
\]

\[
\text{SELECT etm := select &}
\]

\[
\text{sum < price(drink) &}
\]

\[
\text{THEN ANY cc WHERE cc : PIECES &}
\]

\[
\text{THEN sum := sum + value(cc)||}
\]

\[
\text{display := display-value(cc) ||}
\]

\[
\text{etm := select}
\]

\[
\text{END}
\]

\[
\text{;}
\]

\[
\text{distribute =}
\]

\[
\text{SELECT etm := select &}
\]

\[
\text{sum >= price(drink) &}
\]

\[
\text{THEN if sum =price(drink) THEN}
\]

\[
\text{etm := distri}
\]

\[
\text{ELSE etm := moreperceive}
\]

\[
\text{END}
\]

\[
\text{;}
\]

201

yy <-- return_currency_1 =
SELECT
    etm := moreperceive
THEN
    yy := sum - price(drink) ||
    sum := price(drink) ||
    display := 0 ||
    etm := distri
END
;

take_bekar=
SELECT
    etm := moreperceive
THEN
    drink := nothing ||
    sum := 0 ||
    display := 0 ||
    etm := ready
END
;
cancel =
SELECT
    etm := select
THEN
    IF sum = 0 THEN
        display := 0 ||
        drink := nothing ||
        etm := repay
    ELSE
        etm := repay
    END
END
;
zz <-- return_currency_2
SELECT
    etm := repay
THEN
    zz := sum ||
    display := 0 ||
    drink := nothing ||
    etm := distri
END

The guards of events of the drink distributor system are:

g0 = etm = stop

g1 = etm = ready

g2 = (etm = ready) \land \exists(bb) (bb e DRINKS) \land
    -(bb = nothing)

\[ g3 = (etm = select \land \sum < price (drink)) \land
    \exists (cc) (cc e PIECES) \]

\[ g4 = (etm = select \land \sum >= price (drink)) \]

\[ g5 = (etm = moreperceive) \]

\[ g6 = (etm = distri) \]

\[ g7 = (etm = select) \]

\[ g8 = (etm = repay) \]

If we assume an abstraction on all variables except etm, we obtain the predicates:

\[ \sigma 0 = (etm = stop), \sigma 1 = (etm = ready), \]

\[ \sigma 2 = (etm = select), \sigma 3 = (etm = moreperceive), \]

\[ \sigma 4 = (etm = distri), \sigma 5 = (etm = repay) \]

The states of drink distributor system are the states specified by the predicates \( \sigma 0 \land \sigma 1, \sigma 2 \land \sigma 1, \sigma 3 \land \sigma 1, \sigma 4 \land \sigma 1 \) et \( \sigma 5 \land \sigma 1 \). They are significative if they are not empty. The states \( \phi 0, \phi 1, \phi 2, \phi 3, \phi 4 \) et \( \phi 5 \) are significative with:

\[ \phi 0 = (etm = stop), \phi 1 = (etm = ready), \]

\[ \phi 2 = (etm = select), \phi 3 = (etm = moreperceive), \]

\[ \phi 4 = (etm = distri), \phi 5 = (etm = repay) \]

CONCLUSIONS

The tool GeneEtat, in its current version, runs under Sun Solaris System with a rudimentary interface, which will be transformed into a graphical one. GeneSyst and GeneEtat are independent. The aim fixed is to couple them and to experiment them in other cases. The work presented here will be continued by extending it to other formalisms of representation of transition system and behaviour from abstract systems specified in B formalism.

REFERENCES


