http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (2): 197-203, 2005
ISSN 1812-5638

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 valide. 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 bahaviour.

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 wether 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 developp 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 Atelier. PR are prooved 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 scalairs.

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

Predicatesused 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.

Abstract Machine

MACHINE
Signature_Machine
CONSTRAINTS
Prédicat
SETS
Sets
CONSTANTS
List_Id
PROPERTIES
Prédicat
VARIABLES
List_Id
INVARIANT
Predicat
INITIALISATION
Substitution
OPERATIONS
Operations
END

Set_Signature
Identifier
Identifier(List_Id)

Sets
Set;Set_Declaration
Set_Declaration

Set_Declaration
Identifier
Identifier={List_Id}

Operations
Operations :Operation_Declaration
Declaration_Operation

Operation_Declaration
Operation_Signature ≙ Substitutions

Operation_ Signature
List_Id ← Identifier(List_Id)
Identifer (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
INVARIANT	I
INITIALISATION	U
OPERATIONS	$V_1, V_2, \ldots V_i \ldots, V_n$
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:

$$I \Rightarrow [U]I$$

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

$$V = PRE\ P\ THEN\ S\ END$$

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

$$P \wedge I \Rightarrow [S]\, I$$

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 uncallable 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 subtitution 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(BEGIN S END) = true
grd(SELECT P THEN S END)= P∧ grd(S)=P
grd(ANY z THEN S END) =
$\exists z.(P \wedge grd(S)) = \exists z.(P)$

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 quaduplet M = (Q, A, T, Q_{init}) with:

Q: Set of states,
Q_{init}: Initial state,
A: Alphabet of events,
$T \subset Q \times A \times Q$: Transition relation.

A LTS whose state and event set are finite is usually called a finite state automat, 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
        DSK={d1,d2}
CONSTANTS
        maxi
PROPERTIES
        maxi : NAT and maxi>=0 and maxi<3
VARIABLES
        buf
INVARIANT
        buf : DSK -->0..maxi
INITIALISATION
        buf := DSK * {0}
EVENTS
        ctr_cmd = ANY jj WHERE
                jj : DSK and buf(jj)<maxi
        THEN
                buf(jj) := buf(jj)+1
        END
;
dsk_rec = ANY jj WHERE
        jj : DSK and buf(jj)>0
THEN
        buf(jj) := buf(jj)-1
END
END
```

ctr_cmd: the controller sends a command to disk jj
disk_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=(x1,...., xk), an invariant I, an initialization U, a list of events (ξ_i=E_i) with i∈1..m then :

the set of initial states is defined by:

$\sigma_0 = \{\wedge_j (x=v_j)\}$ for all value v_j such as $\langle U \rangle (x=v_j)$

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

The successors σ_p in the transition labelled by event are the states whose values can be obtained after generalized substitution E_i:

$$\sigma_{p+1} = \{\wedge_j (x=v_j) | s \Rightarrow \langle E_i \rangle \wedge_j (x=v_j')\} \; \forall \; S \in \sigma_p \quad (1)$$

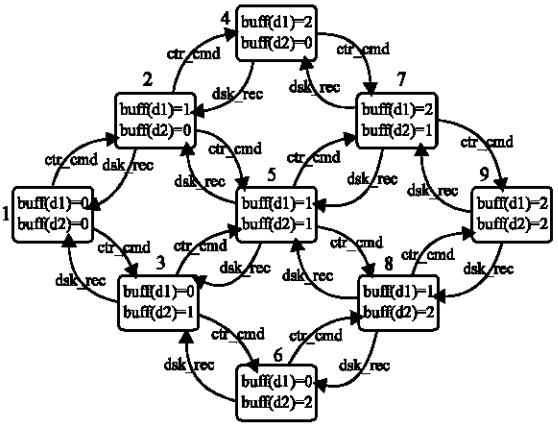

Fig. 1: Transition system of the SCSI-2 system (enumeration states)

For the SCSI-2 system (Fig. 1), the initial state is specified by the Predicate

$$buff(d1) = 0 \wedge buff(d2) = 0$$

Applying the definitions (1), yields the following states

$$\Psi_0 = buff(d1) = 0 \wedge buff(d2) = 0$$
$$\Psi_1 = buff(d1) = 1 \wedge buff(d2) = 0$$
$$\Psi_2 = buff(d1) = 2 \wedge buff(d2) = 0$$
$$\Psi_3 = buff(d1) = 0 \wedge buff(d2) = 1$$
$$\Psi_4 = buff(d1) = 1 \wedge buff(d2) = 1$$
$$\Psi_5 = buff(d1) = 2 \wedge buff(d2) = 1$$
$$\Psi_6 = buff(d1) = 0 \wedge buff(d2) = 2$$
$$\Psi_7 = buff(d1) = 1 \wedge buff(d2) = 2$$
$$\Psi_8 = buff(d1) = 2 \wedge buff(d2) = 2$$

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 valide 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 valide in the abstract system it is also valide in the concrete system.

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

The set of abstract states Ψ is the set of states specified by the no empty predicates $grd(E_i) \wedge I$

$$\Psi = \{grd(E_i) \wedge I \mid grd(E_i) \wedge I \text{ est satisfiable}\}$$

In the abstract bahaviour, each state contains a set of states whose event E_i can be released in the concrete behavior.

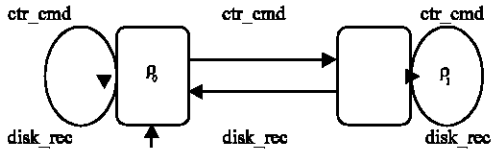

Fig. 2: Transition system corresponding to the SCSI-2 system

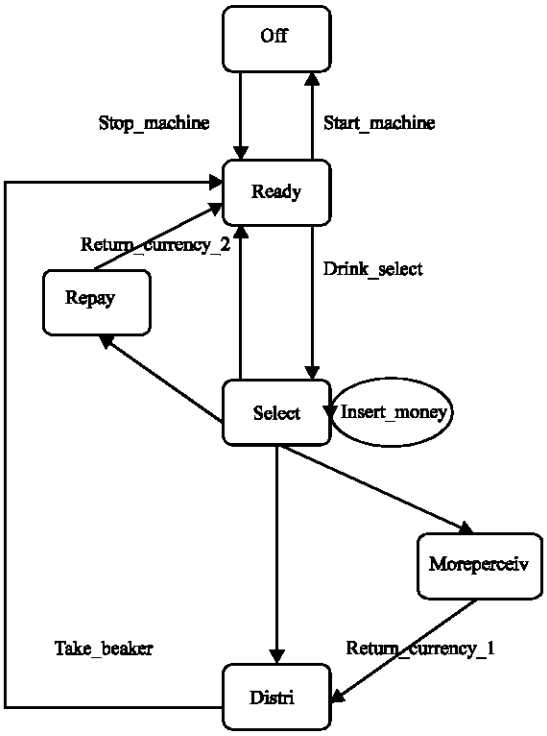

Fig. 3: Transition system of drink distributor based on Abstract Interpretation with guards

For the SCSI-2 system, we distinguish two states ρ_0 and ρ_1 corresponding to the guard of event ctr_cmd and the guard of event disk_rec, respectively (Fig. 2).

$$\rho_0 = \exists jj\ (jj : DSK \wedge buf(jj) < maxi)$$
$$\rho_1 = \exists jj\ (jj : DSK \wedge buf(jj) > 0)$$

If there exists a disk whose buffer is not full, the state ρ_0 contains all the states of the system. The state ρ_1, 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 mothods cited previousely, are not sufficient.

To go further, it is necessary to have the most abstract bahaviour 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

```
MACHINE
    DISTRI3
SETS
    DRINKS ={nothing, coffee, tea, choc};
    PIECES={HalfEuro, OneEuro, TwoEuro, FiveEuro};
    ETM = {off, ready, select, distri, moreperceiv, repay}

CONSTANTS
    price, value

PROPERTIES
    price: DRINKS--> NAT &
    value: PIECES --> NAT &
    price(nothing) = 0 &
    price(coffee)=300 &
    price(tea)=200 &
    price(choc)=250 &
    value(HalfEuro)=50 &
    value(oneEuro)=100 &
    value(TwoEuro)=200 &
    value(FiveEuro)=500

VARIABLES
    drink, sum, display, etm

INVARIANT
    drink: DRINKS&
    sum : 0..max(ran(price))+
            max(ran(value)) &
    display= price(drink)-sum&
    etm : ETM

INITIALISATION
    drink, sum, display,
    etm := nothing, 0, 0, off

OPERATIONS
    Start_machine=
        SELECT
            etm = off
        THEN
            etm := ready
        END
;
stop_machine =
        SELECT
            etm = ready
        THEN
            etm := off
        END
;
select_drink =
        SELECT
            etm = ready
        THEN
            ANY bb WHERE
                bb : DRINKS &
                bb /= nothing
            THEN
                drink := bb ||
                sum := 0 ||
                Display:= price(bb)||
                etm := select
            END
        END
;
insert_money =
        SELECT
            etm = select &
                sum < price(drink)
        THEN
            ANY cc WHERE
                cc : PIECES
            THEN
                sum:=sum + value(cc)||
                display := display-value(cc)
            END ||
            etm := select
        END
;
distribute =
        SELECT
            etm = select &
                sum >= price(drink)
        THEN
            IF sum =price(drink)THEN
                etm := distri
        ELSE
                etm := moreperceiv
        END
END
;
```

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

take_beaker=
        SELECT
            etm: = moreperceiv
        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 ||
            dtm := ready
    END
END
```

The guards of events of the drink distributor system are:

g0 ≡ etm=stop
g1 ≡ etm=ready
g2 ≡ (etm = ready) ∧ ∃(bb).(bb∈DRINKS)∧
¬(bb=nothing)
g3 ≡ (etm = selec t∧ sum < price (drink)) ∧
∃(cc).(cc∈PIECES)
g4 ≡ (etm=select ∧ 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:

σ0≡(etm=stop), σ1≡(etm=ready),
σ2≡(etm=select), σ3≡(etm=moreperceive),
σ4≡(etm=distri), σ5≡(etm=repay)

The states of drink distributor system are the states specified by the predicates σ1∧I, σ2∧I, σ3∧I,σ4∧I et σ5∧I. They are significative if they are not empty. The states φ0, φ1, φ2, φ3, φ4 et φ5 are significative with:

φ0≡(etm=stop), φ1≡(etm=ready),
φ2≡(etm=select), φ3≡(etm=moreperceive),
φ4≡(etm=distri), φ5≡(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

1. Abrial, J.R., 1996. The B-Book : Assigning Programs to Meamings, Cambridge University Press.
2. Abrial, J.R. and L. Mussat, 1998. Introducing dynamic constraints in B. In: Proc. of B'98: Recent Advances in the Development and Use of the B Method, LNCS 1393, Springer-Verlag, pp: 83-128.
3. Bert, D. and F. Cave, 2000. Construction of finite labelled transition systems from B abstract systems. In: Proc. of Integrated Formal Methods, LNCS 1945, pp: 235-/254, Springer Verlag, 2000.
4. Butler, M.J., 1999. CSP2B: A Practical approach to combining CSP and B. In: Proc. of FM'99 WORLD CONGRESS, LNCS 1708, pp: 490-508, Springer-Verlag, 1999.

5. Bharadwaj, R. and C. Heitmeyer, 1999. Model Checking Complete Requirements Specifications Using Abstraction. Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.
6. Graf, S. and H. Saidi, 1997. Construction of abstract state graphs with PVS. In: Proc. of Computer Aided Verification, LNCS 1254, pp: 72-83. Springer-Verlag, 1997.
7. Harel, D., 1987. Statecharts: A visual formalism for complex systems. Science of Computer Programming, pp: 231-274.
8. Harel, D. and A. Naamad, 1996. The statemate semantics of statecharts. ACM Transactions on Software Engineering and Methodology, 5: 293-333.
9. Hoare, C.A.R., 1985. Communicating Sequential Processes, Prentice Hall.
10. Leuschel, M. and M. Butler, 2003. ProB: A model checker for B machines. In: Proc. FM'2003, LNCS 2805, pp: 855-874, Springer-Verlag, 2003.
11. Schneider, S. and H. Treharne, 2002. Communicating B machines. In: Proc. of ZB2002: Formal Specification and Development in Z and B, LNCS 2272, pp: 416-435, Springer-Verlag 2002.
12. Sekerinski, E., 198. Graphical design of reactive systems. In: Proc. of B'98 : Recent Advances in the Development and Use of the B Method, LNCS 1393, pp: 182-197, Springer-Verlag,1998.
13. Tatibouet, B. and A. Hammad, 2002. Génération de diagrammes de classes UML à partir de machines abstraites. In: Proc. JIE1'02:Les 1ères Journées d'Informatique pour l'Entreprise 04-06 Mars 2002, Université de Blida, Algérie.