International Journal of Computer Applications (0975 – 8887)
Volume 37– No.10, January 2012

Transforming DATA* with Dotty Format to Aggregate Region Automaton

H. Hachichi
MISC Laboratory
University Mentouri
Constantine, Algeria

I. Kitouni
MISC Laboratory
University Mentouri
Constantine, Algeria

D.E. Saidouni
MISC Laboratory
University Mentouri
Constantine, Algeria

ABSTRACT
In this paper we propose an approach for translating DATA* structure of a high number of states to aggregate region automaton. Firstly, we propose a program written in python language that transforms a DATA* structure, presented as a dotty file, to a DATA* structure written in the form of a python file respecting the syntax of AToM3. Secondly, we define a meta-model of the DATA* model and a meta-model of the aggregate region automata model thus a transformation grammar using graph transformation and the modeling tool AToM^3 to perform this transformation automatically.

General Terms
Formal methods, Graph transformation, Real time systems.

Keywords
Formal validation, Graph transformation, DATA*, Region automata, Aggregate region automata, AToM^3.

1. INTRODUCTION
Distributed applications, such as communication protocols which apprehend the temporal aspect, are characterized by their big complexity.

Therefore, systems validation (verification and testing) always take a special interest among the research areas in computer science [1] [7] [20].

In this context the Durational Action Timed Automata model (DATA*) [17] [13] has more interest. This model takes into account temporal and structural non-atomicity of actions; it is based on maximality semantics [16].

Timed models consider a dense time domain, however; the state space generated in this case is infinite. The region automata imitate the infinite execution of these timed models by a finite set of transitions. They are very used in verification and system testing [19].

In this paper, firstly, we propose a program written in python language that transforms a DATA* structure, presented as a dotty file, to a DATA* structure written in the form of a python file respecting the syntax of AToM3. The aim of this transformation is to consider DATA* structures with a high number of states. Secondly, we propose an approach and a tool for transforming DATA* to an aggregate region automaton using AToM3 which is a graph transformation tool. This study is based on an aggregation region automata procedure to reduce the combinatorial explosion of regions [11][18].

This paper is organized as follows. In section 2 and 3, we recall some basic concepts about Durational Action Timed Automata model, region automata, and the aggregate region automata. In section 4 we explain model transformation concepts and especially graph transformation with its main tools and methods. In section 5, we describe our approach. An example is dealt in section 6. The final section concludes the paper and gives some perspectives.

2. DATA* MODEL
The DATA* model (Durational Action Timed Automata) [4] is a temporal model defined by a timed transitions system over an alphabet representing actions to be executed. This model takes into account in the specification, the duration of actions based on an intuitive idea: temporal and structural non-atomicity of actions. This model seems interesting and funneling more and more research [4] because it coated models of timed automata by maximality semantics [16].

The DATA* model, as the temporized models takes in charge the notions of urgency and deadlines as temporal constraints of the system. Figure1 illustrates an example of this model:

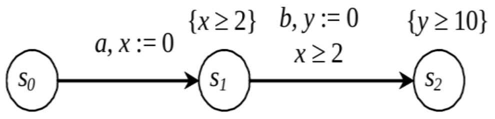


Fig 1: DATA* (a)

The durations associated to the actions are represented by constraints on the transitions and in the states targets of each one. In this sense, any enabled transition represents the beginning of the action execution. On the target state of transition, a timed expression means that the action is possibly under execution. From operational point of view, each clock is associated to an action. This clock is reset to 0 at the start of the action and will be used in the construction of the temporal constraints as guard of the transitions.

The Figure 1 presents a system of two consecutives actions *a* and *b*, the clock x is associated to the action *a*, on the locality s_1 the temporal expression $\{x \geq 2\}$ represents the duration of the action *a*. The end of the execution of an action is deduced implicitly in the case of an action that it is causally dependent.

The action *b* depends on *a*, so the transition is guarded by the relative duration constraint of *a*.

Other concepts of real time systems such as the deadlines and the urgency are considered [5][6].

2.1 Formalisation
Definition 1 : a DATA* A is a tuple (ACT, S, s_0, Ή, T_D, L_S) where ACT is a finite set of actions, S is a finite set of states, $s_0 \in S$ is the initial state, Ή is a finite set of clocks and T_D is a set of transitions. An edge t=(s, g, a, x, s′) represents a transition from state s to state s’ on input symbol a, x is a

clock to be reset with this transition. g is the corresponding guard which must be satisfied to run this transition. $L_S : S \to 2^{\phi_t(H)}_{fin}$ is a function which gives to each state s a set of actions conditions potentially executed in it.

Definition 2: The semantic of a DATA* A is defined by associating to it an infinite transitions system S_A over $ACT \cup R^+$. A state of S_A (or configuration) is a pair <s,v> such as s is a state of A and v is a valuation over ʹH.

A valuation v is a mapping on ʹH to R^+. Let x be a clock, the valuation $v[x \leftarrow 0]$ resets clock x to 0 and each other clock y to v(y). The valuation v+d maps every clock y to v(y)+d (d ∈ R^+). A configuration $<s_0,v_0>$ is initial if s_0 is the initial state of A and ∀ x ∈ ʹH , $v_0(x)=0$.

Two types of transitions between configurations of S_A are possible and correspond respectively to time passing thus the run of transition from A.

3. REGION AUTOMATA

In this section, we resume the classical definition of region automata [1][7].

3.1 Clock Regions

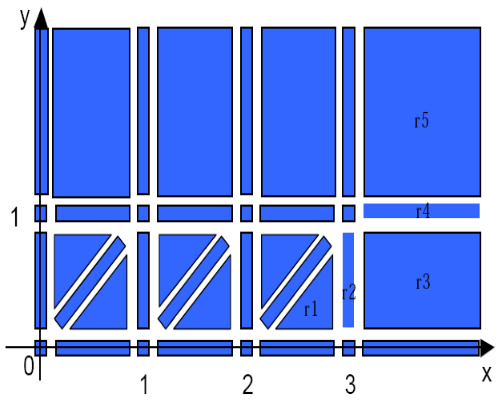
A region is a set of valuations over a finite set of clocks. Two valuations of the same region must satisfy the same constraints.

Definition 3: the region equivalence ≡ is an equivalence relation defined over the clocks valuations as follows:

≡ ', if 8 ( ) 2 ʹH
1. ( ) , '( )
2. ( ) ≤) ((b ( )c = b 0( )c) and frac(( )) = 0 , *frac*(0( )) = 0))
3. (( ) ≤ and ( ) ≤)) (*frac*(( )) ≤ *frac*(( )) ,*frac*(0( )) ≤ *frac*(0( )))

M_x is the maximum constant appearing in the constraints of clock x, and for any real *d*, b*d*c denotes the integral part of *d* and *frac*(*d*) denotes the fractional part of *d.*

Example: if we consider two clocks x, y with M_x = 3 and M_y = 1 we will have 38 regions: (Figure 2)



Successors of r1 [(2<x<3), (0<y<x<1)] :

r2 [(x=3), (0<y<1)]

r3 [(x>3), (0<y<1)]

r4 [(x>3), (y=1)]

r5 [(x>3), (y>1)]

Fig 2: Standard regions

- The time-successors succ(r) of a region r are the regions that can be reached by moving along a line drawn from some point in r in the diagonally upwards direction (parallel to the line x = y).

3.2 Region Automata

Definition 4: Let A = (ACT, S, s_0, ʹH, T_D, L_S) be a DATA*, the region automaton R_A corresponding to A is a finite automaton defined as follows:

- All localities of R_A have the form (s, r) where s ∈ S and r is a clock region. The initial locality is (s_0, r_0).
- The set of transitions T_R is,

$$T_R = \left\{ t'/t' = (s,r) \xrightarrow{a} (s',r') \middle| \begin{cases} \exists s \xrightarrow{g,a,x} s' \in T_D \text{ and } \exists r'' \in succ(r) \\ \text{such as } r \subseteq g \text{ and } r' = r''[x \leftarrow 0] \end{cases} \right\}$$

(1)

We present an example of the region automaton associated to a DATA* A in Figure 3 and Figure 4.

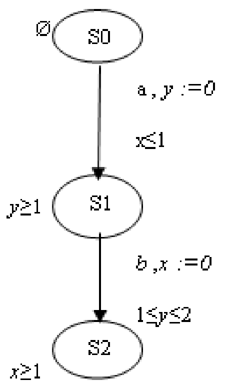


Fig 3: DATA* A

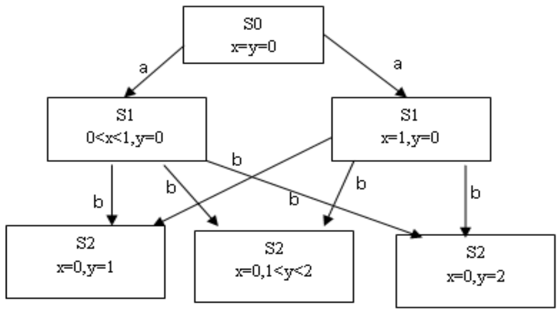


Fig 4: Region automaton associated to a DATA* A

3.3 Aggregate Region Automata

An aggregation operation based on a BI-relation between the localities of region automata has been defined in [18]. This operation consists to substitute the grouped localities by a new locality (s,R) where R is the summation over the regions r_i. This summation is defined as follows:

$$R = \oplus\ (r_1, r_2, \ldots. r_k) = \oplus_k (r_k) = \begin{cases} r & \text{if} \quad k = 1 \\ r_1 \cup r_2 \cup \ldots. \cup r_k & \text{if} \quad k > 1 \end{cases}$$

(2)

The union operation $\cup$ is defined like the one in integer intervals.

The localities are regrouped iff they have the same "forward mirror" and "backward mirror" transitions [18].

It was proved in [18] that, from the specification of DATA* model we can generate an aggregate region automaton by graph transformation. The rules of this transformation are based on a translation of the guards and a reset of clocks for the target state of transition.

Example: Figure 5 presents the aggregate region automaton associated to the DATA* A of Figure 3.

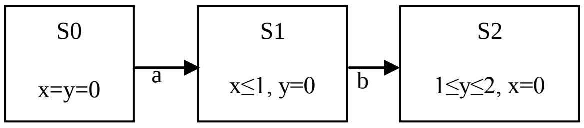
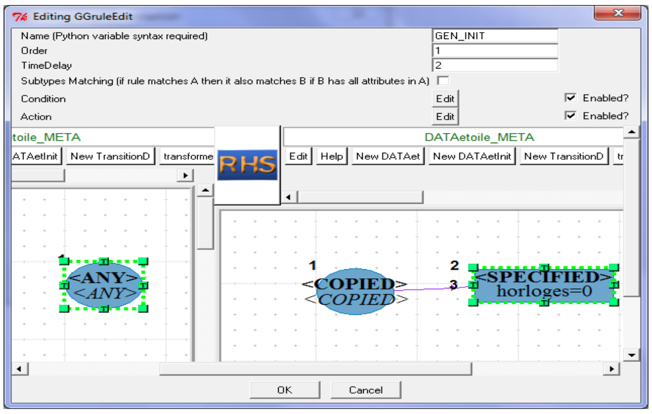


Fig 5: Aggregate region automaton associated to A

4. GRAPH TRANSFORMATION

The transformation between models is a process that converts a model to another model. This task requires a set of rules that define how the source model has to be analyzed and transformed into other elements of the target model. The transformation operation takes the source model in input then executes the rules of transformation and generates the target model in output.

Graph Grammars [15] are used for model transformation [3] [8] [12]. They are composed of production rules; each one have graphs in their left and right hand sides (LHS and RHS) (Figure 6). Rules are compared with an input graph called host graph. If a matching is found between the LHS of a rule and a sub graph in the host graph, then the rule can be applied and the matching sub graph of the host graph is replaced by the RHS of the rule. Furthermore, rules may also have a condition that must be satisfied in order to apply the rule, as well as actions to be performed when the rule is executed. A rewriting system iteratively applies matching rules in the grammar to the host graph, until no more rules are applicable. AToM^3 [2] [9] is a graph transformation tool among others. In this paper we use it.


Editing GGruleEdit

Name (Python variable syntax required) GEN_INIT
Order 1
TimeDelay 2
Subtypes Matching (if rule matches A then it also matches B if B has all attributes in A) ☐
Condition Edit ☑ Enabled?
Action Edit ☑ Enabled?

Fig 6: A grammar rule (LHS and RHS)

5. THE APPROACH

In order to allow the use of DATA* structures with a high number of states and transitions, we propose firstly a program written in python language that transforms a DATA* structure, presented as a dotty file [10], to a DATA* structure written in a python file [14] which respect the syntax of AToM3. Also we define a meta-model of the DATA* model and a meta-model of the aggregate region automata model thus a transformation grammar. The meta-model is represented by UML class diagrams and the constraints are expressed using python language.

5.1 Generation of a DATA* Respecting the Syntax of AToM^3

Figure 7 presents an example of DATA* structure with the graph editor dotty [10], the translation from a dotty representation to a python representation (Figure 8.b) is done by the python program 'DATAstarDotty2DATAstarPython.py' (Figure 8.a).

A graphical representation of DATA* structure with AToM3 is presented in Figure 9.

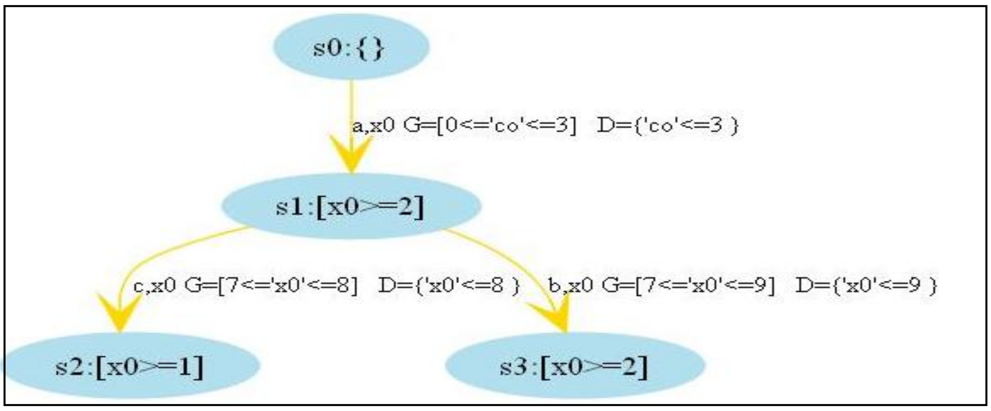
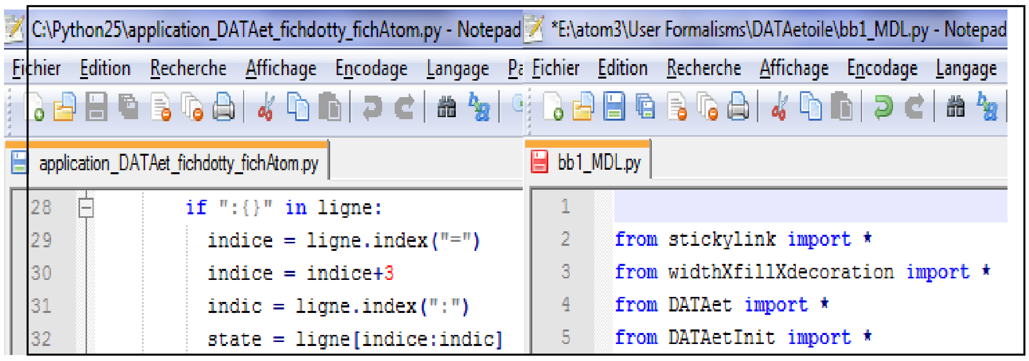

s0:{}
a,x0 G=[0<='co'<=3] D={'co'<=3 }
s1:[x0>=2]
c,x0 G=[7<='x0'<=8] D={'x0'<=8 }
b,x0 G=[7<='x0'<=9] D={'x0'<=9 }
s2:[x0>=1]
s3:[x0>=2]

Fig 7: A dotty representation of a DATA* A


C:\Python25\application_DATAet_fichdotty_fichAtom.py - Notepad
*E:\atom3\User Formalisms\DATAetoile\bb1_MDL.py - Notepad
Fichier Edition Recherche Affichage Encodage Langage
application_DATAet_fichdotty_fichAtom.py
bb1_MDL.py

```
        if ":{}" in ligne:
            indice = ligne.index("=")
            indice = indice+3
            indic = ligne.index(":")
            state = ligne[indice:indic]
```

```

from stickylink import *
from widthXfillXdecoration import *
from DATAet import *
from DATAetInit import *
```

(a) (b)

Fig 8: Translation step (dotty-python) of a DATA* A

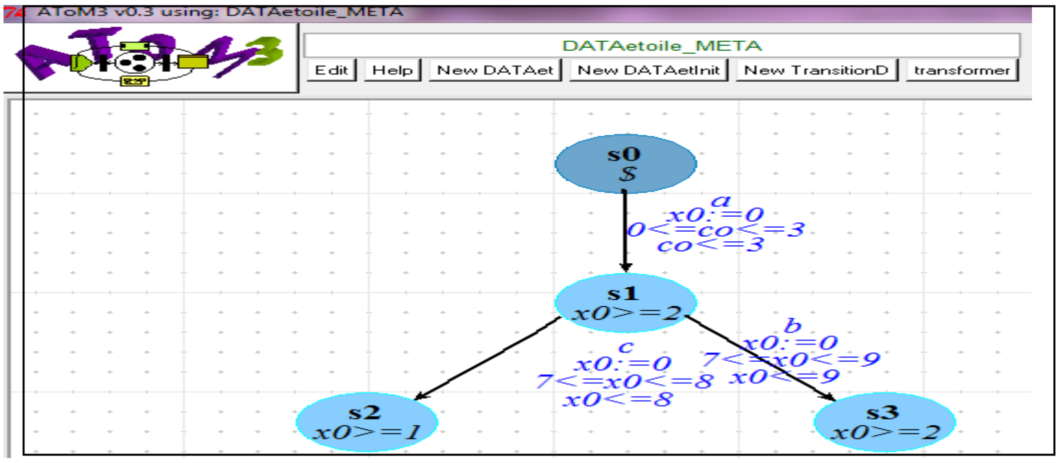


Fig 9: A graphical representation of a DATA* A with AToM3

5.2 DATA* Meta-Model

The first meta-model proposed is a class diagram composed of the following classes (Figure 10):

1) *DATAet class*: it represents the states of DATA*, each state has two attributes: a name (name), and duration conditions (CD). It is connected to TransitionD and DATAetInit by inheritance link.

2) *TransitionD association*: it represents the transitions of DATA*, each transition is identified by an action, a clock and a guard.

3) *DATAetInit class*: it represents the initial state of DATA*, it inherits the attributes from DATAet.

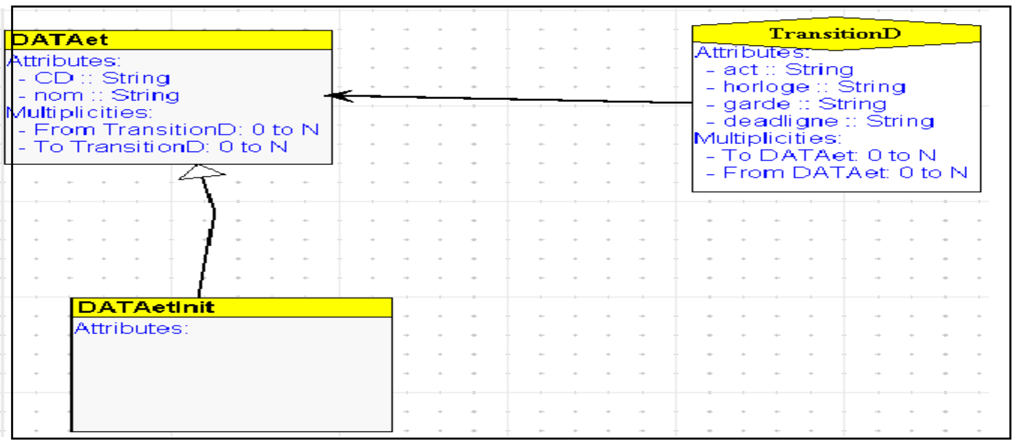


Fig 10: DATA* meta-model

5.3 Aggregate Region Automata Meta-Model

The second meta-model is a class diagram composed of the following classes (Figure 11) :

1) ARstate class: it represents the localities of the aggregate region automaton, each locality has two attributes: a name (name), and a clock region (RegionHorloge). It is connected to ARtransition and ARStateInit by inheritance link.

2) ARtransition association: it represents the transitions of the aggregate region automaton, each transition is identified by an action.

3) ARStateInit class: it represents the initial locality of the aggregate region automaton; it inherits the attributes from Arstate class.

Each class has an only graphical appearance.

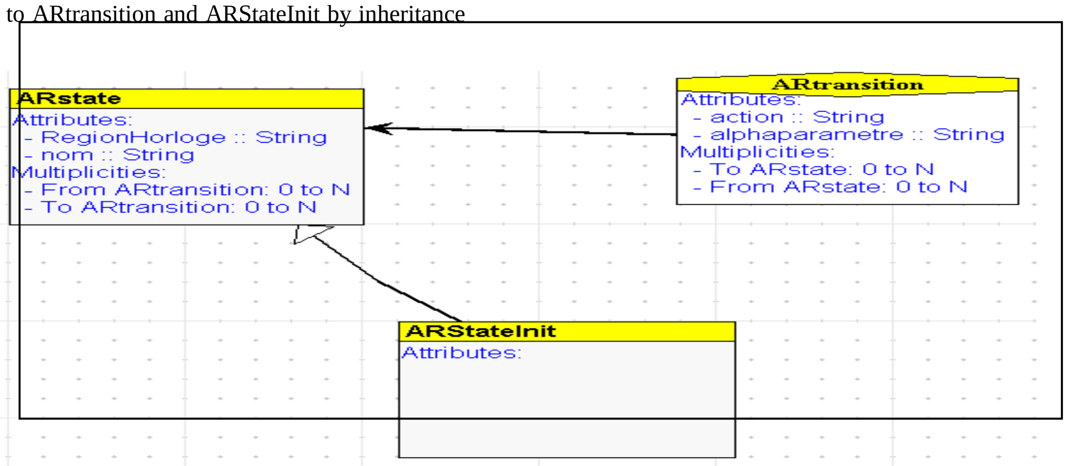


Fig 11: Aggregate region automata meta-model

5.4 Modeling Tool of Data* and the Aggregate Region Automata

The two meta-models defined previously are represented in AToM3 (Figure 10, Figure 11) and allow to generate a tool for modeling systems in DATA* and the aggregate region automata (Figure 12).

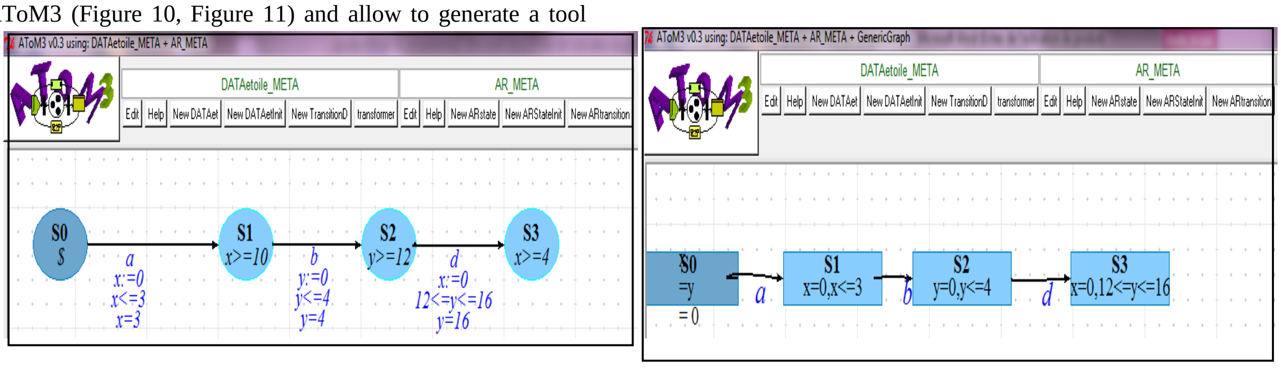


Fig 12: Modeling tool of DATA* and the aggregate region automata

5.5 Graph Grammar

The proposed graph grammar is composed by 9 rules organized in 3 categories.

The first three rules allow the construction of an aggregate region automaton based on the principle detailed in Section 3.3.

•The 1st rule is used to generate the first region associated to the initial state of DATA* where all clocks are reset to zero. (Figure 13).

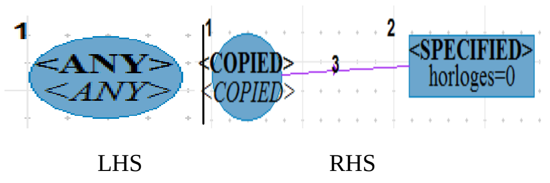


Fig 13: Generating the initial locality of an aggregate region automaton (Rule 1)

•Rules 2 and 3 (Figure 14) generate the rest of the aggregate region automaton and save it in a text file (Figure 20).

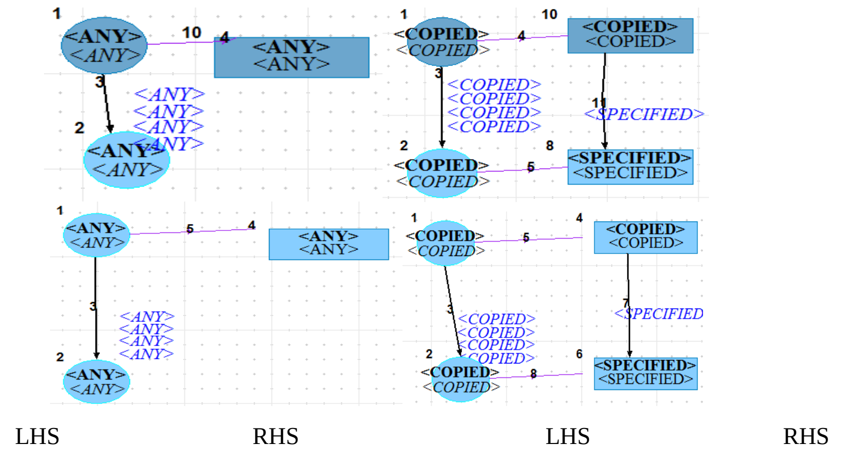


Fig 14: Generating localities of an aggregate region automaton and saving it in a text file (Rules 2 and 3)

•Rules 4 and 5 eliminate a generic links between the source model and the target model (Figure 15).

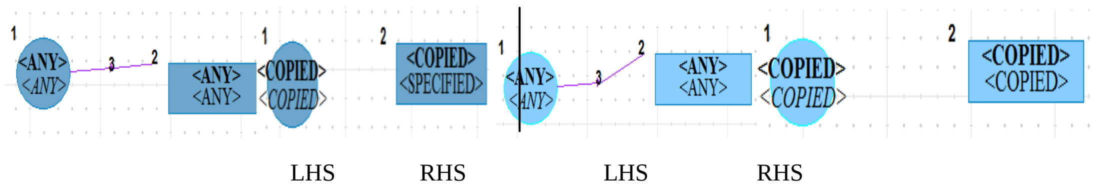


Fig 15: Removing generic links (Rules 4 and 5)

•Rules 6, 7, 8 and 9 (Figure 16) eliminate the graphical representation of DATA* model.

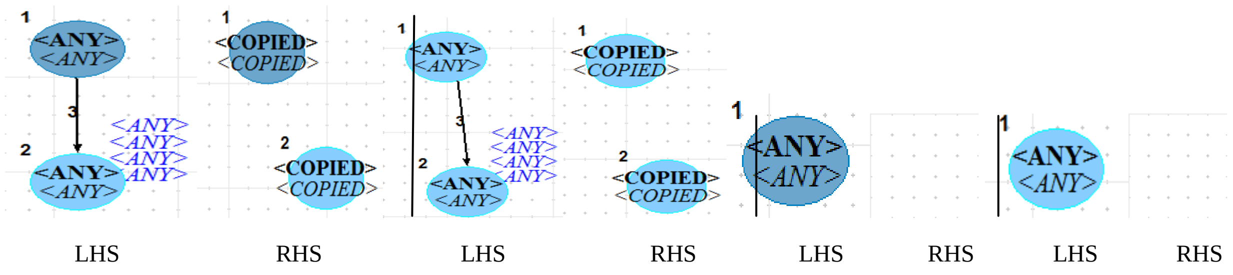


Fig 16: Removing DATA* model (Rules 6,7,8 and 9)

6. EXAMPLE

To illustrate our approach we propose the example of the ticket reservation system "TRS". This example supposes that to buy a ticket, we generally pass by two counters. The first counter R is for making a reservation and the second counter C is for paying and taking the ticket. This agency has one waiting room, three counters of type R and two of type C.

On arrival, the client goes to the waiting room, when a counter of type R is free, he can make a reservation. Once the operation is complete, he waits until a counter C becomes free for paying and taking the ticket.

Figure 17 presents a DATA* of TRS for two clients with the graph editor dotty.

The mapping of this DATA* with the graph editor dotty to the equivalent DATA* model of Figure 18 is performed using python program.

We have applied our tool on the DATA* model and obtained automatically the aggregate region automaton of Figure 19. The result is saved in the text file of Figure 20.

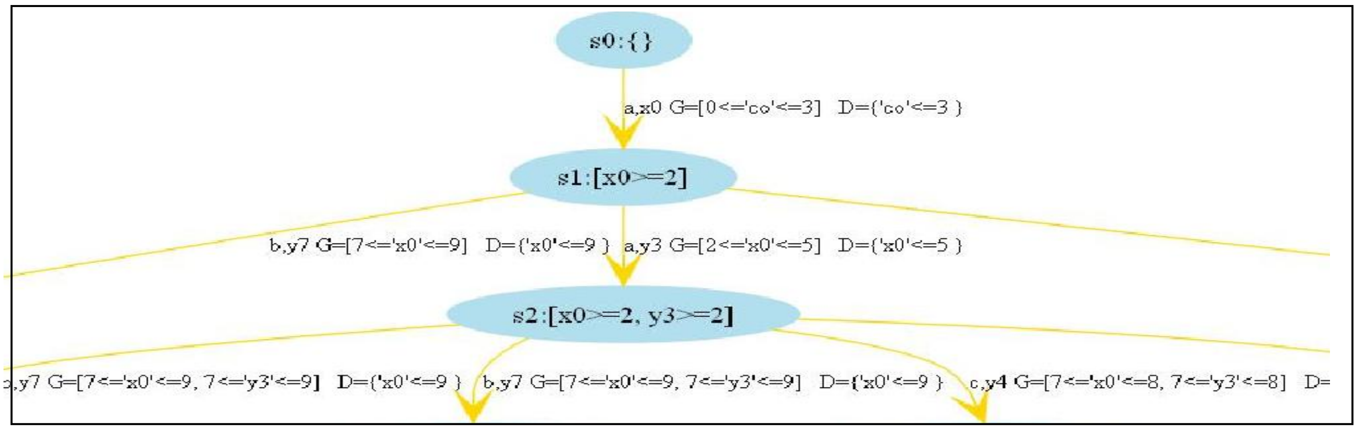

s0:{}
a,x0 G=[0<='co'<=3] D={'co'<=3 }
s1:[x0>=2]
b,y7 G=[7<='x0'<=9] D={'x0'<=9 }
a,y3 G=[2<='x0'<=5] D={'x0'<=5 }
s2:[x0>=2, y3>=2]
b,y7 G=[7<='x0'<=9, 7<='y3'<=9] D={'x0'<=9 }
b,y7 G=[7<='x0'<=9, 7<='y3'<=9] D={'x0'<=9 }
c,y4 G=[7<='x0'<=8, 7<='y3'<=8] D=

Fig 17: DATA* of TRS with the graph editor dotty

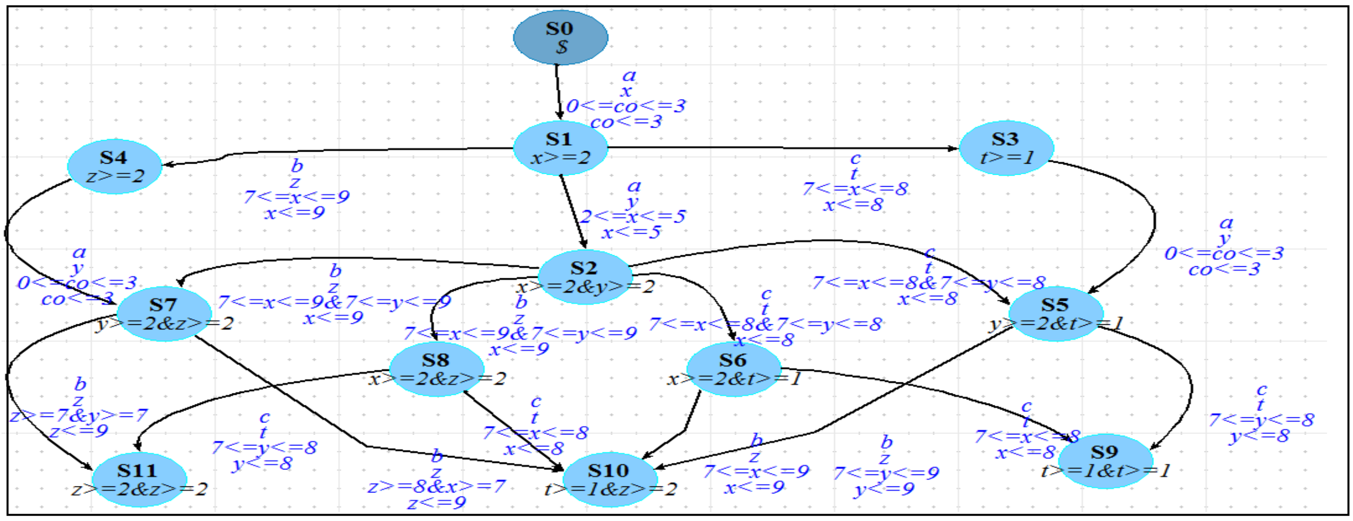


Fig 18: DATA* of TRS with AToM3

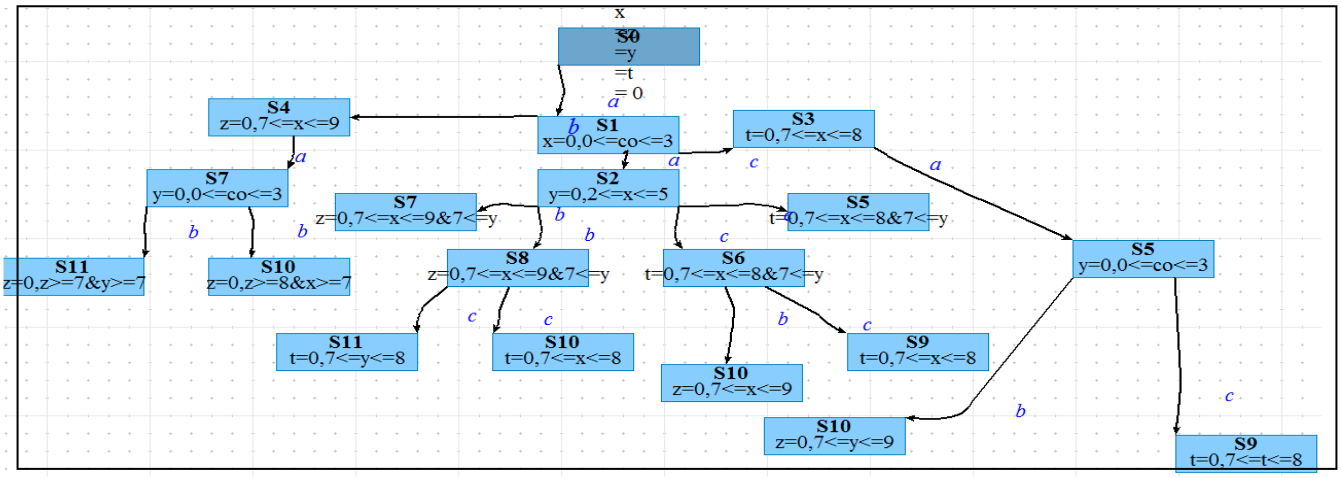
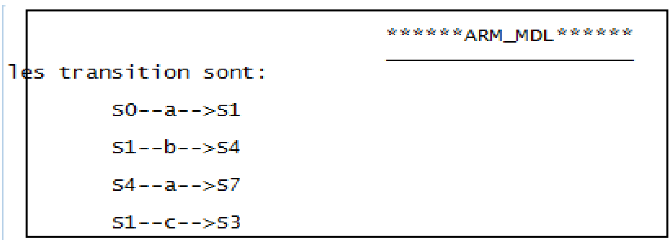


Fig 19: Aggregate region automaton


```
                                   ******ARM_MDL******

les transition sont:
        S0--a-->S1
        S1--b-->S4
        S4--a-->S7
        S1--c-->S3
```

Fig 20: A textual aggregate region automaton

7. CONCLUSION

In this paper we proposed a method for generating an aggregate region automaton from a DATA* by the graph transformation approach and using the environment AToM3 in order to provide a finite abstraction of DATA* structures with a high number of states.

Firstly, we have proposed a program written in python language that transforms a DATA* structure, presented as a dotty file, to a DATA* structure written in the form of a python file respecting the syntax of AToM3. We have proposed also two meta-models; one for the input model and the other for the output model. Based on these meta-models, we have proposed a graph grammar that deals with the transformation process. The meta-modeling tool AToM3 is used for this purpose. We have illustrated our approach through an example. In future work, we plan to implement our approach with other tools as AGG in order to compare performances. We plan also to Study the complexity of this transformation and its use in system testing.

8. REFERENCES

[1] Alur, R., Dill, D. L.1994. A theory of timed automata. Theoretical Computer Science, 126(2):183-235.

[2] AToM^3 Home page, version 3.00, http://atom3.cs.mcgill.ca/

[3] Baresi, L., Hekel, R. 2004.Tutorial Introduction to graph transformation. A software Engineering perspective, Lecture Notes in Co0mputer Science, Volume 3256/2004, Springer Berlin, pp.431-433.

[4] Belala, N. 2010. Modèles de Temps et leur Intérêt à la Vérification Formelle des Systèmes Temps-Réel. PHD's thesis, Mentouri University, 25000 Constantine, Algeria.

[5] Bornot, S., Sifakis, J., Tripakis, S.1997. Modeling urgency in timed systems. In Proc. International Symposium Compositionality (COMPOS'97), volume 1536 of LNCS. Springer-Verlag.

[6] Bornot, S., Sifakis, J. 1998. On the composition of hybrid systems. In Proceedings of HSCC"98", volume 1386 of LNCS, Springer-Verlag, pp. 69–83.

[7] Bouyer, P. 2002. Modèles et Algorithmes pour la Vérification des Systèmes Temporisés, PhD thesis. Laboratoire Spécification et Vérification – CNRS UMR 8643 & ENS de Cachan 61, avenue du Président Wilson – 94230 Cachan – France.

[8] Czarecki, K., Helsen, S. 2006. Feature-based survey of model transformation approaches. IBM SYSTEMS JOURNAL, VOL 45, NO 3.

[9] De Lara, J., Vangheluwe, H. 2002. AToM^3: A Tool for Multi-Formalism Modeling and Meta-Modeling. Proc. Fundamental Approaches to Software Engineering, FASE'02, Vol. 2306. LNCS. Grenoble, France, pp. 174-188.

[10] Graphviz Home page,http://www.graphviz.org/

[11] Hachichi, H., Kitouni, I., Saïdouni, D. E. 2011. A Graph Grammar Approach for calculation of Aggregate Regions Automata. The International Arab Conference on Information Technology (ACIT).

[12] Karsai, G., Agrawal, A. 2004. Graph Transformations in OMG's Model-Driven Architecture. Lecture Notes in Computer Science, Vol 3062, Springer Berlin / Heidelberg, pp.243-259.

[13] Kitouni, I. 2008. Déterminisation des automates temporisés avec durées d'actions pour le test formel. Master's thesis. Mentouri University, 25000 Constantine, Algeria.

[14] Python Home page, htpp://www.python.org.

[15] Rozenberg, G. 1997. Handbook of Graph Grammars and Computing by Graph Transformation, vol 1: Foundations, World Scientific.

[16] Saïdouni, D. E., Courtiat, J. P. 2003. Prise en Compte des Durées d'Action dans les Algèbres de Processus par l'Utilisation de la Sémantique de Maximalité. In CFIP.2003. Hermes, France.

[17] Saïdouni, D. E., Belala, N. 2006. Actions duration in timed models. The International Arab Conference on Information Technology (ACIT).

[18] Saïdouni, D. E., Kitouni, I., Hachichi, H. 2011. Formalisation du calcul de l'automate des régions agrégé d'un automate temporisé avec durées d'actions. MISC REPORT 11001. Mentouri University, 25000 Constantine, Algeria.

[19] Springntveld, J., Vaandrager, F., D'Argenio, P. 2001. Testing timed automata. Theoretical Computer Science, 254.

[20] Stainer, A. Test d'automates temporisées. 2010. Master's thesis, INRIA Rennes, France.