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 AToM3 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, AToM3.

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 infinite execution of these timed models is a problem. 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 funneled 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. Figure 1 illustrates an example of this model:

\[
\begin{align*}
S_0 & : a, x := 0 \\
b, y := 0 & : x \geq 2 \\
S_1 & : y \geq 10
\end{align*}
\]

Figure 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 \(\{x\leq 2\}\) 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 \((ACT, S, s_0, H, T_D, L_S)\) where \(ACT\) is a finite set of actions, \(S\) is a finite set of states, \(s_0\in S\) is the initial state, \(H\) is a finite set of clocks and \(T_D\) is a set of transitions. An edge \(\texttt{e}(s, g, a, s')\) 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.

\[ L_S : S \rightarrow 2^{H(S)} \] 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 \( \langle s, v \rangle \) such as \( s \) is a state of \( A \) and \( v \) is a valuation over \( H \).

A valuation \( v \) is a mapping on \( H \) 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 \( v(y)+d \) (\( d \in R^+ \)). A configuration \( \langle s_0, v_0 \rangle \) is initial if \( s_0 \) is the initial state of \( A \) and \( \forall x \in H \), \( v_0(x)=0 \).

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].

- 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 \( A = (ACT, S, s_0, 'H', T_0, L_0) \) 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 \( s \in S \) and \( r \) is a clock region. The initial locality is \( (s_0, r_0) \).
- The set of transitions \( T_R \) is:

\[
T_R = \{ (s', r', a) \rightarrow (s, r) \mid \exists s, s', r \in S \text{ and } r \text{ is a clock region such as } t \leq g \text{ and } r' = r'[x \leftarrow 0] \} \\
(1)
\]

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

### 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 \( \equiv \) is an equivalence relation defined over the clock valuations as follows:

\[
\equiv \quad \text{if } b(\text{ }) \leq \text{ and } (\text{ }) \leq (\text{ }) \quad (\text{frac}(\text{ }) \leq \text{frac}(\text{ }) \leq \text{frac}(\text{ }))
\]

\( M_i \) is the maximum constant appearing in the constraints of clock \( x \), and for any real \( d \), \( bdc \) denotes the integral part of \( d \) and \( \text{frac}(d) \) 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)

\[
\text{Successors of } r_1 \left[ (2 < x < 3), (0 < y < 1) \right]:
\]

\[
r_2 \left[ (x=3), (0 < y < 1) \right] \\
r_3 \left[ (x > 3), (0 < y < 1) \right] \\
r_4 \left[ (x > 3), (y = 1) \right] \\
r_5 \left[ (x > 3), (y > 1) \right]
\]

**Fig 2**: Standard regions
### 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:

\[
R = \bigoplus (r_1, r_2, \ldots, r_k) = \bigoplus_k (r_k) =
\begin{cases}
  r & \text{if } k = 1 \\
  r_1 \cup r_2 \cup \ldots \cup r_k & \text{if } k > 1
\end{cases}
\]

(2)

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

The localities are regrouped if 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](image)

**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. ATOM3 [2] [9] is a graph transformation tool among others. In this paper we use it.**

![Fig 6: A grammar rule (LHS and RHS)](image)

### 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 ATOM3

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.
Fig 7: A dotty representation of a DATA* A

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.
5.3 Aggregate Region Automata Meta-Model

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

1) \textit{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 \textit{ARtransition} and \textit{ARStateInit} by inheritance link.

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

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

Each class has an only graphical appearance.

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).

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).

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

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

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

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

Fig 18: DATA* of TRS with AToM3

Fig 19: Aggregate region automaton

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