Abstract

Besides being adopted as the new general data representation format for the Internet, XML is finding increasing acceptance as a native data exchange language. In order to fully exploit its potential XML data streams management systems have to be fully equipped with data verification and analysis capabilities.

In this paper, we propose a solution to this need through a framework for the history checking and model checking of the data. The proposed method uses a linear temporal logic, called TRIO, to describe data constraints and properties. The constraints are automatically translated in Promela, the input language of the model checker SPIN, to verify the properties of sequences of XML data sets.

1 Introduction

During the recent years, we have seen the dramatic development of the eXtensible Markup Language (XML) as a major standard for storing and exchanging information. XML enables the self-description of hierarchies of semi-structured information, by intermixing data content with semantic tags which describe such content. It is easy to foresee that, in the near future, a large portion of available data will be represented in XML and that XML will become the standard language for data streams.

In many modern applications, data may take the form of infinite data streams, rather than finite stored data sets. Several aspects of data management need to be reconsidered in the presence of data streams, offering a new research direction for the database community. In this paper we focus primarily on the problem of defining methods to describe and verify properties on data and define methods capable to alert the user if some of these properties are violated.

One can envision streams of XML data flowing throughout the Internet: a stream of stock quotes or minute-by-minute updates on positions of a fleet of vehicles - one XML fragment per vehicle report. In particular XML dialects for financial data spread out in the last years: FinXML [2], ebXML [1], XFRML [6], FpML [3] and XBRL [5] are some examples. Especially in this context we need to specify complex business rules regarding XML streams. Such rules predicate on historical trends of values. At the state of the art those rules are checked only by ad-hoc applications, which correctness and reliability are related only to the test cases run. It is easy to foresee that the spreading of the use of XML in such contexts will cause an increasing interest on formal methods and techniques applied to XML data.

However the high number of possible scenarios in which XML is employed to described information makes hard the choice of an appropriate and enough powerful formalism to describe the specification depending on the scenario the data have to satisfied. More precisely, during the years formal methods developed many different specification languages depending on the particular field, e.g. statecharts, dataflow, many different kinds of specific logic. In this work, due to the multiplicity of applications in which data are employed, a general purpose specification language, such as a powerful logic, is needed. Then the properties and the specification that a set of data has to satisfy is always strictly connected with temporal constraints. Hence the chosen specification language has to have the capability to describe temporal behaviors. The temporal description need not only to be able to describe the ordering among events, such as in LTL, but also to describe specific and detailed temporal constraints among data. For these reasons we consider as specification language a first order linear time temporal logic with both past and future modalities and a quantitative metric on time, called TRIO [13]. The logic TRIO can also be enriched with constructs, inspired by Object-Oriented Analysis and Design, for structuring specifications into a set of modules with clearly defined interfaces, thus providing a very useful support to the structuring and management of specification of highly complex systems, and at the same time building a bridge from requirements specification to high-level design.

Over the years a variety of methods and tools have been defined to support typical activities in TRIO. Validation of TRIO specification is obtained through generation of execu-
tion traces or checking of such simulations for consistency against the TRIO specification. The execution traces derived from TRIO specifications, suitably classified and annotated, can be employed as functional test cases to support post-design verification. A more systematic and general means of validation and verification can be pursued through proof of properties derivable from the TRIO specification. As TRIO is a first order logic that includes arithmetic on the temporal domain, it is undecidable in the general case; hence two basic approaches were devised to address the goal of providing mechanical support to the verification of TRIO specifications. One consists of adopting a deductive approach, based on the definition of a suitable axiomatization of the logic and on its encoding in the notation of a general purpose theorem prover, such as PVS [12]; this allows the construction of a tool supporting the semiautomatic (i.e., manual with assistance from the tool) derivation of system properties in the form of theorems. In this approach one maintains the generality and expressive power of the full language, at the price of sacrificing the construction of a completely automatic (so-called push-button) tool. Another, complementary approach to verification aims at the construction of tools that are fully automatic, or at least provides a quite strong support to the designer: it consists of defining a decidable approximation of a specification, upon which applying methods and algorithms for deciding satisfiability or, less generally but more efficiently, for checking satisfaction with respect to a given interpretation structure. In the past, the latter approach has been based on finitizing the domains of the variables that appear in the specifications [18], leading to the construction of tools built either on tableaux-based verification procedures [11, 14, 18] or on the encoding of TRIO into propositional languages and the use of sophisticated SAT-solvers. This approach has the advantage of allowing the development of pushbutton tools but the approximations introduced to make verification decidable (and feasible) may not assure the conservation of properties of the original specification.

The basic idea is to specify by TRIO the properties that the data have to satisfy and to consider the data as a particular temporal evolution of the system, called history. More in details this approach can be considered as history checking, since it corresponds to verify if the particular evolution satisfy a given formula. Moreover since a given particular evolution in a fixed point is finite, it would be allowed to apply the method presented in [12], where no restriction on TRIO is required. However the choice of use a model checker, such as SPIN, in this work is due to two main reasons. First the model checking approach is completely automatic, and moreover the chosen model checker implements very efficient algorithms. Then the presented work is part of a more general framework that requires to deal also with infinite traces. The complete architecture of which this work

```xml
<stock date="2005_08_24">
  <share name="Google" price="12.678" minPrice="12.125" maxPrice="12.900" quantity="4000"/>
  <share name="IBM" price="12.478" minPrice="12.279" maxPrice="12.999" quantity="8000"/>
  <share name="HP" price="7.865" minPrice="6.111" maxPrice="7.999" quantity="2000"/>
</stock>
```

```xml
<exchanges>
  <exchange seller="E8886" buyer="E4535" name="IBM"
    timestamp="2005_08_26_10_03_34" price="25900" quantity="499"/>
  <exchange seller="E4566" buyer="E5667" name="IBM"
    timestamp="2005_08_26_10_03_34" price="12.675" quantity="199"/>
  <exchange seller="E4566" buyer="E5667" name="HP"
    timestamp="2005_08_26_10_45_34" price="11.111" quantity="678"/>
  <exchange seller="E8886" buyer="E4535" name="IBM"
    timestamp="2005_08_26_10_56_34" price="23.234" quantity="200"/>
</exchanges>
```

**Figure 2. Running example**

is part is presented in figure 1. Hence the basic idea is that once described the specification of the system in TRIO and generated the translation in Promela, the input language of SPIN, the framework states if the data already present satisfy the specification; if not the model checker shows where the specification is violated, if yes the history is made infinite in an exhaustive way and the particular traces in which the specification is violated are annotated. From this general plan the idea of use a model checker to make history checking. In fact in this way it is possible to use a unique tool and a unique language for both the goals.

### 1.1 Running example

Coherently with the considerations stated above, in Figure 2 we show stock performance data. Information contained in the `stocks` element describe data at the beginning of the day. Instead, data in the `exchanges` element represent the stream of stock sales during the day. Given this scenario, we want to monitor some business rules:

1. **a.** Between buying and selling of stocks of the same society should be at least a fixed temporal distance.
2. **b.** Nobody can buy more than a fixed amount of stocks of a certain company in a day.
3. **c.** No stock can be sold with a price double than the price of the stock at the beginning of the day.
4. **d.** During the day the gap between the minimum price and the maximum price cannot be greater than a fixed limit.
2 TRIO

TRIO is a typed first order logic that supports a linear notion of time. In TRIO, first-order variables and quantifiers are allowed over finite or infinite, dense or discrete domains, including the time domain. Besides the usual propositional operators and the quantifiers, one may compose TRIO formulae by using a single basic modal operator, called Dist, that relates the current time, which is left implicit in the formula with another time instant: the formula Dist(Υ, t), where Υ is a formula and t a term indicating a time distance, specifies that Υ holds in x₀ + t, where x₀ denotes the current instant which is implicitly defined. Many derived temporal operators can be defined from the basic Dist operator through propositional composition and first order quantification on variables representing a time distance. Moreover, to refer to values of a variable or term in the past or in the future the operator dist is introduced: for a given term X, dist(X, t) is the value that X had or will have at the time instant whose distance is t from now. The operators futr and past derived from dist, refer to value of a variable respectively in the future and in the past. For an accurate description of TRIO see [13].

In general TRIO formulae (adopting the power of first-order logic) are undecidable. For this work we consider a decidable subset where no variable is allowed, the time domain is the set of natural numbers and other domains are finite. Syntax is described by the following grammar, where \( \phi \) is the axiom, \( p \) any element in a finite set of atomic propositions, \( d \) any element of a finite set of natural numbers.

\[
\phi ::= p \mid \phi \land \phi \mid \neg \phi \mid Until(\phi, \phi) \mid Since(\phi, \phi) \mid Futr(\phi, d) \\
\mid Past(\phi, d) \mid Lasts(\phi, d) \mid Lasted(\phi, d)
\]

2.1 TRIO formulae for the running example

The translation in TRIO of the properties presented above is the following:

a. \( Alw(\forall a(price(a) \leq past(price(a), 1day)) \)

b. \( Alw(\forall a(price(a) \leq past(price(a), 1day)) \)

c. \( Alw(\forall p \forall a(sell(p, a) \rightarrow Lasted_{ee}(Buy(p, a), K))) \)

d. \( Alw(\forall t1, t2 \forall a(past(price(a), t1) \rightarrow past(price(a), t2) \leq k)) \)

Formulae are defined on the following domains: \( p \) is on the domain of persons, \( a \) takes value in the possible stocks and \( t \) on the timestamps in a day. All these domains can be considered finite. Notice that all the names of predicate, temporal or not, begin with a capital letter, while the names of function with lower case letters.

3 Translation into Promela and verification with SPIN

Our approach to model and history checking data, whose specifications is given in TRIO, is based on the translation of the TRIO formulae into Promela explained in details in [17, 4]. In the used approach the Promela processes obtained from the translation of the TRIO specification act globally as an acceptor of a language defined over the alphabet of the specification, and therefore they must be coupled with some additional Promela program fragments generating the values, over time, for the logical variables that constitute the specification alphabet. In this work at the first step, as shown in figure 1, we use this automaton to verify if the data that we have, opportunely represented as a finite word, is accepted or not by the automata. The proposed translation technique, combined with other minor optimizations, related for example with the management of TRIO past-time operators, allowed us to perform efficiently the verification in SPIN that very efficient for finite stream of data it is feasible also for infinite stream.

Let us consider an example. If we use the property (a) and the set of data shown in Figure 2, we have to add two channels, one for the stock information and one for the exchange information. If we try to verify if the data stream is conform to the specification, we obtain the counterexample
shown in figure 3. The first process (the one on the left) represents the history manager and the second one (on the right) the specification. Each time instant the history manager sends to the specification data $d_1$ and $d_2$ through the channel message. After evaluating the data, the specification sends to the history manager the truth value with respect to the received data. In this example at the second time instant the sent data violates the specification.

**4 Evaluation**

We now present some experiments conducted on a series of XML data streams (in the form presented in Figure 2), varying in size from 100 to 2000 data in order to evaluate the performance of our approach. Figures 4(a) and 4(b) refer to the constraints of example (a) presented in Section 1.1. The data were generated re-mapping data from a real stock quotation Web site. The size of the stream is indicated on the x-axis. Each figure corresponds to one of the typical SPIN indicators and reports one curve representing the value. Figure 4(a) shows the steps necessary to complete the process, index related to the required time. Figure 4(b) shows the required memory (in KB). These indexes are indicated on y-axis and represent the average of the measures of 10 attempts for each experiment. These results show that our approach is feasible also for a considerable size of data streams. We observe that, in our analysis, we do not have to take into account the time spent to produce the Promela version of the constraint, because it is generated only once at schema design time and thus do not interfere with run time performance. On the contrary the time needed to translate data from XML to Promela is considered.

**5 Related work**

Different approaches try to apply formal methods to the verification of temporal constraints over data. Datalog LITE [10] is a deductive query language with a linear-time model-checking algorithm. Datalog LITE is a variant of Datalog that uses stratified negation, restricted variable occurrences and a limited form of universal quantification in rule bodies. Despite linear-time evaluation, Datalog LITE is highly expressive: it encompasses modal and temporal logics such as CTL or $\mu$-calculus. [16] defines a query language for sequence databases. The language, called Sequence Datalog, extends Datalog with interpreted function symbols for manipulating sequences.
In recent years many data streams formats were defined in order to manage, exchange and analyze financial data. FinXML [2] is an XML based framework developed to support a single universal standard for data interchange. eXtensible Business Reporting Language [5] is a modular suite born with the ambition to enable enterprises of any size and in any geographical location to conduct business via Internet. XFRML (eXtensible Financial Reporting Markup Language) [6] will be the digital language of business. FpML (Financial Products Markup Language) [3] is a business information exchange standard for electronic dealing and processing of financial derivatives instruments. XBRL (eXtensible Business Reporting Language) [5] is a language for the electronic communication of business and financial data.

Since the introduction of XML, several textual query languages were proposed and analyzed by the database community. In particular, a long stream of research addresses the data stream management. Continuous queries are an important component of Tapestry [20], which performs content-based filtering over an append-only database of email and bulletin board messages. The notion of continuous queries is formalized in [8]. Xyleme [19] is an XML repository enabling very high throughput with a subset of XML Query. OpenCQ [15] and NiagaraCQ [9], support continuous queries for monitoring persistent data sets distributed on WAN. OpenCQ is based on incremental view maintenance, while NiagaraCQ proposes techniques for grouping continuous queries for efficient evaluation. STREAM [7] focuses primarily on the problem of query processing, specifically on how to define and evaluate continuous queries over data streams. Our approach can overlap with some of the objectives posed by these results: continuous queries allow to monitor the result of queries and to express constraints and business rules by means of predicates about the emptiness of the results of such queries. This approach can be very effective and offers good performance, but is not tailored to express constraints on temporal behavior of data, especially for the lack of time management functions in XML query languages.

6 Conclusions and future works

In this paper, we proposed a framework for the history checking of the data based on TRIO, a linear temporal logic, that can be used to describe data constraints and properties. We showed how the constraints can be automatically translated in Promela. Moreover we introduced a platform for the validation and verification of data of which the history checker is the basic step.

As future works we envision the possibility to use this technique to implement a framework devoted catch possibly dangerous behaviors using full model checking capabilities; investigate prediction models and use just the traces obtained by them for the model checking consider always more complex specification exploiting TRIO modularity.

References