

Real-time, Online, Low-Impact Temporal Pattern Matching

Doron Drusinky

Department of Computer Science

and

J. L. Fobes

Transportation Security Administration Chair in Security

Naval Postgraduate School

drusinsky@nps.navy.mil and jlfobes@nps.navy.mil,

Abstract

A temporal pattern matching technique based on formal specifications using Linear-Time Temporal Logic (LTL) is described. The method is based on Remote Execution and Monitoring (REM) of LTL assertions. Unlike verification applications where the formal specification is used to locate errors in a corresponding program, REM is not concerned with correctness but rather with temporal pattern detection. Unlike comparable techniques, such as SQL in temporal databases, the method is completely on-line and does not require storage of the input sequence. This makes REM especially suitable for low-impact, real-time, temporal pattern matching of potentially never ending applications such as security monitoring and financial temporal business rule checking.

1 Introduction

Temporal Logic is a special branch of modal logic that investigates the notion of time and order. Pnueli [1] suggested using LTL for reasoning about concurrent programs. Since then, several researchers have used LTL to state and measure correctness of concurrent programs, protocols, and hardware (e.g., [2], [3]).

LTL is an extension of propositional logic where, in addition to the propositional logic operators there are four future-time operators and four dual-past time operators: *always* in the future (always in the past), *eventually*, or *sometime* in the future (sometime in the past), *Until* (Since), and *next cycle* (previous cycle).

Metric Temporal Logic (MTL) was suggested by Chang, Pnueli, and Manna as a vehicle for the verification of real time systems [4]. MTL extends LTL by supporting the specification of relative-time and real-time constraints. All four LTL future-time operators (*Eventually*, *Always*, *Until*,

Next) can be characterized by relative-time and real-time constraints specifying the duration of the temporal operator.

REM is a method of tracking the behavior of an underlying application, such as an embedded system, financial database, or airline reservation systems. REM methods range from simple print statement logging methods to run-time tracking of complete formal requirements (e.g., written in LTL/MTL) for verification purposes. Indeed, first applications of REM were for verification applications where REM was and is used to track how formal specification requirements are conformed to by the actual executing system.

Recent adaptations of REM methods enable run time monitoring for non-verification purposes such as temporal business rule checking and temporal security rule checking. This adaptation uses REM methods as on-line, real-time, and low-impact temporal pattern matchers.

This paper describes the DBRover, an online, real-time, low-impact REM tool based on LTL and MTL, and its application to temporal pattern detection for financial and security applications.

2 Online Temporal Pattern Detection

Consider the following two airline security related temporal pattern rules, both concerned with detecting a foreign national male passenger with a student visa flying to the Harrisburg International Airport near the Three Mile Island nuclear power plant:

R1. Detect such a passenger if he flew to the Middle East at least once within a year of obtaining his student visa.

R2. Detect such a passenger if he flew to the Middle East at least once within a year of obtaining his student visa and he received two or more direct deposits from non-US banks within the last year.

Both rules describe temporal patterns that contain potentially discernable elements from an aviation security system operating automatically and in real-time. The two primary methods for performing such temporal pattern detection are *off-line* and *online*, as described in the sequel.

The Transportation Security Administration has an automated profiling system originally termed Computer Assisted Passenger Screening [5] that relies upon the data in each Passenger Name Record. This profiling system is being upgraded to access a more extensive range of data. The Computer Assisted Passenger Prescreening System (CAPPS II) will profile airline passengers based on secret criteria in order to identify potential terrorists. Personal information about passengers could additionally include that from immigration, law enforcement, and customs. Having such history information stored in the system, or in constituent subsystems, enables SQL based implementation of a temporal pattern rule such as R1. We call such an implementation *off-line* because it relies on storing and querying historical information. An off-line solution induces the following three impact consequences.

1. Temporal historical information is stored within the system (e.g., within CAPPSII and/or its constituent subsystems).
2. Temporal and non-temporal pattern detection is initiated by the security-related query, querying the constituent resources at will. We regard this as *high-impact* solution because a temporal query is initiated from outside the original scope of the queried system thereby impacting the performance of the queried system. For example, an immigration subsystem of CAPPS II being impacted by repeated external queries from CAPPS II which sooner or later will degrade the INS system performance. Performance degradation will occur because of the actual query processing forced on the subsystem and because of the fact that with time, temporal queries can query monotonically increasing datasets of historical data.

In addition to performance issues, CAPPS II and its constituent subsystems need to agree on a shared data representation for merging query results from multiple subsystems (e.g., immigration and law enforcement query results).

This paper is concerned with *low-impact*, *online* temporal pattern detection. It uses REM to detect temporal patterns without using historical data (i.e., it is *online*), and without querying the underlying application (i.e., it is *low-impact*). The only communicated information it requires from underlying applications (e.g., CAPPS II constituent subsystems) are Boolean messages for basic propositions such as *deposit of more than \$1000 was made to account of SSN=222 11 2222*.

While pattern rule R1 described earlier is programmable within the suggested CAPPS II framework, R2 requires an

extension that includes banking information. Such an extension however will not lend itself to off-line temporal pattern detection methods for the following reasons:

1. The banking data systems only store historical/temporal information for a limited duration (e.g., 3 months). The industry is unlikely to make any significant change to this policy.
2. Banking data systems are not likely to permit high-impact, CAPPS II initiated queries because of the performance consequences discussed earlier as well as their own security requirement to be in full control over the content of any such query.

In contrast, a REM temporal pattern detection method, being online and low-impact, can be used in tandem with CAPPSII, while supporting extensions that support rules such as R2.

3 The Temporal Rover and DBRover

The Temporal Rover [6] is a code generator whose input is a Java, C, C++, or HDL source code program where LTL/MTL assertions are embedded as source code comments. The Temporal Rover parser converts this program file into a new file, identical to the original file except for the assertions that are now implemented in source code. The following example contains an embedded MTL assertion for a traffic light controller written using the Temporal Rover syntax asserting that *for 100 milliseconds, whenever light is red, camera should be on*:

```
void tlc(int Color_Main, boolean CameraOn) {
... /* Traffic Light Controller functionality */
/* TRBegin
TRClock{C1=getTimeInMillis()} // get time from the OS
TRAssert{ Always({Color_Main == RED} Implies
            Eventually_C1<1000_{CameraOn==1})
} =>
            // Customizable user actions
            {printf("SUCCESS\n");
             printf("FAIL\n");
             printf("DONE!\n");}
TREnd */
} /* End of tlc */
```

The Temporal Rover generates code that replaces the embedded LTL/MTL assertion with real C, C++, Java, or HDL code which executes in process (i.e., as part of the

- [6] D. Drusinsky, The Temporal Rover and ATG Rover. Proc. Spin2000 Workshop, Springer Lecture Notes in Computer Science, 1885, pp. 323-329. (((YEAR?)))
- [7] D. Drusinsky, Formal Specs Can Handle Exceptions, CMP Embedded Developers Journal, Nov. 2001, pp., 10-14.