# A Constraint Optimisation Model for Analysis of Telecommunication Protocol Logs

Olga Grinchtein[1(✉)], Mats Carlsson[2], and Justin Pearson[3]

[1] Ericsson AB, Stockholm, Sweden
olga.grinchtein@ericsson.com
[2] SICS, Stockholm, Sweden
Mats.Carlsson@sics.se
[3] Uppsala University, Uppsala, Sweden
justin.pearson@it.uu.se

**Abstract.** Testing a telecommunication protocol often requires protocol log analysis. A protocol log is a sequence of messages with timestamps. Protocol log analysis involves checking that the content of messages and timestamps are correct with respect to the protocol specification. We model a protocol specification using constraint programming (MiniZinc), and we present an approach where a constraint solver is used to perform protocol log analysis. Our case study is the Public Warning System service, which is a part of the Long Term Evolution (LTE) 4G standard. We were able to analyse logs containing more than 3000 messages with more than 4000 errors.

**Keywords:** Telecommunication protocol · Testing · Constraint programming

## 1 Introduction

In this paper we investigate the use of constraint programming to implement a part of a test harness for equipment involved in the Long Term Evolution (LTE) 4G standard [1,2], in particular the broadcast of public warning messages [3]. The protocol specification includes a number of messages with complex timing requirements between them. The contribution of the paper is a new approach to analyse the correctness of protocol logs. The main novelty is that we use constraint programming [4] to directly model the protocol and use a constraint solver as a test harness in order to find incorrect behavior in logs. Some results of this paper appeared in the workshop paper [5] presented as work in progress.

In this work, we model a part of the protocol directly in the MiniZinc [6] language (see Section 2). This approach requires a script that reads a protocol log that is a plain text, creates arrays of MiniZinc parameters, and assigns values to the parameters according to the information provided in the log. We also have variables that represent correct timestamps of some messages. There are parameters such as delays of messages for which we know only boundary values,


J.C. Blanchette and N. Kosmatov (Eds.): TAP 2015, LNCS 9154, pp. 137–154, 2015.
DOI: 10.1007/978-3-319-21215-9_9

which adds complexity to the model. The complexity of the model also depends on the number of messages, and we use a technique to partition timestamps of messages into classes. Protocol log analysis in this case is an optimisation problem. We use a constraint solver to find the optimal solution minimising the number of unsatisfied constraints.

The rest of this paper is structured as follows: in Section 2 we give a very brief overview of constraint programming and MiniZinc; in Section 3 we explain main steps of the approach by an example; in Section 4 we give the necessary telecommunication background to understand the case study; in Section 5 we give in some detail the constraint model that is required to test the protocol logs for correctness; in Section 6 we describe optimisation technique to reduce complexity of the model; and in Section 7 we present experimental results.

## 2 MiniZinc and Constraint Programming

Constraint Programming [4] (CP) is a framework for modelling and solving combinatorial problems including verification and optimisation tasks. A constraint problem is specified as a set of *decision variables* that have to be assigned values so that the given constraints on these variables are satisfied, and optionally so that a given objective function is minimised or maximised. Constraint solving is based on the constructive search for such an assignment. Constraint propagation plays an important role: a constraint is not only a declarative modelling device, but has an associated propagator, which is an algorithm to prune the search space by removing values that cannot participate in a solution to that constraint. The removal can trigger other propagators, and this process continues until a fixpoint is reached, at which time the next assignment choice must be made.

MiniZinc [6] is a constraint modelling language, which has gained popularity recently due to its high expressivity and large number of available solvers that support it. It also contains many useful modelling abstractions such as quantifiers, sets, arrays, and a rich set of global constraints. MiniZinc is compiled into FlatZinc, a constraint solving language which specifies a set of built-in constraints that a constraint solver must support. The compilation process is based on flattening by introducing auxiliary variables, substituting them for nested subexpressions, and selecting the appropriate FlatZinc constraints. Common sub-expression elimination plays an important role as well. All the constraints presented in this paper are shown in a form that is very close to their MiniZinc version. We use `fzn-gecode`, the Gecode FlatZinc back-end.

An application of constraint programming to testing in industry is reported in [7] and [8]. In [9] constraint solving is used to derive test cases that distinguish between a piece of code and a mutation of that piece of code. More recently there has been a lot of work on using recent advances in constraint programming applied to white box testing of Java or C [10,11]. In [12] constraint programming is used to generate protocol logs to test telecommunication test harness.

## 3 Overview of the Approach by an Example

We analyse protocol logs that consists of a sequence of messages with timestamps. An abstract sequence of protocol messages is shown in Figure 1. This is not a real log, but we use it to illustrate the approach. The radio base station transmits three messages `M1`, `M2` and `M3` to the mobile phone. Message `M1` does not contain any parameters. Message `M2` contains the parameter $y$ and message `M3` contains the parameters $z_1$ and $z_2$. The first message `M1` the mobile phone reads with some delay, and we introduce decision variable *delay*, which is between 0 and 100 milliseconds.

| | |
|---|---|
| $10:00:00.000$ `M1`$\{\}$ | $10:00:01.600$ `M3`$\{z_1 = 1, z_2 =$ `aaba`$\}$ |
| $10:00:00.080$ `M2`$\{y = 80\}$ | $10:00:01.920$ `M3`$\{z_1 = 2, z_2 =$ `abab`$\}$ |
| $10:00:00.400$ `M3`$\{z_1 = 1, z_2 =$ `aaba`$\}$ | $10:00:02.900$ `M1`$\{\}$ |
| $10:00:00.720$ `M3`$\{z_1 = 2, z_2 =$ `abab`$\}$ | $10:00:03.120$ `M3`$\{z_1 = 1, z_2 =$ `aaba`$\}$ |
| $10:00:01.040$ `M3`$\{z_1 = 4, z_2 =$ `aaaa`$\}$ | $10:00:03.440$ `M3`$\{z_1 = 2, z_2 =$ `abab`$\}$ |
| $10:00:01.450$ `M1`$\{\}$ | $10:00:04.350$ `M1`$\{\}$ |
| $10:00:01.580$ `M2`$\{y = 320\}$ | |

**Fig. 1.** Sequence of messages in a log

We introduce three arrays of parameters `M1Time`, `M2Time` and `M3Time` that represent timestamps in milliseconds since the beginning of the log of corresponding messages in the log. In this example they will have the values

$$\begin{aligned}
&\texttt{M1Time} = [0, 1450, 2900, 4350]\\
&\texttt{M2Time} = [80, 1580]\\
&\texttt{M3Time} = [400, 720, 1040, 1600, 1920, 3120, 3440]
\end{aligned}$$

In the example the parameter $y$ can take two values and parameters $z_1$ and $z_2$ can take four values. We introduce three arrays of parameters `M2y`, `M3z1` and `M3z2` which represent content of the messages in the log. In this example they will have the values

$$\begin{aligned}
&\texttt{M2y} = [1, 2]\\
&\texttt{M3z1} = [1, 2, 4, 1, 2, 1, 2]\\
&\texttt{M3z2} = [1, 2, 4, 1, 2, 1, 2]
\end{aligned}$$

In the example there are two cases in the transmission of messages `M3` by the radio base station. After each message `M1` the radio base station transmits several messages `M3`. After a `M1` message, which is transmitted within 1500 milliseconds, the `M3` messages follow with $z_1$ equal to $1, 2, 3$ and 4. After messages `M1`, which are transmitted after 1500 milliseconds, the messages `M3` follow with $z_1$ equal to 1 and 2. Between the two messages `M1` should be the `M3` messages with all possible values of the parameter $z_1$. This can be captured with the constraint

$$
\begin{aligned}
&(\forall 1 \leq i \leq 3) \\
&\quad ((\texttt{M1Time}_i < 1500 - \mathit{delay} \wedge \\
&\quad (\forall 1 \leq j \leq 4)(\exists 1 \leq k \leq 7)\texttt{M1Time}_i < \texttt{M3Time}_k < \texttt{M1Time}_{i+1} \wedge \texttt{M3z1}_k = j) \\
&\quad \vee \\
&\quad (\texttt{M1Time}_i \geq 1500 - \mathit{delay} \wedge \\
&\quad (\forall 1 \leq j \leq 2)(\exists 1 \leq k \leq 7)\texttt{M1Time}_i < \texttt{M3Time}_k < \texttt{M1Time}_{i+1} \wedge \texttt{M3z1}_k = j))
\end{aligned}
$$

where two disjuncts in the constraint represent two cases of transmission of messages M3. If disjuntcs are not satisfiable, then we have errors in the log. We introduce the Boolean decision variable $\mathit{M3contentinc}_i$ equal to 1 indicates an error in the log and rewrite the constraint as

$$
\begin{aligned}
&(\forall 1 \leq i \leq 3) \\
&\quad ((\texttt{M1Time}_i < 1500 - \mathit{delay} \wedge \\
&\quad (\forall 1 \leq j \leq 4)(\exists 1 \leq k \leq 7)\texttt{M1Time}_i < \texttt{M3Time}_k < \texttt{M1Time}_{i+1} \wedge \texttt{M3z1}_k = j) \\
&\quad \vee \\
&\quad (\texttt{M1Time}_i \geq 1500 - \mathit{delay} \wedge \\
&\quad (\forall 1 \leq j \leq 2)(\exists 1 \leq k \leq 7)\texttt{M1Time}_i < \texttt{M3Time}_k < \texttt{M1Time}_{i+1} \wedge \texttt{M3z1}_k = j)) \\
&\leftrightarrow \mathit{M3contentinc}_i = 0
\end{aligned}
$$

After defining constraints we can use constraint solver to find solution by minimising sum of Boolean decision variables in the array *M3contentinc*. In the example from Figure 1, regardless of the value of *delay*, we have $\mathit{M3contentinc}_1 = 1$, since there is no message M3 with the parameter $z_1 = 3$ between first and second messages M1. However, if $\mathit{delay} = 20$, then $\texttt{M1Time}_2 = 1450 < 1500 - \mathit{delay}$ and $\mathit{M3contentinc}_2 = 1$, since there are no messages M3 with $z_1$ equal to 3 and 4 between second and third messages M1. Thus, the minimum number of errors is one, while *delay* is between 50 and 100. To analyse real protocol logs, we also need to define constraints on timestamps and content of messages, but they are more complex. We present such a constraint model in Section 5.

## 4 Public Warning System for LTE

In our case study we use a constraint solver to test a part of Public Warning System (PWS). The Public Warning System is a technology that broadcasts Warning Notifications to multiple users in case of disasters or other emergencies.

### 4.1 E-UTRAN Architecture

LTE (Long Term Evolution) [1] is the global standard for the fourth generation of mobile networks (4G). Radio Access of LTE is called evolved UMTS Terrestrial Radio Access Network (E-UTRAN)[2]. A E-UTRAN consists of eNodeBs (eNBs), which is just another name for radio base stations. Our setup consists of an eNB, a simulated Mobility Management Entity (MME) that forwards PWS messages to the eNB, and some simulated User Equipment (UE). The functions of these entities are described in more detail below.

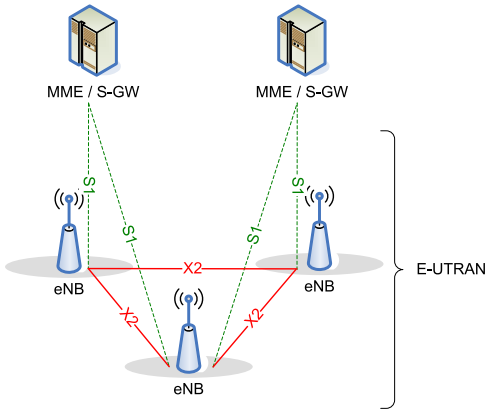



**Fig. 2.** E-UTRAN architecture [2]

An eNB connects to User Equipment via the air interface. The eNBs may be interconnected with each other by means of the X2 interface. The eNBs are also connected by means of the S1 interface to the EPC (Evolved Packet Core), more specifically to the MME (Mobility Management Entity) by means of the S1-MME interface, and to the Serving Gateway (S-GW) by means of the S1-U interface [13]. The MME performs mobility management; security control; distribution of paging messages; ciphering and integrity protection of signaling; and provides support for PWS message transmission. S-GW is responsible for packet routing and forwarding. The functions of eNBs include radio resource management; IP header compression and encryption, selection of MME at UE attachment; routing of user plane data towards S-GW; scheduling and transmission of paging messages and broadcast information; and measurement and reporting configuration for mobility and scheduling [1]. An eNB is responsible for the scheduling and transmission of PWS messages received from MME.

### 4.2 The Earthquake and Tsunami Warning System

The earthquake and Tsunami warning system (ETWS) is a part of PWS that delivers Primary and Secondary Warning Notifications to the UEs within an area where Warning Notifications are broadcast [3]. We show in Figure 3 the network structure of PWS architecture.

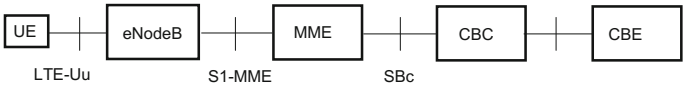



**Fig. 3.** PWS architecture [14]

The Cell broadcast Entity (CBE) can be located at the content provider and sends messages to the Cell Broadcast Center. The Cell Broadcast Center (CBC) is part of EPC and connected to the MME.

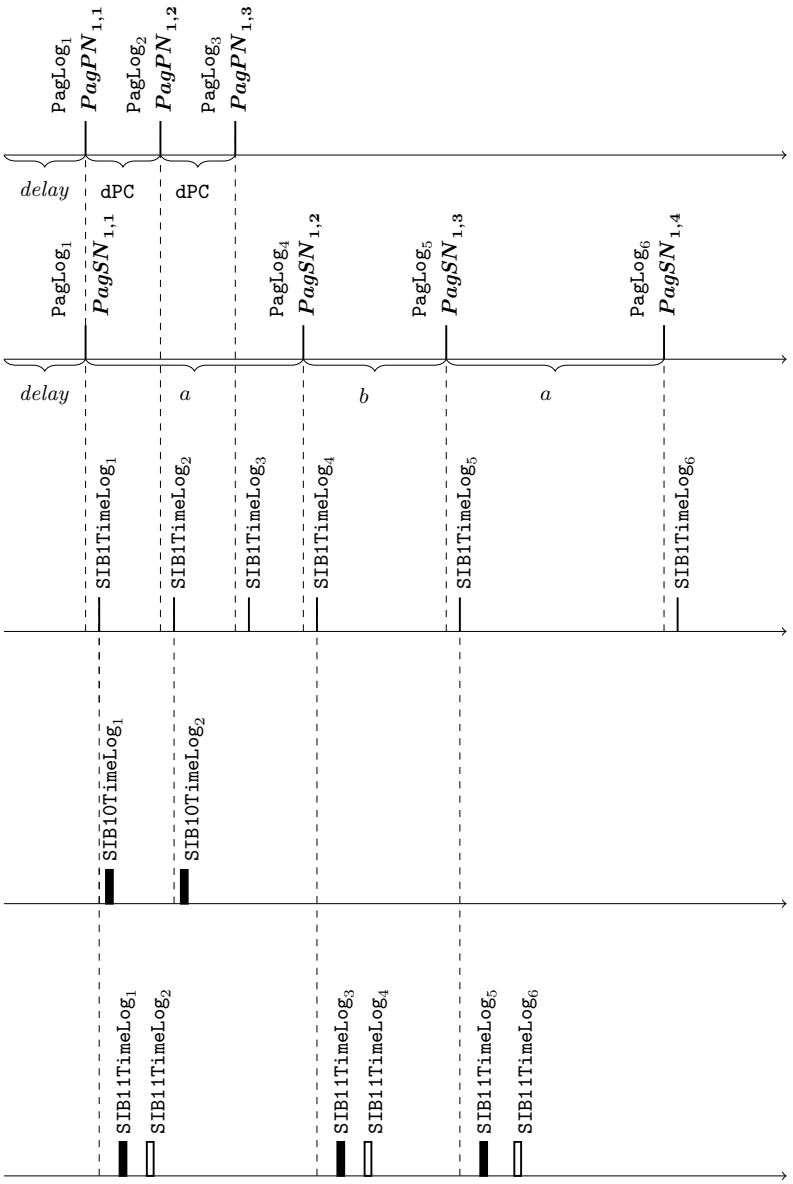



**Fig. 4.** An example of acquiring primary and secondary notification messages by UE

**Table 1.** Parameters and *decision variables* in the models

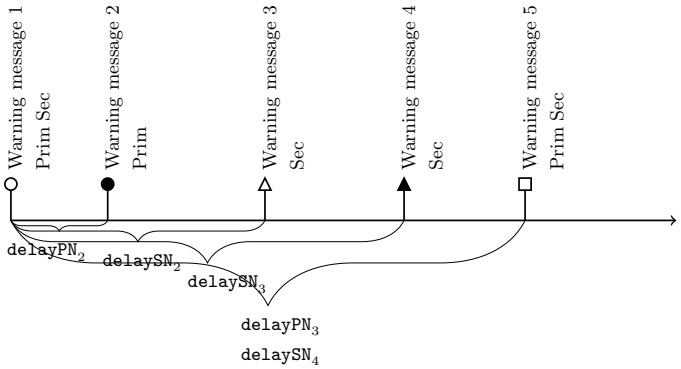
| | |
|---|---|
| *delay* | Time difference between time when eNB starts to transmit primary notification and/or secondary notification and the time when UE reads first paging message. |
| `nPrim` | Number of primary notifications. |
| `delayPN` | An array of timestamps of primary notifications. The size of the array is `nPrim`. |
| `nSec` | Number of secondary notifications. |
| `delaySN` | An array of timestamps of secondary notifications. The size of the array is `nSec`. |
| `dPC` | The length of a paging cycle. |
| `ndPC` | The number of paging cycles, which is configured in eNB. |
| *PagPN* | An array of timestamps of paging messages of primary notification. The size of the array is `ndPC` $\cdot$ `nPrim` |
| `nBR` | An array of `NumberofBroadcastRequested` + 1 of secondary notifications. The size of the array is `nSec`. |
| `nBRmax` | Maximum number in array `nBR`. |
| *PagSN* | An array of timestamps of paging messages of secondary notifications. The size of the array is `nBRmax` $\cdot$ `nSec`. |
| `PagLog` | An array of timestamps of paging messages from the log. The size of the array is `nPagLog`. |
| `rPer` | An array of lengths of repetition periods. The size of the array is `nSec`. |
| `SIB1TimeLog` | An array of timestamps of `SIB1` messages from the log. The size of the array is `nSIB1Log`. |
| `SIB1TypeLog` | An array of values from 0 to 3 that indicate whether `SIB1` messages contain schedulingInfoList for `SIB10` and/or `SIB11` messages. The size of the array is `nSIB1Log`. |
| `SIB10TimeLog` | An array of timestamps of System Information messages with `SIB10` from the log. The size of the array is `nSIB10Log` |
| `SIB11TimeLog` | An array of timestamps of System Information messages with `SIB11` from the log. The size of the array is `nSIB11Log`. |
| `siPerSIB10` | Periodicity of `SIB10`. |
| `siPerSIB11` | Periodicity of `SIB11`. |
| `nSeg` | An array of number of segments in secondary notifications. |

The CBE sends emergency information to the CBC. The CBC identifies which MMEs need to be contacted and sends a Write-Replace Warning Request message containing the warning message to be broadcast to the MMEs. The MME sends a Write-Replace Warning Confirm message that indicates to the CBC that the MME has started to distribute the warning message to eNBs. The MME forwards Write-Replace Warning Request to eNBs in the delivery area. The eNB determines the cells in which the message is to be broadcast based on information received from MME [14]. If a Warning Type IE (information element) is included in a Write-Replace Warning Request message, then the eNB broadcasts a Primary Notification. If Warning Message Contents IE is included in a Write-Replace Warning Request message, then the eNB schedules a broadcast of the warning message according to the value of Repetition Period IE (`rPer`) and Number of Broadcasts Requested IE (`NumberofBroadcastRequested`) [15]. To inform a UE about the presence of an ETWS primary notification and/or ETWS secondary notification, a paging message is used. A UE attempts to read paging messages at least once every defaultPagingCycle (`dPC`). If a UE receives a Paging message including an ETWS-indication, then it starts receiving ETWS primary notification or ETWS secondary notification according to scheduling-InfoList contained in SystemInformationBlockType1 (`SIB1`). ETWS primary notification is contained in SystemInformationBlockType10 (`SIB10`) and ETWS secondary notification is contained in SystemInformationBlockType11 (`SIB11`). The messages `SIB10` and `SIB11` are transmitted in System Information (SI) messages with different periodicity. If a secondary notification contains a large message, then it is divided into several segments, which are transmitted in System Information messages.

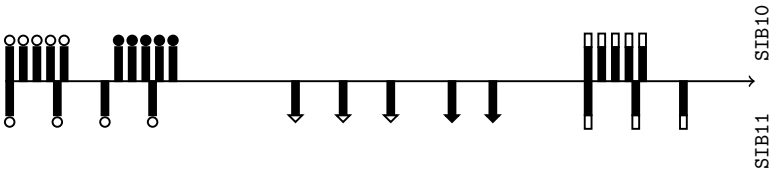
In Table 1 we present a description of some parameters that are constants and *decision variables* used in models. Parameters, which represent the content of `SIB10` and `SIB11` messages, are omitted. In Figure 4 we show an example of a correct reception of paging messages, `SIB1`, `SIB10` and `SIB11` messages of the first warning message by UE, where $a = (\lfloor \mathtt{rPer}_1/\mathtt{dPC} \rfloor + 1) \cdot \mathtt{dPC}$, $\lfloor \mathtt{rPer}_1/\mathtt{dPC} \rfloor$ is the number of whole paging cycles during a repetition period, $b = \lfloor \mathtt{rPer}_1/\mathtt{dPC} \rfloor \cdot \mathtt{dPC}$, $\mathtt{ndPC} = 3, \mathtt{nBR}_1 = 4$ and $\mathtt{nSeg}_1 = 2$. Horizontal lines in Figure 4 represent timelines. The first timeline is used to represent timestamps of paging messages of primary notification, and the second timeline is used to represent timestamps of paging messages of secondary notification. Vertical lines in first and second timelines represent timestamps of paging messages; vertical lines in the third timeline represent timestamps of `SIB1` messages; rectangles in the fourth timeline represent timestamps of `SIB10` messages; and rectangles in the fifth timeline represent timestamps of `SIB11` messages. Unfilled and filled rectangles in the fifth timeline indicate two different message segments.

### 4.3 Replacement of Warning Messages

If a warning message is being broadcast in a certain area and the eNB receives a Write-Replace Warning Request message with an identity which is different from




(a) Transmission of Write-Replace Warning Request messages by MME to eNB. Different shapes on the top of the vertical lines represent different warning messages.




(b) Transmission of primary and secondary notifications for each warning message by eNB to UE. A shape on top (bottom) of a rectangle shows to which warning message SIB10 (SIB11) message corresponds.

**Fig. 5.** Replacement of Write-Replace Warning Request messages

the warning messages being broadcast, the eNB schedules the received warning message for broadcast for that area. Figure 5 illustrates the replacement of warning messages. Figure 5(a) shows timestamps of five Write-Replace Warning request messages (Warning message 1-5) which contain the content of primary notifications (Prim) and/or secondary notifications (Sec). For example Warning message 1 contains the content of primary and secondary notifications and Warning message 2 contains the content of primary notification. The horizontal line is the timeline and vertical lines represent timestamps of warning messages. Warning message 2 is received by eNB $\texttt{delayPN}_2$ milliseconds after Warning message 1 was received. Warning message 5 is received $\texttt{delayPN}_3 = \texttt{delaySN}_4$ milliseconds after Warning message 1 is received. In Figure 5(b) vertical rectangles represent timestamps of `SIB10` and `SIB11` messages which eNB transmit to UE. It can happen that `SIB10` (`SIB11`) messages from a previous warning message continues to transmit after the new warning message is received, if the new message does not contain a content of primary (secondary) notification. For example, when Warning message 2 is received, eNB continues transmit `SIB11` messages from Warning message 1 and start to transmit new `SIB10` messages.

## 5 Modelling of ETWS Notification Acquisition by UE

Our goal is to analyse a UE protocol log that contains paging messages, SIB1, SIB10, and SIB11 messages. We defined a Model that consists of constraints on timestamps and the content of these messages. We divided the Model into three submodels, each of them checks for different information in a protocol log. Some constraints appear in all submodels. The division into submodels was made in order to reduce complexity of the overall model and is useful for a quick check of partial information in case when Model takes a long time to solve. The submodels are

- PagingModel that checks that the log contains all required paging messages, and that the number and timestamps of paging messages are correct
- SIB1Model that checks that the log contains all required SIB1 messages, and that the schedulingInfoList is correct
- PrimSecModel checks that the log contains SIB10 messages with correct timestamps, content and identity numbers. It also checks that SIB11 messages have correct timestamps, content, segments and identity numbers.

The recommended values for dPC, siPerSIB10, siPerSIB11 and rPer satisfy constraints dPC > siPerSIB10, rPer > siPerSIB11 and rPer > dPC, which we assume in all our models. Other values are possible, but would require different testing strategies. We also assume that the first message in the log is a paging message and we assign value 0 to $\texttt{PagLog}_1$. We assign to the array SIB10TimeLog of timestamps of SIB10 messages, and to the array SIB11TimeLog of timestamps of SIB11 messages values, which are time differences between timestamps of the messages in the log and the timestamp of the first paging message in the log. The main ingredient of the model is *delay*, which is an integer decision variable in the model that can be between 0 and dPC. It represents the delay of first paging message. We use binary search as a search strategy for *delay*. We use arrays of Boolean decision variables which are equal to 1 if the corresponding constraints are unsatisfiable. Then we search for a solution that minimises *objective* that is the sum of the Boolean variables.

### 5.1 Delays of Warning Messages as Decision Variables

The arrays delayPN and delaySN represent timestamps of warning messages sent to eNB by a MME. Since these timestamps are constant and some variable delay can occur, we introduce arrays of decision variables *delayPN50* and *delaySN50* which represent extra delay of warning messages. We assume that delays are less than 50 milliseconds.

Simply introducing a decision variable between 0 and 50 as an extra delay will increase complexity of the problem drastically. When there is extra delay some paging messages, SIB1, SIB10 and SIB11 could belong to the previous warning message.Thus, we can set values to extra delay $delayPN50_i$ by calculating distances between timestamps of SIB1, SIB10 and SIB11 messages and timestamp

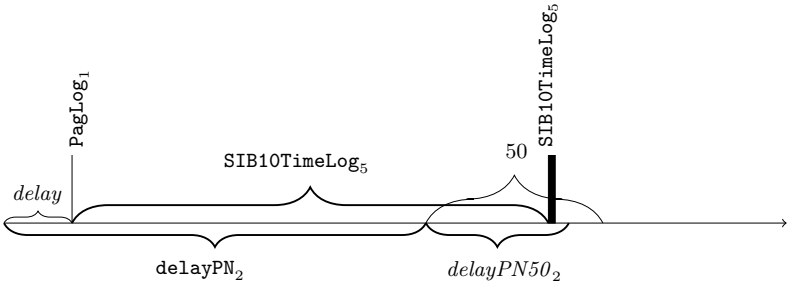



**Fig. 6.** Illustration of $\texttt{setdelayPN50}(2, \texttt{nSIB10Log}, \texttt{SIB10TimeLog})$

of $i$th warning message and choose distance with less than 50 milliseconds as a value for $delayPN50_i$. We constrain $delayPN50$ by expression:

$$
\begin{aligned}
&(\forall 1 \leq i \leq \texttt{nPrim}) \\
&\quad (delayPN50_i = 0 \\
&\quad \vee \\
&\quad (\texttt{dPC} - ((\texttt{delayPN}_i - delay) \mod \texttt{dPC}) < 50 \wedge \\
&\qquad delayPN50_i = \texttt{dPC} - ((\texttt{delayPN}_i - delay) \mod \texttt{dPC}) + 1) \\
&\quad \vee \\
&\quad \texttt{setdelayPN50}(i, \texttt{nSIB1Log}, \texttt{SIB1TimeLog}) \\
&\quad \vee \\
&\quad \texttt{setdelayPN50}(i, \texttt{nSIB10Log}, \texttt{SIB10TimeLog}) \\
&\quad \vee \\
&\quad \texttt{setdelayPN50}(i, \texttt{nSIB11Log}, \texttt{SIB11TimeLog})) \qquad (1)
\end{aligned}
$$

where

$$
\begin{aligned}
&\texttt{setdelayPN50}(i, k, \texttt{TimeLog}) = \\
&(\exists 1 \leq j \leq k) \\
&\qquad (0 \leq \texttt{TimeLog}_j - \texttt{delayPN}_i + delay < 50 \wedge \\
&\qquad\quad delayPN50_i = \texttt{TimeLog}_j - \texttt{delayPN}_i + delay + 1) \qquad (2)
\end{aligned}
$$

Figure 6 shows illustration of $\texttt{setdelayPN50}(2, \texttt{nSIB10Log}, \texttt{SIB10TimeLog})$, where $delayPN50_2 = \texttt{SIB10TimeLog}_5 - \texttt{delayPN}_2 + delay + 1$.

The array $delaySN50$ is defined in a similar way. Then we replace in the model $\texttt{delayPN}_i$ by $\texttt{delayPN}_i + delayPN50_i$ and $\texttt{delaySN}_i$ by $\texttt{delaySN}_i + delaySN50_i$. We have a constraint to guarantee that if $\texttt{delayPN}_i = \texttt{delaySN}_j$ then $delayPN50_i = delaySN50_j$, $1 \leq i \leq \texttt{nPrim}$ and $1 \leq j \leq \texttt{nSec}$ .

### 5.2 Modeling of Timestamps of Paging Messages

The first timeline in Figure 4 shows timestamps of paging messages, which are part of transmission of primary notification and the second timeline shows timestamps of paging messages, which correspond to secondary notification. Since paging messages of primary and secondary notifications look identical in the logs,

we introduce one array PagLog of timestamps of paging messages from the log, which contain timestamps of paging messages of primary and secondary notifications in the order as they appear in the log. However paging messages of primary and secondary notifications have different periodicity and we need to distinguish them in order to check correctness of messages in the log. Periodicity of paging messages of primary notifications is dPC, but the time difference between two consecutive paging messages of secondary notification depends on the repetition period of notification and can take two different values for the same notification as shown in Figure 4. On the other hand, if there are more paging messages in the log than there should be or there are other errors in paging messages in the log, SIB10 and SIB11 messages can still be correct, but we cannot use timestamps of paging messages from the log. Therefore we introduce the array of correct timestamps of paging messages of primary notification $PagPN$ and the array of correct timestamps of secondary notification $PagSN$. We post constraints on these arrays which calculate periodicity of paging messages. These constraints appear in all models.

### 5.3 Description of the PagingModel

We check that all required paging messages of primary and secondary notifications are present in the log. We check that every paging message in the log is a paging message of primary notification or paging message of a secondary notification.

### 5.4 Description of the SIB1Model

We have several constraints to check the timing and content of SIB1 messages. The constraint (3) is an example of constraint, that checks the correctness of messages. In (3) we check that if the SIB1 message is between the first paging message of primary notification and the last paging message of primary notification, then it contains scheduling information for SIB10. The array SIB1TimeLog contains timestamps of SIB1 messages in the log. SIB1TypeLog is array of values from 0 to 3 that indicates whether SIB1 contains schedulingInfoList for SIB10 and/or SIB11. Then we post a constraint

$$
\begin{aligned}
&(\forall 1 \le k \le \texttt{nSIB1Log}) \\
&\quad \big(\big(\big((\exists 1 \le i \le \texttt{nPrim} - 1) \\
&\qquad (\texttt{SIB1TimeLog}_k \ge \texttt{delayPN}_i + delayPN50_i - delay \wedge \\
&\qquad ((PagPN_{i,\texttt{ndPC}} = -1 \wedge \texttt{SIB1TimeLog}_k < \\
&\qquad \texttt{delayPN}_{i+1} + delayPN50_{i+1} - delay) \vee \\
&\qquad (PagPN_{i,\texttt{ndPC}} \neq -1 \wedge \texttt{SIB1TimeLog}_k \le PagPN_{i,\texttt{ndPC}}))\big) \\
&\quad \bigvee \\
&\quad (\texttt{SIB1TimeLog}_k \ge \texttt{delayPN}_{\texttt{nPrim}} + delayPN50_{\texttt{nPrim}} - delay \wedge \\
&\qquad \texttt{SIB1TimeLog}_k \le PagPN_{\texttt{nPrim,ndPC}})\big) \\
&\leftrightarrow (\texttt{SIB1TypeLog}_k = 1 \vee \texttt{SIB1TypeLog}_k = 3)\big) \\
&\leftrightarrow SIB1PrimTypeinc_k = 0
\end{aligned}
\tag{3}
$$

where the Boolean variable $SIB1PrimTypeinc_k$ equal to 1 indicates an error in log. Since we can assign different values to *PagPN* and *PagSN* due to unknown value for *delay*, we use constraints and arrays of Boolean decision variables from `PagingModel` in `SIB1Model`. Minimisation of sum of Boolean decision variables from `PagingModel` helps reduce variations in the values of the timestamps in *PagPN* and *PagSN*.

### 5.5 Description of the `PrimSecModel`

The model `PrimSecModel` checks correctness of timing and content of `SIB10` and `SIB11` messages in the log. For example, we check that notifications have correct identity numbers. We also check the correctness of sequences of `SIB11` segments, and that there are messages every paging cycle and repetition period. As in `SIB1Model` we use constraints and arrays of Boolean decision variables from `PagingModel` also in `PrimSecModel`.

## 6 Partitioning of Timestamps of Messages

If the log is large and contains for example 1000 `SIB10` messages, then we can have many constraints of the form

$$(\forall 1 \leq i \leq \texttt{nPrim})(\exists 1 \leq k \leq 1000).\phi(i,k) \tag{4}$$

$$(\forall 1 \leq k \leq 1000)(\exists 1 \leq i \leq \texttt{nPrim}).\phi'(i,k) \tag{5}$$

and

$$(\forall 1 \leq i \leq \texttt{nPrim})(\forall 1 \leq j \leq \texttt{ndPC})(\exists 1 \leq k \leq 1000).\phi''(i,j,k) \tag{6}$$

where k is index of `SIB10` message in the log, $\texttt{nSIB10Log} = 1000$ and $\phi$, $\phi'$ and $\phi''$ are some constraints. Even with small values for `nPrim` and `ndPC`, MiniZinc cannot process such constraints. However, we can partition messages into classes, where a message belongs to class $i$ if its timestamp is between $\texttt{delayPN}_i - \texttt{dPC}$ and $\texttt{delayPN}_{i+1} + \texttt{dPC}$, where $1 \leq i \leq \texttt{nPrim} - 1$ or greater than $\texttt{delayPN}_i - \texttt{dPC}$ if $i = \texttt{nPrim}$. It can happen that the message belongs to several classes, but it helps to significantly reduce the size of constraint, it does not change the set of solutions, and makes the approach practical. For example, for `SIB10` messages we can have arrays of integers $f^{min}$ and $f^{max}$ such that

$$f_i^{min} = \min_{1 \leq k \leq \texttt{nSIB10Log}} \{k | \texttt{delayPN}_i - \texttt{dPC} \leq \texttt{SIB10TimeLog}_k \leq \texttt{delayPN}_{i+1} + \texttt{dPC}\},$$

where $1 \leq i \leq \texttt{nPrim} - 1$ and

$$f_{\texttt{nPrim}}^{min} = \min_{1 \leq k \leq \texttt{nSIB10Log}} \{k | \texttt{SIB10TimeLog}_k \geq \texttt{delayPN}_{\texttt{nPrim}} - \texttt{dPC}\}$$

$$f_i^{max} = \max_{1 \leq k \leq \texttt{nSIB10Log}} \{k | \texttt{delayPN}_i - \texttt{dPC} \leq \texttt{SIB10TimeLog}_k \leq \texttt{delayPN}_{i+1} + \texttt{dPC}\},$$

where $1 \leq i \leq \texttt{nPrim} - 1$ and $f_{\texttt{nPrim}}^{max} = \texttt{nSIB10Log}$.

Since `delayPN`, `SIB10TimeLog` and `dPC` are constants, we can easily calculate $f^{min}$ and $f^{max}$ and rewrite (4) as

$$(\forall 1 \leq i \leq \texttt{nPrim})(\exists f_i^{min} \leq k \leq f_i^{max}).\phi(i,k) \tag{7}$$

Similary, we can define arrays of integers $g^{min}$ and $g^{max}$ and more complex arrays of integers $\gamma_{i,j}^{min}$ and $\gamma_{i,j}^{max}$ and rewrite (5) as

$$(\forall 1 \leq k \leq 1000)(\exists g_k^{min} \leq i \leq g_k^{max}).\phi'(i,k) \tag{8}$$

and (6) as

$$(\forall 1 \leq i \leq \texttt{nPrim})(\forall 1 \leq j \leq \texttt{ndPC})(\exists \gamma_{i,j}^{min} \leq k \leq \gamma_{i,j}^{max}).\phi''(i,j,k) \tag{9}$$

Similar calculations have been done for paging messages, `SIB1` and `SIB11` messages.

## 7 Experiments

We used our constraint model to find errors in real logs, and generated logs of different size with injected errors. The experiments were done on the computer equipped with 8GB RAM and an Intel Core i5-3210M processor (2.50GHz).

### 7.1 Analysis of Real Logs

We analysed nine real logs, which were documented and were in an internal archive of Ericsson. Each log was captured in a UE simulator after sending two Write-Replace Warning Request messages from a MME simulator to an eNB. The logs have different structures, and represent all possible combinations of primary and secondary notifications in case of two warning messages. For example, the first warning message contains primary and secondary notifications, and the second warning message contains primary notifications; or another example, the first warning message contains secondary notifications, and the second warning message contains secondary notifications. Nine combinations are possible in case of two warning messages. The size of logs is between 138KB and 578KB. The number of paging messages is between 8 and 26, the number of `SIB1` messages is between 8 and 26, the number of `SIB10` messages is between 0 and 75, and the number of `SIB11` messages is between 0 and 24.

The running time for the `Model` was a few seconds for each log, and the found *objective* was between 0 and 70. The optimisation as presented in Section 6 was not needed. Eight logs have the property that the Boolean decision variables, which have value 1 in optimal solution, have the same value in all other solutions. Thus, the found errors are present in all solutions for different values of *delay*. This was checked by adding to the model a constraint with negated conjunction of values of non-zero Boolean decision variables of optimal solution, then MiniZinc reported that the model is unsatisfiable. One log does not have such property, but it has the property that there is only one solution with the value of *objective* being optimal, and the difference between other solutions and optimal solution is at least 6 errors.

**Table 2.** Analysis of correct generated logs

| | | log1 | log2 | log3 | log4 | log5 |
|---|---|---|---|---|---|---|
| | `nPrim` | 30 | 25 | 20 | 15 | 10 |
| | `nSec` | 30 | 25 | 20 | 15 | 10 |
| `PagingModel` | `nPagLog` | 625 | 510 | 419 | 307 | 218 |
| | time | 0:03:59 | 0:02:39 | 0:00:22 | 0:00:11 | 0:00:07 |
| | time,gecode | 0:00:04 | 0:00:02 | 0:00:02 | 0:00:01 | 0:00:01 |
| | *objective* | 0 | 0 | 0 | 0 | 0 |
| `SIB1Model` | `nSIB1Log` | 625 | 510 | 419 | 307 | 218 |
| | time | 0:04:21 | 0:02:56 | 0:00:42 | 0:00:21 | 0:00:11 |
| | time,gecode | 0:00:04 | 0:00:03 | 0:00:02 | 0:00:01 | 0:00:01 |
| | *objective* | 0 | 0 | 0 | 0 | 0 |
| `PrimSecModel` | `nSIB10Log` | 3062 | 2628 | 2091 | 1435 | 1057 |
| | `nSIB11Log` | 2753 | 2209 | 1500 | 1052 | 867 |
| | time | 0:16:47 | 0:12:46 | 0:03:49 | 0:02:13 | 0:01:27 |
| | time,gecode | 0:00:05 | 0:00:03 | 0:00:02 | 0:00:02 | 0:00:01 |
| | *objective* | 0 | 0 | 0 | 0 | 0 |
| `Model` | time | 0:20:59 | 0:14:31 | 0:04:46 | 0:02:45 | 0:01:43 |
| | time,gecode | 0:00:05 | 0:00:04 | 0:00:03 | 0:00:02 | 0:00:02 |
| | *objective* | 0 | 0 | 0 | 0 | 0 |

### 7.2 Analysis of Generated Logs

We generated logs, with and without errors, in order to understand how the model scales. The generation of protocol logs was described in [12] where SICStus Prolog [16] was used as constraint solver. We extended the approach and used Gecode[17] and C++ for log generation. Note that this was not a pure constraint model, but it included some imperative pre and post processing steps.

Table 2 shows results of the analysis of generated correct logs, where the found *objective* is 0. The optimization presented in Section 6 was used. The total time includes translation of models to FlatZinc using `mzn-gecode` and execution time of Gecode on the compiled FlatZinc using `fzn-gecode`. We can see that a very large log with 625 paging messages, 625 SIB1 messages, 625 `SIB10` messages, 3062 `SIB10` messages, and 2753 `SIB11` messages requires 21 minutes to compile to FlatZinc. While `fzn-gecode` found the solution in a few seconds. A log that is three times smaller containing 10 primary and 10 secondary notifications requires 2 minutes to compile to FlatZinc and only 2 seconds to find solution.

In Table 3 we present the results of the analysis of generated logs where errors were introduced. All logs in Table 3 are incorrect versions of Log6. The Log7 to Log15 were generated by changing the timestamp of messages (t), removing messages (r), adding extra messages (a) and changing the content of messages (c). In Log7 and Log8 some paging messages are not correct. In Log9 and Log10 some `SIB1` messages are not correct. In Log11 and Log12 some `SIB10` messages

**Table 3.** Analysis of generated logs with injected errors

| | | log6 | log7 | log8 | log9 | log10 | log11 | log12 | log13 | log14 | log15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | `nPrim` | 20 | | | | | | | | | |
| | `nSec` | 20 | | | | | | | | | |
| | errors | | r,a | t | r,c | t,c | r | t,c | r,t,c | r,a,t,c | r,a,t,c |
| | in messages | | paging | paging | `SIB1` | `SIB1` | `SIB10` | `SIB10` | `SIB11` | all | all |
| `PagingModel` | `nPagLog` | 374 | 174 | 374 | 374 | 374 | 374 | 374 | 374 | 374 | 398 |
| | time | 0:00:23 | 0:00:27 | 0:01:23 | | | | | | 0:00:58 | 0:23:15 |
| | time,gecode | 0:00:02 | 0:00:08 | 0:01:02 | | | | | | 0:00:39 | 0:22:56 |
| | *objective* | 0 | 413 | 155 | | | | | | 192 | 789 |
| `SIB1Model` | `nSIB1Log` | 374 | 374 | 374 | 75 | 374 | 374 | 374 | 374 | 383 | 473 |
| | time | 0:00:39 | 0:00:41 | 0:01:30 | 0:01:07 | 0:04:23 | | | | 0:01:22 | 0:32:16 |
| | time,gecode | 0:00:02 | 0:00:06 | 0:00:50 | 0:00:38 | 0:03:44 | | | | 0:00:45 | 0:31:33 |
| | *objective* | 0 | 413 | 155 | 684 | 395 | | | | 304 | 1386 |
| `PrimSecModel` | `nSIB10Log` | 1875 | 1916 | 1994 | 1871 | 1858 | 406 | 1922 | 1888 | 1683 | 2034 |
| | `nSIB11Log` | 1430 | 1429 | 1429 | 1429 | 1429 | 1430 | 1429 | 1143 | 1295 | 1242 |
| | time | 0:03:38 | 0:03:45 | 0:04:49 | | | 0:03:28 | 0:07:54 | 0:04:11 | 0:06:24 | 1:25:55 |
| | time,gecode | 0:00:03 | 0:00:08 | 0:00:58 | | | 0:01:00 | 0:04:11 | 0:00:59 | 0:02:57 | 1:22:21 |
| | *objective* | 0 | 413 | 155 | | | 220 | 3611 | 2061 | 4267 | 6789 |
| `Model` | time | 0:05:27 | 0:04:48 | 0:06:04 | 0:04:42 | 0:08:38 | 0:04:22 | 0:08:34 | 0:05:14 | 0:07:12 | 1:40:13 |
| | time,gecode | 0:00:03 | 0:00:09 | 0:01:04 | 0:00:45 | 0:03:40 | 0:01:00 | 0:03:46 | 0:01:01 | 0:02:46 | 1:35:18 |
| | *objective* | 0 | 413 | 155 | 684 | 395 | 220 | 3611 | 2061 | 4379 | 7396 |

are not correct, and in Log13 some `SIB11` messages are not correct. In Log14 and Log15 there are errors in all types of messages. It took 7 minutes to find solution for `Model` of Log14, which consists of 3735 messages and 4379 errors. This is still a good result, since in a real environment it would require more than 30 minutes to collect a log of the same order of magnitude as Log14.

However, as the analysis of Log15 shows, when there are significantly incorrect timestamps of messages the solving time can increase significantly. It appears that incorrect timestamps of messages are harder for the solver to handle, since they are appear in constraints more often.

## 8 Learning Delay

Constraint programming can be used to analyse small logs if some parameter is unknown. This is often the case with logs in an archive. There was a log in the archive, with two warning notifications, that was not well documented. There was no information about the delay of second warning message. The warning messages consisted of primary and secondary notifications, that is $\texttt{delayPN}_2 = \texttt{delaySN}_2$. We estimated that $\texttt{delayPN}_2$ must be less than 80 seconds. The log was $1,5$ MB and there were 23 paging messages, 183 `SIB10` messages and 40 `SIB11`

messages in the log. We used $\texttt{delayPN}_2$ as decision variable in `PagingModel`. It took one second for constraint solver to find a solution with the *objective* being 0. That is it found a value of $\texttt{delayPN}_2$ such that all paging messages in log are correct. We used the generated value of $\texttt{delayPN}_2$ in `Model` and got a solution after 9 seconds with the *objective* strictly greater than 0. We also used $\texttt{delayPN}_2$ as a decision variable in `Model` and got a solution after 6 minutes with the same objective value.

## 9 Conclusion

There are a number of advantages of using MiniZinc and constraint programming: it was easy to translate the required parts of the telecommunication specification [3] directly into MiniZinc; these MiniZinc specifications are automatically translated into a constraint program that can be used to test protocol logs for correctness directly; the MiniZinc specification is a declarative specification of the protocol behaviour rather than the procedural implementation that is usually used for implementation of the checker; and finally adding more functionality to the MiniZinc implementation is done by simply adding more constraints.

Constraint solvers can easily handle complex requirements on timestamps. We used the MiniZinc model to analyse real logs and also larger generated logs with a lot of errors, which shows its usability in practice. The constraint solver was able to handle big domains of parameters, and we do not need to reduce or scale the domains. Protocol log analysis with constraint programming can be a part of test automation and can be useful for functional testing as well as in regression testing. Further, we believe that the protocol itself has independent interest as a useful case study for other formal modelling approaches. As a future work we plan to apply the approach to other case studies and protocols.

**Acknowledgments.** The authors would like to thank Noric Couderc for fruitful discussions on protocol log generation with Gecode. The first author is supported by VINNMER Program 2011-03229 funded by Swedish Governmental Agency for Innovation Systems. The third author is supported by grant 2012-4908 of the Swedish Research Council(VR).

## References

1. Chadchan, S., Akki, C.: 3GPP LTE/SAE: An overview. International Journal of Computer and Electrical Engineering **2**(5), 806–814 (2010)
2. 3GPP: Evolved universal terrestrial radio access (e-utra) and evolved universal terrestrial radio access network (e-utran); overall description; stage 2. TS 36.300, 3rd Generation Partnership Project (3GPP)
3. 3GPP: Public warning system (PWS) requirements. TS 22.268, 3rd Generation Partnership Project (3GPP)
4. Rossi, F., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier (2006)

5. Carlsson, M., Grinchtein, O., Pearson, J.: Protocol log analysis with constraint programming (work in progress). In: Proceedings of the 12th International Workshop on Satisfiability Modulo Theories, SMT, pp. 17–26 (2014)
6. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.R.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
7. Mossige, M., Gotlieb, A., Meling, H.: Testing robotized paint system using constraint programming: an industrial case study. In: Merayo, M.G., de Oca, E.M. (eds.) ICTSS 2014. LNCS, vol. 8763, pp. 145–160. Springer, Heidelberg (2014)
8. Mossige, M., Gotlieb, A., Meling, H.: Using CP in automatic test generation for ABB robotics' paint control system. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 25–41. Springer, Heidelberg (2014)
9. DeMilli, R., Offutt, A.J.: Constraint-based automatic test data generation. IEEE Transactions on Software Engineering **17**(9), 900–910 (1991)
10. Williams, N., Marre, B., Mouy, P., Roger, M.: PathCrawler: automatic generation of path tests by combining static and dynamic analysis. In: Dal Cin, M., Kaâniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 281–292. Springer, Heidelberg (2005)
11. Carlier, M., Dubois, C., Gotlieb, A.: FocalTest: a constraint programming approach for property-based testing. In: Cordeiro, J., Virvou, M., Shishkov, B. (eds.) ICSOFT 2010. CCIS, vol. 170, pp. 140–155. Springer, Heidelberg (2013)
12. Balck, K., Grinchtein, O., Pearson, J.: Model-based protocol log generation for testing a telecommunication test harness using clp. In: DATE (2014)
13. 3GPP: General packet radio service (GPRS) enhancements for evolved universal terrestrial radio access network (E-UTRAN) access. TS 23.401, 3rd Generation Partnership Project (3GPP)
14. 3GPP: Technical realization of cell broadcast service (CBS). TS 23.041, 3rd Generation Partnership Project (3GPP)
15. 3GPP: Evolved universal terrestrial radio access (E-UTRA); S1 application protocol (S1AP). TS 36.413, 3rd Generation Partnership Project (3GPP)
16. Carlsson, M., Ottosson, G., Carlson, B.: An open-ended finite domain constraint solver. In: Hartel, P.H., Kuchen, H. (eds.) PLILP 1997. LNCS, vol. 1292, pp. 191–206. Springer, Heidelberg (1997)
17. Gecode Team: Gecode: A generic constraint development environment (2006). http://www.gecode.org