Protocol Log Analysis with Constraint Programming (Work in progress)

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

[1] SICS, Stockholm, Sweden
`Mats.Carlsson@sics.se`
[2] Ericsson AB, Stockholm, Sweden
`olga.grinchtein@ericsson.com`
[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 the 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.

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 [8, 2] in particular the broadcast of public warning messages [3]. The protocol includes a number of messages with complex timing requirements between them. The main novelty is that we use constraint programming [11] to directly model the protocol and to implement a test harness directly. Further, we believe that the protocol itself has independent interest as useful case study for other formal modelling approaches.

In our previous work [5] we presented an approach where we were testing an existing test harness written in Java. We generated protocol logs to test the existing Java implementation in order to find errors in the implementation. We created a model of the protocol in constraint programming in SICStus Prolog [7] and used the solutions of the constraint program to generate protocol logs. The model was then modified to produce protocol logs that were nearly correct, that is we injected faults, and these nearly correct logs were used to test if the test harness could spot errors in the protocol logs.

However, another approach can be applied in order to check that protocol log contains correct messages with correct timing: Our new approach is to use constraint solver to analyze logs directly, and hence implement the test harness using a constraint solver.

In this work, we model a part of the protocol directly in the MiniZinc [9] language (see Section 2). This approach requires a script that reads the protocol log, creates arrays of MiniZinc variables, and assigns values to the variables according to the information provided in the log.

There are a number of advantages of using MiniZinc and constraint programming: first it was very 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 was used in the existing Java implementation of the checker; and finally the part of the protocol modelled here already provides more functionality and requires three times less lines in MiniZinc than existing Java code, and adding more functionality to the MiniZinc implementation is simply of adding more 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 give the necessary telecommunication background to understand the case study; and in section 4 we give in some detail the constraint model that is required to test the protocol logs for correctness.

2 MiniZinc and Constraint Programming

Constraint Programming [11] (CP) is a framework for modelling and solving combinatorial problems such as verification and optimization tasks. A constraint problem is specified as a set of 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 to fixpoint, at which time the next assignment choice must be made. A distinguishing feature of CP is the use of global constraints [11, 6]. They capture commonly occurring combinatorial patterns such as constraints on sequences, constraints on order, and constraints on placement of objects and tasks in space and time, to name a few.

MiniZinc [9] is a constraint modelling language, which has gained popularity recently due to its high expressivity and large number of available solvers. The MiniZinc language is a superset of SMT over quantifier-free formulas with linear arithmetic [10]. 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.

3 Public Warning System for LTE

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

3.1 E-UTRAN architecture

LTE (Long Term Evolution) [8] 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.

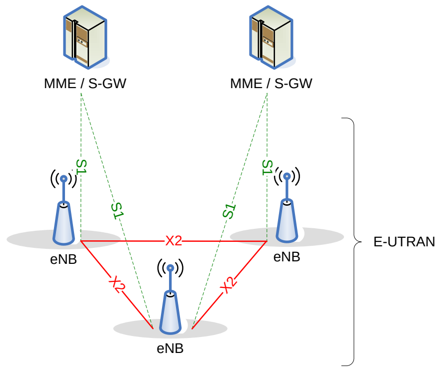

Figure 1: This figure is from 3GPP TS 36.300

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 [2]. 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 [8]. An eNB is responsible for the scheduling and transmission of PWS messages received from MME. 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.

3.2 ETWS

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 2 the network structure of PWS architecture.

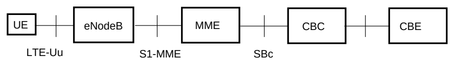

Figure 2: This figure is from 3GPP TS 23.041

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

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 [4]. 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`) [1]. To inform UE about presence of an ETWS primary notification and/or ETWS secondary notification, a paging message is used. UE attempts to read paging at least once every defaultPagingCycle (`dPC`). If UE receives a Paging message including ETWS-indication, then it starts receiving ETWS primary notification or ETWS secondary notification according to schedulingInfoList contained in SystemInformationBlockType1 (`SIB1`). ETWS primary notification is contained in SystemInformationBlockType10 (`SIB10`) and ETWS secondary notification is contained in SystemInformationBlockType11 (`SIB11`). `SIB10` and `SIB11` are transmitted in System Information (SI) messages with different periodicity. If secondary notification contains a large message, then it is divided in several segments, which are transmitted in System Information messages.

4 Modelling of ETWS notifications acquisition by UE

Our models describe how UE acquires ETWS notifications sent by an eNB after receiving one Write-Replace Warning Request message from the MME. In Section 4.1 we give an overview of a model presented in [5]. In Section 4.2 we present a model in MiniZinc that analyzes protocol logs, and we compare MiniZinc model with a model from [5]. In Table 1 we present a description of parameters used in models.

4.1 Model for generation of protocol logs

ETWS requires testing that the paging messages, `SIB1`, `SIB10` and `SIB11` are transmitted correctly by the eNB. These messages appear in a UE protocol log. To test functionality automatically, the test harness initiates transmission of Write-Replace Warning Request messages by the MME simulator; configures the UE simulator and initiate logging; configures the eNB; and captures and reads a UE protocol log. The use of the MiniZinc model simply requires plugging into an existing framework that captures and reads the protocol logs.

It is useful to understand our previous work [5] where the goal was to generate UE protocol logs for ETWS, which consists of sequences of messages with timestamps, where different types of errors are introduced. To do this, we defined a model in SICStus Prolog consisting of constraints on arrays of timestamps and message contents, and based on solutions provided by SICStus Prolog we generated UE protocol logs. The constraints specified ordering constraints between messages; constraints on the number of messages of a certain type and content; and temporal constraints on the timestamps. The constraint that defines time difference between

Table 1: Parameters in the models

`delay`	Time difference between time when eNodeB starts to transmit primary notification and/or secondary notification and the time when UE reads first paging message.
`PagPN`	An array of timestamps of paging messages (also used in the model in [5]), which are transmitted every paging cycle. The size of the array is `ndPC`, which is configured in eNodeB.
`dPC`	The length of a paging cycle.
`PagSN`	An array of timestamps of paging messages (also used in the model in [5]), which are transmitted every repetition period. The size of the array is `nBR` $=$ `NumberofBroadcastRequested` $+ 1$.
`PagLog`	An array of timestamps of paging messages from the log. The size of the array is `nPagLog`.
`rPer`	The length of a repetition period.
`SIB1SIB10Time`	An array of timestamps of `SIB1` messages during paging cycles (used only in the model in [5]).
`SIB1SIB10Type`	An array of values from 0 to 3 that indicate whether `SIB1` messages contain schedulingInfoList for `SIB10` and/or `SIB11` (used only in the model in [5]).
`SIB1SIB11Time`	An array of timestamps of `SIB1` messages during repetition periods (used only in the model in [5]).
`SIB1SIB11Type`	An array of values from 0 to 3 that indicate whether `SIB1` messages contain schedulingInfoList for `SIB10` and/or `SIB11` (used only in the model in [5]).
`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`. The size of the array is `nSIB1Log`.
`SIB10Time`	An array of timestamps of System Information messages with `SIB10` (used only in the model in [5]).
`SIB10TimeLog`	An array of timestamps of System Information messages with `SIB10` from the log.
`SIB11Time`	An array of timestamps of System Information messages with `SIB11` (used only in the model in [5]).
`SIB11TimeLog`	An array of timestamps of System Information messages with `SIB11` from the log. The size of the array is `nSIB11Log`.
siPer	Periodicity of `SIB11`.
`nSeg`	Number of segments in a secondary notification.
`SIB11Segment`	An array of segment numbers of `SIB11` (used only in the model in [5]).
`SIB11SegmentLog`	An array of segment numbers of `SIB11`. The size of the array is `nSIB11Log`.

two consecutive paging messages transmitted every repetition period is

$$\begin{aligned}&(\forall 1 \leq i \leq \texttt{nBR}-1)\\&(\texttt{PagSN}_{i+1}-\texttt{PagSN}_i = \lfloor \texttt{rPer}/\texttt{dPC}\rfloor \cdot \texttt{dPC})\\&\vee\\&(\texttt{PagSN}_{i+1}-\texttt{PagSN}_i = (\lfloor \texttt{rPer}/\texttt{dPC}\rfloor + 1) \cdot \texttt{dPC})\end{aligned} \tag{1}$$

where $\texttt{PagSN}_i$ is ith element in the array `PagSN`. The constraint that guarantees that there is at least one paging message every repetition period is

$$\begin{aligned}&(\forall 2 \leq i \leq \texttt{nBR})\\&(i-1)\cdot \texttt{rPer} - \texttt{dPC} < \texttt{PagSN}_i - \texttt{PagSN}_1 < (i-1)\cdot \texttt{rPer} + \texttt{dPC}\end{aligned} \tag{2}$$

We have also array `PagPN` of timestamps for paging messages which are transmitted every `dPC`.

Timestamps for `SIB10` and `SIB11` are elements of two-dimensional arrays, since several messages can be transmitted during the same paging cycle or repetition period. The constraint that defines that there are n System Information messages with `SIB11` during every repetition period is

$$(\forall 1 \leq i \leq \texttt{nBR}-1)(\forall 1 \leq j \leq n)\texttt{PagSN}_i < \texttt{SIB11Time}_{i,j} < \texttt{PagSN}_{i+1} \tag{3}$$

where `SIB11Time` is a two-dimensional array of timestamps of System Information messages with `SIB11`. It can be that UE reads different number of `SIB11` during different repetition periods, but since we were interested in incorrect behaviour, we did not not model in [5] all possible correct behaviours.

Secondary notification can come in one or several segments. $\texttt{SIB11Segment}_{ij}$ contains the segment number of `SIB11` with timestamp $\texttt{SIB11Time}_{i,j}$. The UE should read every segment at least once during every repetition period.

$$(\forall 0 \leq i < \texttt{nSeg})(\forall 1 \leq j \leq \texttt{nBR}-1)(\exists 1 \leq k \leq n)\texttt{SIB11Segment}_{j,k} = i \tag{4}$$

We also constrain the time difference between two consecutive `SIB10` received by UE in the same paging cycle and two consecutive `SIB11` received by UE in the same repetition period. The constraint on two consecutive `SIB11` received by UE is

$$\begin{aligned}&\forall(1 \leq i \leq \texttt{nBR}-1)\forall(1 \leq j \leq n-1)\\&(\texttt{SIB11Time}_{i,j+1} - \texttt{SIB11Time}_{i,j} > 0\\&\wedge\\&\texttt{SIB11Time}_{i,j+1} - \texttt{SIB11Time}_{i,j} \mod \texttt{siPer} = 0\\&\wedge\\&((\texttt{SIB11Time}_{i,j+1} - \texttt{SIB11Time}_{i,j})/\texttt{siPer}) \mod \texttt{nSeg} =\\&(\texttt{SIB11Segment}_{i,j+1} - \texttt{SIB11Segment}_{i,j}) \mod \texttt{nSeg})\end{aligned} \tag{5}$$

The model contains parameters that represent timestamps and content of `SIB1` messages. `SIB1SIB11Time` is a array of timestamps of `SIB1` messages during repetition periods.

`SIB1SIB11Type` is array of values from 0 to 3 that indicates whether `SIB1` contains `schedulingInfoList` for `SIB10` and/or `SIB11`. Then we post a constraint

$$\begin{aligned}
&\forall(1 \le i \le \mathtt{nBR})\\
&((\mathtt{SIB1SIB11Time}_i \le \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i \le \mathtt{PagSN}_{\mathtt{nBR}} \wedge\\
&\qquad \mathtt{SIB1SIB11Type}_i = 1)\\
&\bigvee\\
&(\mathtt{SIB1SIB11Time}_i > \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i \le \mathtt{PagSN}_{\mathtt{nBR}} \wedge\\
&\qquad \mathtt{SIB1SIB11Type}_i = 2)\\
&\bigvee\\
&(\mathtt{SIB1SIB11Time}_i \le \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagSN}_{\mathtt{nBR}} \wedge\\
&\qquad \mathtt{SIB1SIB11Type}_i = 3)\\
&\bigvee\\
&(\mathtt{SIB1SIB11Time}_i > \mathtt{PagPN}_{\mathtt{ndPC}} \wedge \mathtt{SIB1SIB11Time}_i > \mathtt{PagSN}_{\mathtt{nBR}} \wedge\\
&\qquad \mathtt{SIB1SIB11Type}_i = 0))
\end{aligned} \tag{6}$$

4.2 Model for protocol log analysis

In this section we present our new approach to use a constraint solver to find incorrect behaviour in protocol logs, by using a MiniZinc model of the correct behaviour of the protocol. There are some differences between a model in our previous work [5], outlined in Section 4.1 and the MiniZinc model here.

In Section 4.1 we had arrays `PagPN` and `PagSN` of paging messages. We keep the arrays in the MiniZinc model, but we introduce additional array `PagLog` of paging messages. `PagLog` contains timestamps of all paging messages from the log, and we use a constraint solver to check which paging message can be primary notification messages, and which can be secondary notification message. If a paging message is not first in the log we do not assign a value to $\mathtt{PagLog}_1$ and add the constraint $\mathtt{PagLog}_1 > 0$, otherwise we assign value 0 to $\mathtt{PagLog}_1$. Then

$$(\forall 2 \le i \le \mathtt{nSIB11Log})\mathtt{PagLog}_i = \mathtt{PagLog}_1 + \delta_i^{pag}, \tag{7}$$

where δ_i^{pag} is difference between timestamp of ith paging message in the log and timestamp of the first paging message in the log. As in [5] we define constraints on `PagPN` and `PagSN` to model possible time differences between paging messages, where $\mathtt{PagPN}_1 = 0$ and $\mathtt{PagSN}_1 = 0$. Then we check if there is a correspondence between `PagLog`, `PagPN` and `PagSN`.

In Section 4.1 we had the constraint (1) that defines time difference between two consecutive paging messages transmitted every repetition period, and the constraint (2) that guarantees that there is at least one paging message every repetition period. However, the exact sequence of timestamps of paging messages which are transmitted every repetition period can be captured by the constraint

$$\begin{aligned}
&(\forall 2 \le i \le \mathtt{nBR})\\
&((\mathtt{rPer} \cdot (i-1) - \mathtt{delay}) \mod \mathtt{dPC} = 0 \rightarrow\\
&\mathtt{PagSN}_i = \mathtt{rPer} \cdot (i-1) - \mathtt{delay})\\
&\bigwedge\\
&((\mathtt{rPer} \cdot (i-1) - \mathtt{delay}) \mod \mathtt{dPC} \ne 0 \rightarrow\\
&\mathtt{PagSN}_i = (((\mathtt{rPer} \cdot (i-1) - \mathtt{delay}) /\ \mathtt{dPC}) + 1) \cdot \mathtt{dPC})
\end{aligned} \tag{8}$$

We did not have constraint (8) in [5] since `delay` can be any value between 0 and `dPC` and test harness does not make checks based on (8). However, in the case when test harness is implemented as a constraint solver, (8) can be used to check that there is a value for `delay` such that the sequence of timestamps of paging messages from the log is a valid sequence.

The constraint that defines that among paging messages from the log there are messages that correspond to paging messages of primary notification with correct timestamps is

$$(\forall 1 \le i \le \mathtt{ndPC})((\exists 1 \le j \le \mathtt{nPagLog})\mathtt{PagLog}_j = \mathtt{PagPN}_i) \leftrightarrow \mathtt{PagPNinc}_i = 0, \tag{9}$$

where $\mathtt{PagPNinc}_i$ is a boolean variable which indicates that there is paging message in the log which corresponds to $\mathtt{PagPN}_i$.

The constraint that defines that among paging messages from the log there are messages that correspond to paging messages of secondary notification with correct timestamps is

$$(\forall 1 \le i \le \mathtt{nBR})((\exists 1 \le j \le \mathtt{nPagLog})\mathtt{PagLog}_j = \mathtt{PagSN}_i) \leftrightarrow \mathtt{PagSNinc}_i = 0, \tag{10}$$

where $\mathtt{PagSNinc}_i$ is a boolean variable which indicates that there is paging message in the log which corresponds to $\mathtt{PagSN}_i$.

We also have the constraint

$$\begin{aligned}&(\forall 1 \le i \le \mathtt{nPagLog})\\ &\quad(((\forall 1 \le j \le \mathtt{ndPC})\mathtt{PagLog}_i \ne \mathtt{PagPN}_j) \wedge ((\forall 1 \le j \le \mathtt{nBR})\mathtt{PagLog}_i \ne \mathtt{PagSN}_j))\\ &\quad\leftrightarrow \mathtt{Paginc}_i = 1,\end{aligned} \tag{11}$$

where $\mathtt{Paginc}_i$ is a boolean variable which indicates that ith paging message does not correspond to paging message of a primary or a secondary notification.

We first check that MiniZinc can find solution such that sum of elements of `PagPNinc`, `PagSNinc` and `Paginc` is equal to 0. If there is a solution we keep all constraints, but if MiniZinc does not find a solution we remove all constraints on `PagLog`.

Then we add constraints on a content and timestamps of `SIB1`, `SIB10` and `SIB11` messages. If a paging message is the first message in the log, then

$$(\forall 1 \le i \le \mathtt{nSIB11Log})\mathtt{SIB11TimeLog}_i = \delta^p_i, \tag{12}$$

where δ^p_i is difference between a timestamp of ith SIB11 message in log and a timestamp of first paging message in log.

If a paging message is not the first message in the log, then we have a variable $\mathtt{SIB11TimeLog}_1$ that represent timestamp of first `SIB11` message in the log and

$$(\forall 2 \le i \le \mathtt{nSIB11Log})\mathtt{SIB11TimeLog}_i = \mathtt{SIB11TimeLog}_1 + \delta^s_i, \tag{13}$$

where δ^s_i is difference between a timestamp of ith `SIB11` message in the log and a timestamp of first `SIB11` in the log. If a `SIB10` message is the first message in the log then $\mathtt{SIB11TimeLog}_1 = \delta^{s10}$, where δ^{s10} is time difference between first `SIB10` message and first `SIB11` message. If a `SIB1` message is the first message in the log then $\mathtt{SIB11TimeLog}_1 = \delta^{s1}$, where δ^{s1} is time difference between first `SIB1` message and first `SIB11` message.

We assign values to `SIB1TimeLog` and `SIB10TimeLog` using the same approach. We also assign values to `SIB1TypeLog` and `SIB11SegmentLog`.

The UE should read every segment at least once during every repetition period. Similar to (4), we have

$$\begin{aligned}&(\forall 2 \le i \le \mathtt{nBR})(\forall 1 \le k \le \mathtt{nSeg})\\ &\qquad((\exists 1 \le j \le \mathtt{nSIB11Log})\mathtt{SIB11SegmentLog}_j = k \wedge\\ &\qquad\qquad\mathtt{PagSN}_{i-1} < \mathtt{SIB11TimeLog}_j < \mathtt{PagSN}_i)\\ &\qquad\leftrightarrow \mathtt{PagSNSegmentinc}_{i-1,k} = 0,\end{aligned} \tag{14}$$

where $\texttt{PagSNSegmentinc}_{i,k}$ is a boolean variable which indicates that there is kth segment of secondary notification during ith repetition period.

Similar to (5) we constraint the time difference between to consecutive SIB11 messages

$$\begin{aligned}&(\forall 2 \leq i \leq \texttt{nSIB11Log})\\&\qquad((\texttt{SIB11TimeLog}_i - \texttt{SIB11TimeLog}_{i-1}) \bmod \texttt{siPer} = 0 \wedge\\&\qquad((\texttt{SIB11TimeLog}_i - \texttt{SIB11TimeLog}_{i-1})/\texttt{siPer}) \bmod \texttt{nSeg} =\\&\qquad(\texttt{SIB11SegmentLog}_i - \texttt{SIB11SegmentLog}_{i-1} + \texttt{nSeg}) \bmod \texttt{nSeg})\\&\qquad\leftrightarrow \texttt{SIB11TimeLoginc}_i = 0,\end{aligned} \tag{15}$$

where $\texttt{SIB11TimeLoginc}_i$ is a boolean variable which indicates that the timestamp of the ith SIB11 message is correct.

We check that there are no SIB11 messages after the last paging message of secondary notification

$$((\exists 1 \leq i \leq \texttt{nSIB11Log})\texttt{SIB11TimeLog}_i > \texttt{PagSN}_{\texttt{nBR}}) \leftrightarrow \texttt{SIB11afterpaginc} = 1 \tag{16}$$

where SIB11afterpaginc indicates that there is SIB11 message after last paging message of secondary notification.

We have also constraints on the timestamps of SIB10 messages.

In the previous section we had two lists of timestamps of SIB1 messages. Since in protocol log it can be difficult to differentiate between which SIB1 is after paging for primary notification and which SIB1 is after paging for secondary notification, we create one array SIB1TimeLog of timestamps of SIB1 messages in MiniZinc. SIB1TypeLog is list of values from 0 to 3 that indicates whether SIB1 contains schedulingInfoList for SIB10 and/or SIB11. Similar to (6) we have the constraint for SIB1TypeLog.

We minimize sum of all “inc” boolean parameters and we use “inc” parameters to indicate errors in the log.

5 Conclusion

We think that using MiniZinc for protocol log analysis is a promising approach, since it is easy to model the protocol in MiniZinc and a constraint solver can easily handle complex requirements on time stamps. In comparison with [5], we do not need to generate random values for parameters, since we have a solution, that is values from protocol log. Since we have a solution, constraint solver can handle bigger domains of parameters than in [5]. As a future work we plan to extend the model to be able to capture behaviour in UE after receiving several Write-Replace Warning Request messages from MME and to integrate constraint solver into automation environment.

References

[1] 3GPP. Evolved universal terrestrial radio access (E-UTRA) ; S1 application protocol (S1AP). TS 36.413, 3rd Generation Partnership Project (3GPP).

[2] 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).

[3] 3GPP. Public warning system (PWS) requirements. TS 22.268, 3rd Generation Partnership Project (3GPP).

[4] 3GPP. Technical realization of cell broadcast service (CBS). TS 23.041, 3rd Generation Partnership Project (3GPP).

[5] Kenneth Balck, Olga Grinchtein, and Justin Pearson. Model-based protocol log generation for testing a telecommunication test harness using clp. In *DATE*, 2014.

[6] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present, and future. *Constraints*, 12(1):21–62, March 2007. The current working version of the catalogue is at `http://www.emn.fr/z-info/sdemasse/aux/doc/catalog.pdf`.

[7] Mats Carlsson, Greger Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kuchen, editors, *PLILP 1997*, volume 1292 of *LNCS*, pages 191–206. Springer, 1997.

[8] SM Chadchan and CB Akki. 3GPP LTE/SAE: An overview. *International Journal of Computer and Electrical Engineering*, 2(5):806–814, 2010.

[9] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543, Berlin, Heidelberg, 2007. Springer-Verlag.

[10] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, 53(6):937–977, 2006.

[11] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.