EASN: Integrating ASN.1 and Model Checking

Vivek K. Shanbhag[1], K. Gopinath*[1], Markku Turunen[2], Ari Ahtiainen[2], and Matti Luukkainen[3]

[1] CSA Dept, Indian Institute of Science, Bangalore, India
{vivek,gopi}@csa.iisc.ernet.in
[2] Nokia Research Center, Helsinki, Finland
{markku.turunen,ari.ahtiainen}@nokia.com
[3] Department of Computer Science, University of Helsinki, Finland
Matti.Luukkainen@cs.Helsinki.fi

Abstract. Telecommunication protocol standards have in the past and typically still use both an English description of the protocol and an ASN.1[5] specification of the data-model. ASN.1 (Abstract Syntax Notation One) is an ITU/ISO data definition language which has been developed to describe abstractly the values protocol data units can assume; this is of considerable interest for model checking as ASN.1 can be used to constrain/construct the state space of the protocol accurately. However, with current practice, any change to the English description cannot easily be checked for consistency while protocols are being developed. In this work, we have developed a SPIN-based tool called EASN (Enhanced ASN.1) where the behavior can be formally specified through a language based upon Promela for control structures but with data models from ASN.1. We use the X/Open standard on ASN.1/C++ translation so that our tool can be realised with pluggable components. We have used EASN to validate a simplified RLC in the W-CDMA (3G GSM) stack. In this short paper[1], we discuss the EASN language, the tool, and an example usage.

1 Introduction

Next generation protocols for mobile devices have become very complex and it is becoming increasingly difficult for standards bodies to be sure of the correctness of protocols during the standardization process. This has become an impediment in defining new standards. What one needs is a way of specifying an evolving protocol and have some confidence that, at a certain level of abstraction, the protocol is consistent inspite of modifications.

Why ASN.1? There are languages like Promela that can be used, but their data structuring capabilities do not match those of ASN.1, for instance, that is

* Supported by funding from Nokia Research Center, under SID project 99033.
[1] See [2] for a full paper discussing the implementation & some performance indicators.

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 382–386, 2001.

widely used in telecommunication protocol specification. It will help the standardization process if a model checker could be augmented with ASN.1 data modeling capabilities to check correctness of interim versions of a protocol before establishing a standard.

ASN.1 separates data modeling into abstract and transfer syntax. The abstract syntax only specifies the universe of abstract values that can be assumed by variables in the model without any concern for how they are mapped to a particular machine, compiler, OS, etc. Hence from the point of view of model checking, an abstract syntax constrains the state space as much as possible IF there is a mechanism by which a system state vector can be encoded with exactly only the possible values of its constituent substates. The latter is a chief feature of the state compaction infrastructure that has been developed for the EASN system described here. ASN.1 has a subtyping feature with a well developed notation for expressing constraints. Note that data here actually means the control data in the protocols and hence our concerns are different from those approaches that exploit symmetry, etc. We derive our EASN tool by marrying ASN.1 with the well known model checker SPIN.

Why SPIN? SPIN[1] is an effective model checking tool for asynchronous systems, especially designed for communication protocols. Nondeterminism and guarded commands in Promela (input language of SPIN) makes it convenient to express behavior of communicating protocol entities. SPIN has many capabilities for validation of safety and liveness properties[4]. Algorithms that effect substantial space and time savings, like bit-state hashing, on-the-fly[3] model-checking and partial-order reduction have been incorporated into SPIN.

SPIN has a **simulator** that randomly checks only a portion of the state space and also a (generated) **validator** that can attempt to exhaustively check the state space of the system or can use techniques like bit-state hashing to check a substantial portion of the state space with a fairly high level of assurance. Our EASN system also has these components.

EASN Language. ASN.1 can only be used to define the datatypes and constant values in an application. Promela, however, is a complete language with a set of basic data types and typedef construct to help users compose datatypes, and control constructs that are used to define the behavior of protocol entities.

The EASN Language ***replaces*** all the datatyping capabilities of Promela with ASN.1. Hence, none of the data types of Promela are retained in EASN, except the *chan* construct. As ASN.1 has richer and more expressive datatypes compared to Promela, EASN needs to overload the semantics of many of the operators of Promela, so as to support a natural set of operations on data. In addition, we have also augmented the set of operators as necessary. In brief,

EASN = Promela - {mtype, typedef, bit, byte, bool, short, int} + ASN.1
+ overloaded semantics for existing operators + few new operators.

2 EASN, the Verification Tool

Encoding State Efficiently: SPIN represents state quite efficiently but, for reasons of alignment, allows padding and other extraneous matter in the state vector. Since our system uses ASN.1 data modules, we require that all variables be as constrained as possible in the space of values that they can take through the use of subtyping. For instance, if there are only two variables, that are constrained between (say) 5..7 and 3..7, there are only 15 possibilities, and can be represented in only 4 bits instead of either 2+3 (5 bits) or worse 3+3 (6 bits)[2].

Our state compaction infrastructure views the state space of the system as a multi-dimensional array (with one dimension for every component of the state), and consequently, every state of the system, as a point in this multi-dimensional space. We use column-major linearisation.

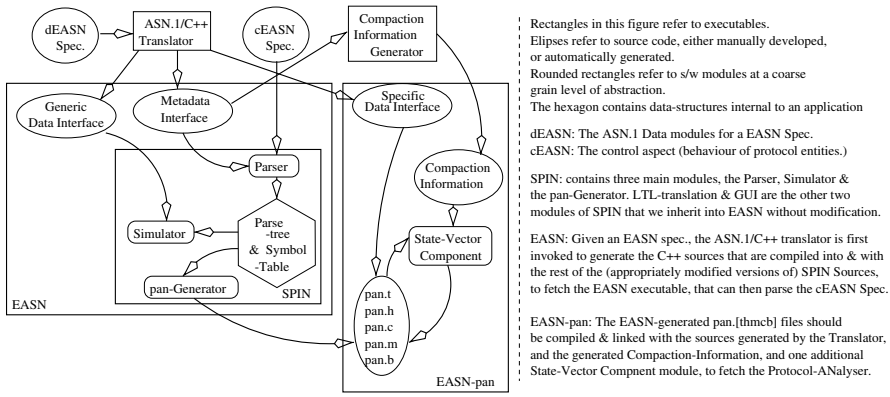
SPIN does various kinds of state compaction, and in EASN, we have a comparable mechanism for most of them that perform atleast as well in space. But some are unnecessary in EASN. Geldenhuys and Villiers[8] also attempt state compression in SPIN along similar lines as ours but by adding a simple construct to Promela but with restrictions. For example, different orders of process activation along different execution paths are forbidden in their approach as much of the state component placement is done statically. The ranges of their variables must start at zero. We do not have such restrictions.

The EASN Tool: SPIN is open source[7]. We intend EASN to be open source too. NRC has an ASN.1 parser that we could use but we did not want to compromise others from using EASN as open source. We, therefore, have used the X/Open ASN.1/C++ translator std[6] to architect the tool so as to enable other users besides us and NRC to realise it by plugging in any compliant ASN.1/C++ translator into the system. A block diagram of the EASN system is given in the accompanying figure.

An EASN system specification (for simulation/verification) consists of two *compilation units*. One containing all the ASN.1 modules (the dEASN spec.) that is parsed by the ASN.1/C++ translator to generate C++ source, and the other containing the *behavioral* specification of the protocol entities (the cEASN spec.) that is parsed by the EASN parser (a modified Promela parser, derived from SPIN). It is the variable declarations in the cEASN spec that ties it to the dEASN spec as their types are defined in the ASN.1 modules. The EASN parser *imports* all the relevant information regarding a type, from the generated C++ source, by querying its meta-data interface.

The Parser and Simulator: The executable that can parse and simulate a given cEASN spec. is fetched from linking the C++ generated by the translator (corresponding to the associated dEASN spec.) along with all the (appropriately modified) SPIN modules. This executable is the EASN tool. The EASN simulator requires to access data values and modify them through permitted operations.

[2] Experienced ASN.1 users may note that such an encoding is even better than the often very compact PER encoding.



However, since the simulator engine has no knowledge of the specific ASN.1 types that might be used in different EASN specifications, these data operations must be carried out using the *ASN.1/C++ Generic Data Interface* that supports operations on objects conforming to the *ASN.1 data-model*.

The Generated Validator: SPIN generates C code that is compiled to obtain the validator. EASN, however, generates C++ code that has to be linked with the code generated by the ASN.1/C++ translator, the code generated by the *Compaction information generator* and the compaction infrastructure to obtain the validator.

The *compaction information* is a set of C++ functions that export all information about value-constraints expressed in the original ASN.1 spec, through the C++ interface as required by the generic *compaction infrastructure* module. Through these two additional components of our framework, we implement incrementally the computation of the linearised representation of the state of the system that needs to be stored into the hash-table, and also the hash-value of the bucket in the table.

RLC/ABP Examples: We have used EASN to validate a simplified RLC in the W-CDMA (3G GSM) stack. It uses less memory but more time than SPIN. Further details of the performance of EASN have been submitted to the FMICS workshop. Due to its length, we present a much simpler ABP protocol in figure 2. Note that the state vector for EASN is half the size of SPIN's.

Correctness of Implementation *vis-a-vis* **SPIN:** In crafting EASN from SPIN, we identified the following invariant that could be a necessary and sufficient condition to convince oneself that our implementation is sane:

> Given a Promela spec. s and a cEASN spec. e, derived from s by changing its variable types to equivalent ASN.1 types (defined in an ASN.1 module

appropriately imported into EASN): A. Simulation runs of SPIN over s and of EASN over e should select identical sequence of state-transitions, for the same seed value; B. The sequence in which the reachable states of the system are visited by the generated validators (by SPIN for s and by EASN for e) must be identical (for exhaustive searches), with/without partial-order reduction or never-claims.

EASN preserves this invariant for all the tests that we have tried so far.

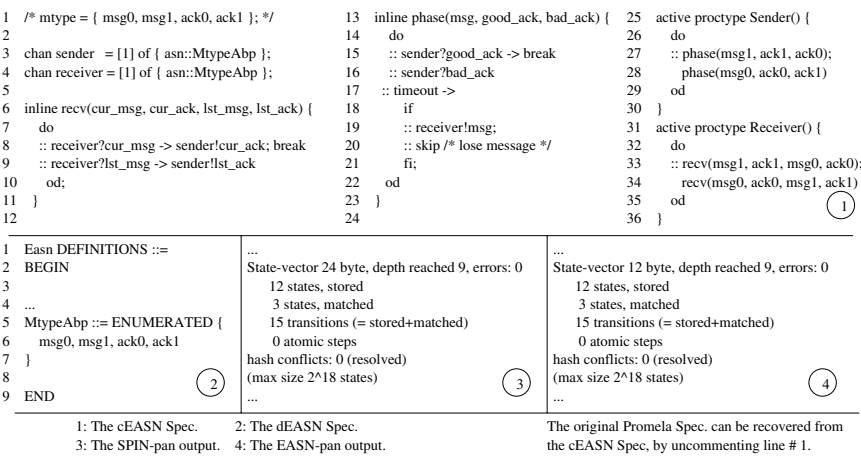


Fig. 1. ABP in SPIN and EASN.

References

1. Holzmann, Gerald J., Doron Peled, "The state of SPIN", CAV '96.
2. Shanbhag, Vivek K., K. Gopinath, "A Spin based model checker for telecommunication protocols", May 2001, 8th Intl SPIN Workshop on Model Checking of Software.
3. G. Gerth, D. Peled, M. Y. Vardi, P. Wolper, "Simple On-the-fly Automatic Verification of Linear Temporal Logic", PSTV94.
4. Holzmann, G.J., Design and Validation of Computer Protocols, Prentice Hall, 1992.
5. Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, ITU-T Rec. X.680 (1994); Information Object Specification, ITU-T Rec. X.681 (1994); Constraint Specification, ITU-T Rec. X.682 (1994); Parametrization of ASN.1 specifications, ITU-T Rec. X.683 (1994).
6. ASN.1/C++ Application Programming Interface, Part 1: Base Classes & Specific Interface, NMF 040-1; Part 2: Generic Interface, NMF 040-2, Issue 1.0, Feb. 1998
7. Holzmann, G.J., SPIN Sources, Version 3.4.6, 29th March. 2001.
8. J.Geldenhuys, PJA de Villiers, 'Runtime Efficient State Compaction in SPIN,' 5th Intl SPIN Workshop on Theoretical Aspects of Model Checking, ed. D. Dams, M. Massnik.