The Use of Conditional Grammars for Specifying and Verifying Communication Protocols

Petr Matoušek
Dept. of Computer Science and Informatics
University of Technology Brno
Božetěchova 2, 612 66 Brno
Czech republic
E-mail: matousp@dcse.fee.vutbr.cz

Keywords: protocol, formal method, conditional grammar, verification, communication

Abstract

Formal specification of communication protocols is not a new problem. It seems to be a routine to formally specify protocols using regular grammars, finite state machines, extended FSM, Petri nets or other tools. These can partially satisfy designer's requirements for protocol design but mostly they can be applied only on a simple version of a protocol. To describe real-world application protocols with protocol parameters, time constraints, synchronization in distributed environment seems to be difficult by these means. The paper presents a new approach to protocol specification based on conditional grammars. It demonstrates that proposed formal specification is both intuitive and descriptive, and it offers uniform formal approach to different classes of protocols. This specification is a good base for following simulation and verification.

1 Introduction

Importance of Internet grows year to year. Many services are built to be accessed via Internet. They are based on a standard client-server architecture communicating by a protocol. The protocol defines communication sequences interchanged among two or more partners.

For successful building of such system based on communication protocols formal specification and further analysis of the specified protocol is required. The specification should unambiguously describe the system, show its main features and behavior and - on the other hand - reveal possible bottle-necks and flaws in design. Formal specification is a good base for system verification and simulation.

Mostly the formal specification and testing is applied only on a simple version of a protocol, because standard formal methods for protocol description (regular grammars, finite state machines, Petri nets) are not suitable for the specification of a full protocol, e.g. with protocol parameters, time constraints etc. By that simplification the protocols are examined only from structural points of view.

The paper presents a new formal framework for the protocol specification based on conditional grammars. It is a part of the author's PhD. project *"Formal description and verification of a class of high-level protocols with real-time constraints"*.

The first part of this paper introduces the motivation and a part of background theory of the presented work. In next parts the proposed theory is presented and demonstrated on an example. The last part of the paper resumes benefit of the proposed approach in protocol verification and depicts future steps of the research.

2 Theoretical background

Considering the protocol as a dialogue consisting of sentences it is natural to use formal grammars to formally specify it. The sentences characterize the communication and the interaction between peer entities.

The grammar is formally characterized by a tuple

$$G = (N, T, P, S),$$

where N is *a final set of nonterminals*, T is *a final set of terminals*, P is *a set of rules* and S is the starting nonterminal ($S \in V$).

Considering communication protocols, the set T is composed of symbols sent by both communicating parties (i.e. commands). Both set of production rules and set of nonterminals are constructed according to expected communication sequences. For a class of high-level communication protocols we expect a half-duplex communication channel, and the protocol to be of type *request/response*, i.e. every command sent by one partner (called request) yields appropriate response (confirmation, data, etc.).

Mostly only single commands without any additional information are supposed. This can model a simple structure of the system using, for instance, a finite state machine or regular grammar. However, response depends not only on a request command but mostly on request command parameters that determine an appropriate answer of the system.

The proposed formalism is based on conditional grammars. Its design is descriptive and easy to read. It provides formalism to define parameters as a part of terminals. Every rule contains a condition - to make behavior deterministic according to protocol parameters. Consequently, only those productions are applied on a sentential form that satisfy a corresponding condition evaluated by protocol parameters.

A global context conditional grammar (GCC-grammar) of degree i is according to [5] a triple

$$(G, FOR, PER),$$

where G=(N,T,P,S) is *a context free grammar*, FOR $\subseteq (N \cup T)$, and PER $\subseteq (V \cup N)^+$ such that $y \in$ PER implies $|x| \leq i$.

A context free grammar is a quadruple $G = (N, T, P, S)$, where N, V and S have the same meaning as written above and every production in P is of the form $A \to x$, where $A \in N, x \in (N \cup T)^*$.

A production as $A \to x \in P$ can be applied to a word w only when

(i) no forbidding symbol from FOR occurs in w, or

(ii) A occurs in a permitting word from PER which is a subword of w.

A global context conditional grammar works with static conditions, i.e. the definition of GCC grammar contains both sets FOR and PER, and conditions depends only on a current sentential form. For real world protocols those conditions that depend not only on a current sentential form but also on protocol parameters and the state of internal memory of the system are required.

The principle of the internal control is not new. In their work [3, pp. 98-100], the authors introduce a cooperating distributed grammar system with memories (mCD grammar system). The system is based on semi-conditional grammars. The productions depend on a left-side nonterminal and as well on permitting and forbidding context conditions and the state of a memory string. The memory is implemented as a queue or a stack containing both terminals and nonterminals. Then rewriting is regulated by existence of a certain symbol in the memory string.

In our approach we propose a different view on context conditions. Just as in GCC grammar or in mCD grammar we test existence of a substring u in a given set or a memory string. Unlike GCC grammar our "set FOR" is not static but it is dynamically "updated". In contrast to mCD grammar we allow searching a substring along the whole sentential form, not only at the beginning or at the end.

Conditions in our case are represented in form of dynamic memory string - a sentential form that is rewrited by special grammar. This sentential form contains parameters of the protocol. Instead of rewriting the first or the last symbol of the memory string (as proposed by mCD grammar systems) we allow rewriting of any symbol within the memory string. Productions of this grammar supports adding of a parameter, deleting or testing the existence of a parameter in the memory string.

- *A memory grammar* is a quadruple $G_m = (N_m = \{S_m, Symbol(x)\}, T_m = \{x\}, P_m, S_m)$, where P_m is a final set of productions of three types:

  (r_1) $S_m \to S_m\ Symbol(x)$ ... *addition rule*

  (r_2) $Symbol(x) \to \epsilon$ ... *deletion rule*

  (r_3) $Symbol(x) \to Symbol(x)$ ... *existential rule*

In our proposal a parameter of a protocol command is a value associated with a terminal representing protocol command - for instance *add_user(x)* represents command *add_user* and its parameter x. The value is a part of a given sentential form closed in braces (). In one derivation step the values can be assigned to nonterminals as attributes (cf. attribute grammars [1]). Besides, they are evaluated and assigned using productions of context conditions.

For the use of parameters we define *a grammar with parameters (GP)* as a quadruple

$$G_1 = (N_1, T_1, P_1, S_1),$$

where N_1 is *a final set of nonterminals*, T_1 *a final set of terminals*, P_1 is *a set of semantic productions*, and S_1 is *the starting symbol.* Productions are of form

- $(A \to x; \emptyset)$, where $A \in N_1, x \in (N_1 \cup T_1)^*$, or
- $(A \to xy(\alpha)z; C.par = \alpha)$, where $A \in N_1, x, z \in (N_1 \cup T_1)^*, y(\alpha) \in T_1 \times N_1$ and $C \in N_1$ is a right side nonterminal of this production.

A production can contain a semantic part that assigns a value of parameter α to a nonterminal on the right site of a production. Values of parameters are stored in a similar way like attributes in the attribute grammar.

Roughly speaking, the GP grammar is a context free grammar with some internal semantic actions.

Let $u, v \in (N_1 \cup T_1)^*$. Then we say *u directly derives* v according to $A \to y$ or $A \to y(\alpha)$, denoted by $u \Rightarrow v\ [p]$, if exists $p \in P_1, p = A \to y$ or $A \to y(\alpha)$ and $u = xAz, v = xyz$ or $v = xy(\alpha)z$ for some $x, y, z \in (N_1 \cup T_1)^*, y(\alpha) \in T_1 \times N_1$.

Now, we define *a grammar system with conditions and parameters (GCP)*. A GCP grammar system is a tuple $\Gamma = (G, G_1, S)$, where

(i) G_1 is GP grammar $G_1 = (N_1, T_1, P_1, S_1)$ as defined above

(ii) G is a quadruple (N,T,P,S), where

- $N = \{N_1 \cup \{S\} \cup N_m = \{S_m, Symbol(\alpha)\}\}$
- $T = \{T_1 \cup \{\alpha\}\}$
- P is a final set of production of form

  (a) $p = (A \to x; \emptyset; r)$, or

  (b) $p = (A \to xy(\alpha)z; C.par = \alpha; r)$

  where $C \in N$ is a substring of x, $A \in N, x, z \in (N \cup T)^*, y(\alpha) \in (N \times T)$ and r is either one production from a set R

  (r_1) $\{(S_m \to S_m\ Symbol(x)),$

  (r_2) $(Symbol(x) \to \epsilon),$

  (r_3) $(Symbol(x) \to Symbol(x))\},$

  or empty set $\emptyset$.

(iii) S is the starting symbol in a form $(S_1; S_m)$

Let $u, v \in (N \cup T)^*, m, m' \in \{S_m, Symbol(\alpha)\}^*, u = (xAz; m), v = (xyz; m')$ for some $x, y, z \in (N \cup T)^*$. Then we say u directly derives v according to $p = (A \to y; \emptyset, r)$ or $p = (A \to y(\alpha); C.par = \alpha; r)$, denoted by $u \Rightarrow v\ [p]$, *iff* $p \in P$ and one of the following conditions holds:

- (i) r is empty set $\emptyset$, or
- (ii) exists $r \in R$ such that $m \Rightarrow m'[r]$.

In the standard manner, we can define $\Rightarrow^i$ (for $i \geq 0$), $\Rightarrow^*, \Rightarrow^+$.

The language of Γ, denoted by $L(\Gamma)$, is defined by

$$L(\Gamma) = \{(w; m) \in (T^*; m \in N_m^*) \mid (S; S_m) \Rightarrow^* (w; m)\}.$$

Language $L(\Gamma)$ describes a protocol sentence accepted by the protocol defined using grammar Γ. In comparison with other approaches the specification includes parameters of the protocol and the memory that represents the internal state of the protocol. This brings us advantages as follows:

- A protocol is specified using simple context free grammar which is easy to create and read.
- Protocol parameters are added to a grammar definition and to a sentential form in a descriptive form.
- Context conditions are defined in a form of productions over memory that contains parameters. A string of parameters (the memory) represents dynamic set of context conditions. Using special grammar we are not limited by certain number and type of parameters and before applying any rule we can test their existence in a memory string.
- Memory string evolves in the course of rewriting. At the end it represents a state of a system after protocol operation. This can be used for server/client simulation.

3 Case study

The formalism introduced in the previous part is demonstrated on a simple communication protocol of user database. Following requests are implemented: ADD_USER, REMOVE_USER, BYE, and corresponding answers: USER_ADDED, USER_REMOVED, ALREADY_EXISTS, NO_SUCH_USER.

Request commands ADD_USER, REMOVE_USER contain a parameter - username. According to the value of the parameter corresponding answer is generated.

For instance, possible answers for command ADD_USER are USER_ADDED or ALREADY_EXISTS. Regular grammar would apply one of these answers in a non-deterministic way. But the answer depends good deal on an argument (parameter) of the command. Considering e.g. argument *username* we can determine the answer after evaluation of that argument.

Context free grammar G=(N,T,P,S) of the system is defined as follows:

(i) N = {A,B,C,S}

(ii) T = {add_user, remove_user, bye, user_added, user_removed, already_exist, no_such_user}

(iii) P is a set of productions:

  (a) S $\to$ hello A

  (b) A $\to$ add_user B | remove_user C | bye

  (c) B $\to$ user_added A | already_exists A

  (d) C $\to$ user_removed A | no_such_user A

Now we add parameters (semantic action) to this grammar so a new GP grammar is $G_1 = (N_1 = \{N \cup \{\alpha\}\}, T, P_1, S_1)$. Set of productions P_1 is following:

(a) $\{(S_1 \to hello\ A; \emptyset),$

(b) $(A \to add\_user(\alpha)\ B; B.par = \alpha),$
$(A \to remove\_user(\alpha)\ C; C.par = \alpha),$
$(A \to bye; \emptyset),$

(e) $(B \to user\_added\ A; \emptyset),$
$(B \to already\_exists\ A; \emptyset),$

(d) $(C \to user\_removed\ A; \emptyset),$
$(C \to no\_such\_user\ A; \emptyset)\}$

A GCP system of this system is a triple $\Gamma = (G, G_2, S)$, where

(i) $G_1 = (N = \{N_1 \cup \{S_m, User(x)\}\}, T_1 \cup \{x\}, P, S)$ is an GP grammar with productions P as follows:

(a) $\{(S_1 \rightarrow \mathit{hello}\ A; \emptyset; \emptyset),$

(b) $(A \rightarrow \mathit{add\_user}(\alpha)\ B; B.\mathit{par} = \alpha; \emptyset),$
$(A \rightarrow \mathit{remove\_user}(\alpha)\ C;$
$C.\mathit{par} = \alpha; \emptyset),$
$(A \rightarrow \mathit{bye}; \emptyset; \emptyset),$

(c) $(B \rightarrow \mathit{user\_added}\ A; \emptyset;$
$S_m \rightarrow User(B.par)\ S_m),$
$(B \rightarrow \mathit{already\_exists}\ A; \emptyset;$
$User(B.par) \rightarrow User(B.par)),$

(d) $(C \rightarrow \mathit{user\_removed}\ A; \emptyset;$
$User(C.par) \rightarrow \epsilon),$
$(C \rightarrow \mathit{no\_such\_user}\ A; \emptyset; \emptyset)\}$

(ii) $S = (S_1; S_m)$

Here, there are some examples of derivation process:

- $(S_1; S_m)$ $\Rightarrow$ (hello A; S_m) $\Rightarrow$ (hello add_user(a) B;S_m) $\Rightarrow$ (hello add_adduser(a) user_added A; User(a) S_m) $\Rightarrow$ (hello add_adduser(a) user_added bye; User(a) S_m)
- $(S_1; S_m)$ $\Rightarrow$ (hello A; S_m) $\Rightarrow$ (hello add_user(a) B; S_m) $\Rightarrow$ (hello add_adduser(a) user_added A; User(a) S_m) $\Rightarrow$ (hello add_adduser(a) user_added remove_user(a) C; User(a) S_m) $\Rightarrow$ (hello add_adduser(a) user_added remove_user(a) user_removed A; S_m) $\Rightarrow$ (hello add_adduser(a) user_added remove_user(a) user_removed bye; S_m)

4 Conclusion

This paper introduces a new uniform technique for formal specification of high-level communication protocols. It is based on conditional grammars theory. The paper shows a complete theory and demonstrates the proposed approach on a simple example. Because of its descriptive power seems to be a good tool for real-world protocol specification.

Using GCP grammar we can describe a communication protocol more precisely - parameters are part of the language. In addition context conditions specify how communication partners behave depending on parameters - that is very useful for implementation. Because of the formal description we can use various verification techniques to prove that the communication system responses as expected. Future work is focused on protocol verification using PVS (Prototype Verification System) and definition of real-time constraints as a part of the GCP language.

Acknowledgment

The work presented here in the paper is a part of a project supported by grant FRVŠ no. 99/2001 *"Selection of methods for formal specification and verification of communication protocols"* of Czech Ministry for Education. The author is indebted to Dr. Alexander Meduna and Martin Švec for their valuable suggestions and advises about the grammar systems.

References

[1] Aho, A.V., Sethi, R., Ullman, J.D.:*Compilers. Principles, Techniques, and Tools.* Addison-Wesley publishing Company, 1986.

[2] Crow, J., Owre, S., Shankar, N., Srivas, M.: *A Tutorial Introduction to PVS*, SRI Int., 1995. Available at URL http://www.csl.sri.com/sri-csl-fm.html.

[3] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.:*Grammar Systems - A Grammatical Approach to Distribution and Cooperation*, Budapest, 1992.

[4] Matoušek, P.: *Protocol Proving Using PVS: A Case Study*, to appear in the Proceedings of Conference MOSIS 2001.

[5] Meduna, A.: *Global Context Conditional Grammars*, J. Inform. Process. Cybern. EIK 27 (1991) 3, pp.159 - 165.

[6] Murphy, S.L., Shankar, A.U.:*Service Specification and Protocol Construction for the Transport Layer*, ACM Communication architectures and protocols, 1988, Pages 88-97.

[7] Tarnay, K.:*Protocol Specification and Testing*, Akademia Kiado es Nyomda Vallalat, Budapest, 1991.

Biography

PETR MATOUŠEK was born in Ústí nad Labem, Czech republic and went to University of Technology Brno, where he received in 1997 an MSc. degree in Informatics and Computer Science. In 1997-1998 he worked at CERN, Geneva in IT division on the project of global distributed LDAP directory. Since 1998 he continues his PhD. studies at University of Technology, Brno, focused on formal description and verification of protocols with real-time constraints. He externally teaches at the University.
Contact - http://www.fee.vutbr.cz/˜matousp.