Delft University of Technology, AVS Report #38, June 1982

CONCISE DESCRIPTION of a PROTOCOL VALIDATION ALGEBRA

Gerard J. Holzmann
Delft University of Technology
The Netherlands

The following description of a protocol validation algebra is a simplified and purified version of the one discussed in [1]. The current description will be used for the implementation of the Pandora system [2].
For a more gentle introduction to the theory developed here the reader is referred to [1].

1. Definitions

1.1 Message Queues

A message queue is defined as a pair (S, R), where
S is a set of symbols, named the "queue sort," and
R is a tuple (I, O), named the "queue record", where
I is an ordered set of symbols, named the "append string," and
O is an ordered set of symbols, named the "accept string."
Initially both strings I, and O are empty.

Comments:
Every symbol in S, I, and O represents one specific message or message type. The "sort" of a message queue contains all symbols known to that queue. A queue can only hold messages of the types that are defined in its sort. The size of a message queue, and hence the size of strings I and O, is not bounded.

As special cases of the validation algebra one could consider its application to bounded message queues, in particular to queues of size 1 or 0 (cf. CCS [3]), and to message queues with a sort of size 1 (single token message passing, cf. Petri Net theory).

1.2 Message Passing System

A message passing system is defined by a tuple (M, P), where
M is a non-empty, finite set of message queues, with mutually disjoint sorts, and
P is a protocol expression.

Comments:
The expression P models the behavior of processes in a message passing system. The

interaction of the processes is restricted to the exchange of messages via the queues defined in M. A message name must be unambiguous, that is: from it we must be able to deduce precisely which queue is addressed. A process can only accept messages from one specific queue, named it's "mailbox". No two processes share the same message queue. Any process can address any queue in M, though.

1.3 Vocabulary

The union of all sorts of the queues in a message passing system is called the systems "vocabulary," and is indicated by the letter V.

1.4 Special Symbols

The set of special symbols is named Φ. It contains N+1 symbols, where N is the number of message queues in the system. There is one symbol ϕ_i for every queue. The phi symbols are sometimes referred to as "blocking symbols." The N+1st element is the no-op symbol "1".

By definition:

$$\phi_1 + \phi_2 + ... + \phi_N = D$$

D is named the "deadlock symbol."

Comments:
The symbol D replaces the zero in the earlier versions of the algebra [1]. The above definition allows for a better exposition of the reduction rules (2.5).

1.5 Protocol Expressions

A protocol expression is defined recursively as follows:

- every symbol from set V or set Φ is a protocol expression,
- if p and q are protocol expressions, then so are: (p), p+q, p.q, p/q, and pXq .

Comments:
The evaluation priority of the operators in the protocol expressions is, in descending order: / (slash), . (dot), + (plus), and X (cross). Parentheses can be used for grouping.
A symbol in de denominator of a fraction is a symbol to be sent (to be appended to a message queue). A symbols in the numerator of a fraction is a symbol to be received (to be accepted from a message queue).
The dot operator is used to indicate time ordering. In a dot product "p.q.r" the symbols are processed in the order p, q, r.
The plus operator can indicate ambiguity or selection. In the expression "p+q" we specify a choice between two possible behaviors: "p" and "q".
The cross operator, finally, is used to indicate concurrency. Each term in a cross product represents one "process."
We restrict ourselves here to terminating processes. For this reason the Kleene star operator is not included in the above set.

1.6 String

A "string" is defined as a dot product of fractions of symbols in V or Φ. A string can be represented as an ordered set of symbols.

1.7 The Domain of a String

The domain of a string is defined as the union of the domains of the numerators of all fractions in the string.
The domain of symbol "a" from sort "S-i" is S-i.
The domain of symbol "a" from set Φ is empty iff "a" equals "1", and is Φ in all other cases.

1.8 The Range of a String

The range of a string is defined as the union of the domains of the denominators of all fractions in the string.

1.9 The Queue record of a String

The "queue record" of a string A is obtained in 2 steps:

1) First split string A into two parts: an append string which contains all denominators of the fractions in A, and an accept string with all numerators, while preserving the original orders.
2) For each sort S-i in the system (M, P), do:

a) assign all symbols that both occur in sort S-i and in the append string obtained in step 1 to I-i, while preserving order;

b) assign all symbols that both occur in sort S-i and in the accept string obtained in step 1 to O-i, preserving order.

1.10 Equivalence of Strings

Two strings are said to be "equivalent" if their queue records are equal.

2. Rules

In the equations below the symbols p, q, and r represent arbitrary protocol expressions, as defined in (1.5). Symbol "a" represents an arbitrary symbol from set V. Symbols f and g are fractions of symbols in V.

2.1 Associativity

$(p \times q) \times r = p \times (q \times r)$
$(p + q) + r = p + (q + r)$
$(p \cdot q) \cdot r = p \cdot (q \cdot r)$

2.2 Commutativity

$p \times q = q \times p$
$p + q = q + p$

2.3 Distributivity

(p + q) . r = (p . r) + (q . r)
(p . q) + (p . r) = p . (1 . q + 1 . r)
p / (q + r) = (p / q) + (p / r)
p X (q + r) = (p X q) + (p X r)
(p + q) / r = (p / r) + (q / r)
(p + q) X r = (p X r) + (q X r)

2.4 Inverse

1 / (p / q) = 1 / (p . (1 / q)) = (1 / p) . q
p / 1 = p

2.5 Reduction

p + p = p
p . 1 = p
p X 1 = p

ϕ_i . p = ϕ_i (for all i)

a . q + ϕ_i = a . q iff symbol a is in set S-i

p . 1/a . q . a = p . q with symbol a in set S-i
iff range(p) and S-i are disjoint, and
domain(q) and S-i are disjoint, and
I-i is equal to O-i "at" p (i.e. "directly before" p),
where I-i and O-i are in the queue record of p.

p . a . q = p . ϕ_i with symbol a in set S-i
iff head(I-i - O-i) does not equal "a"
where I-i and O-i are in the queue record of p,
(I-i - O-i) is the remainder of string I-i after
prefix O-i has been deleted, and
head(T) is the first element of string T, and
I-i and O-i are equal at p.

2.6 Expansion

f.p X g.q = f . (p X g.q) + g . (f.p X q)

where f and g are fractions of the type "a/1" or "1/a", with "a" any element from V.

Comments:
- Application of the expansion and reduction rules should always lead to an expression which consists of any combination of the four terms in:

1 + D + 1.D + 1/R

where R is a dot product of "residual messages."

The presence of the term "1" indicates that the system is at least capable of executing "properly" (there is a completely specified subsystem, which executes without residuals

and without deadlocks.)

The presence of any term with "D" indicates a potential deadlock. The presence of any term "1/R" indicates a lack of block structure (see [1]).

- Note that the following rule is not included in the above set:

  p . (q + r) = p.q + p.r

In the left-hand expression the plus indicates a selection between a "q" and an "r" continuation, but in the right-hand expression the plus indicates a non-deterministic choice between "p" and "p". Assuming that performing "r" after "p" would mean an immediate deadlock we find that the choice on the left-hand side can only be resolved in favor of the "q" option. But, on the right-hand side the choice point was moved forward in time, and hence a deadlock in "p.r" will be unavoidable if the option "p.r" was selected. So, the right-hand side system contains an error that is not present in the left-hand side system. For this reason we have rejected the above rule (cf. [3]).
For similar reasons we have rejected the rule "1 . A = A" .

- The expansion of a cross product of two protocol expressions into a sum of strings can easily be automated (rule 2.6). By taking advantage of the reductions rules (2.5) and especially the equivalence rule (1.10) the time complexity of an analysis by expansion can be adequately bounded. Note that from each equivalence class implied by the relation (1.10) we need only examine one single representative string, and we only need to examine it up to the point where we can prove a block, a residual, or a successful completion.

The completeness and consistency of the rules remains to be proven.

3. References

[1] G.J. Holzmann, A theory for protocol validation, IEEE Transactions on Computers, August 1982.
[2] G.J. Holzmann, PANDORA, an interactive protocol development system, in preparation.
[3] R. Milner, A calculus for communicating systems, Lecture Notes in Computer Science, Vol. 92, 1980.