Pan - A Protocol Specification Analyzer

Gerard J. Holzmann
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Pan is a program that can analyze the consistency of protocol specification for up to ten interacting processes. The validation method is based on a special algebra for an extended type of regular expressions, named protocol expressions. A protocol specification is written in a CSP-like language that includes concatenation, selection, and looping (but no variables). Pan analyzes an average protocol faster than troff prepares an average paper. Unlike troff though, the execution time of pan is more sensitive to the quality than to the size of its victim.
(Bell Labs Technical Memorandum 81-11271-5, May 22, 1981.)

1. Introduction

Pan can be used to trace design errors in a protocol specification. It is used together with the program *prc* (a protocol specification compiler), and *orphans* (a tracer of residual messages).

In the specification language, the term *p?msg* indicates that message *msg* is received from process *p*. Similarly, the term *p!msg* indicates that message *msg* is sent to process *p*. The symbol :: can be read as *or*, the symbol -> can be read as *then*. Comments are enclosed by the # symbol and the end of a line. The precise syntax rules for the specification language can be found in the appendix. The usage of prc, pan, and orphans will be illustrated with the examples below.

2. First Example

Consider the following protocol for two processes named *M* and *W*.

```
PROCESS (M)

# Connecting Phase:
W!init; W?ack;

# Data Transfer:
IF
:: true -> W!dreq;     # Timeout and Request Data
     DO
     :: W?data -> skip  # Data Received
     :: W?data -> W!shutup  # Timeout and Shutdown
     OD
:: true -> W!shutup         # Timeout and Shutdown
FI;

# Disconnecting Phase:
W?shutup; W!quiet; W?dead
END
```

```
PROCESS (W)

# Connecting Phase:
M?init; M!ack;

# Data Transfer:
DO
:: M?dreq -> M!data         # M requests data
:: M?data -> skip      # M sends data
:: M?shutup -> M!shutup     # Shutdown
OD;

# Disconnecting Phase:
M?quiet; M!dead
END
```

The specification for process M is placed in a file called M, and the specification for process W is placed in a file called W. The command

```
prc M W
```

will result in the compilation of the protocol expressions. Prc creates a number of files in the user's working directory. None of the files created by prc need be of interest to the user (it is of interest to pan). In the unlikely event that some user is still interested in their contents, here is what they are used for. The files Mx and Wx give a first translation of the language description into an expression. Files Mx.n and Wx.n give the translation of messages within the expression into numbers, and files Mx.n.s abd Wx.n.s contain symbolic expressions which will be read by pan. The file MSGS contains a list of all messages exchanged in the protocol, together with the numbers assigned to them by prc. (Timeout messages are named TAU, skips are named NIL.)

After the compilation the analysis can be invoked with the command:

```
pan M W
```

The symbolic output of pan is written into a file named OUTPUT. The translated output is in a file named TRANSLATED. The syntax of the translated messages differs from the one prescribed by the specification language (it is chosen to simplify the working of the chart plotter, which is part of pan). Pan will print the number of suspect execution sequences traced on the standard output.

The only file generated by pan that might be of interest to the user will be the file CHARTS, which specifies all the design errors detected by pan.

If errors are detected, and the user agrees with their validity, the protocol specification can be amended and the analysis repeated.

If no errors are found, a final check can be done for residual messages with the command.

```
orphans
```

Orphans will read the file OUTPUT and trace any message that can be produced without being consumed. As the analysis of pan only traces design errors that show up in a single cycle of all processes through their part of the protocol, an error caused by a residual message can only show up in a second test where the residual messages are treated as pseudo initial messages.

Orphans will deliver a default initial string in the file INITIAL, which will be read by *pan* in a subsequent call. Note that the INITIAl file has to be removed or renamed if no initial string is wanted.

An analysis can only be considered complete if the command sequence:

```
prc M W; pan M W; orphans
```

returns the messages on the standard error output:

```
No errors found
No residuals
```

as it does for our example with M and W.

The following example shows how the user is notified of error conditions.

3. Second Example

Consider this protocol specificaion for a "dining philosphers' problem" with just two competitors. The specification consists of two fork processes (not 'unix' forks, but

```
PROCESS (Ph0)
IF
:: F1?fork -> F0?fork; F1!fork; F0!fork
:: F0?fork -> F1?fork; F0!fork; F1!fork
FI
END

PROCESS (Ph1)
IF
:: F0?fork -> F1?fork; F0!fork; F1!fork
:: F1?fork -> F0?fork; F1!fork; F0!fork
FI
END

PROCESS (F0)
IF
:: true -> Ph0!fork; Ph0?fork; Ph1!fork; Ph1?fork
:: true -> Ph1!fork; Ph1?fork; Ph0!fork; Ph0?fork
FI
END

PROCESS (F1)
IF
:: true -> Ph1!fork; Ph1?fork; Ph0!fork; Ph0?fork
:: true -> Ph0!fork; Ph0?fork; Ph1!fork; Ph1?fork
FI
END
```

Now the command sequence:

```
prc Ph[01] F[01]; pan Ph[01] F[01]
```

triggers the message:

```
2 errors
```

Inspecting the file CHARTS reveals that there is a deadlock when the two philosophers both succeed in obtaining one of the two forks, and start waiting for the second one.

The charts plotted have the following form:

```
May 16 23:46 1981   Page 1

MESSAGE     F0     F1     Ph0     Ph1
fork               ----------------->
fork        --------------->

MESSAGE     F0     F1     Ph0     Ph1
fork!       ---------------->>
fork!              ---------------->>
fork?                      +
fork?                               +
=======
```

The first column specifies the name of the message. The following columns are labeled with the process names (in this case there are four processes). An arrow from process F0 to process Ph0 means that the message listed in the first column is sent from F0 to Ph0. A single arrow (->) means that the message is sent and received. A double arrow (->>) means that the message is sent and appended to the receiver's mailbox. A plus (+) means that at this moment a message is received from the mailbox (i.e., the oldest message). Similarly (and redundantly), a symbol ! after a message name means that the message is not received (immediately), and a symbol ? means that an old message is received (from the mailbox).

4. Timeouts, Infinite Loops, and Skip Statements

There are two special symbols in the language: *true* and *skip*. The symbol *true*, when used as a guard (i.e., instead of a receive), models timeout. A timeout can only occur if the mailbox of the process for which it is specified as an option is *empty*.

Note that the select statement:

```
IF
:: pro1?msg1 -> pro2!msg2
:: true -> pro2!check
FI
```

will fail if the mailbox is not empty, and does *not* contain *msg1* from *pro1* in the first slot. If there is any message other than *msg1* in the first slot of the mailbox neither the timeout nor the reception of *msg1* is possible, and the sequence that leads into this state will be reported as an error. Timeouts will be represented in file OUTPUT with the number assigned to them by *prc*, but they are not plotted in the charts. Note that for timeouts a "worst case" assumption is made: since we abstract from absolute timings the timeout can occur at the worst possible moment.

The analyzer cannot deal with potentially infinite loops of message exchanges within a single cycle through the protocol. Note therefore that a statement of the form:

```
DO true -> pro3!msg OD
```

will lead to an overflow condition. The following error messages will appear on the standard error output:

```
Output buffer overflow
Check tail OUTPUT
Aborted
```

The last line of file OUTPUT will hold a string of numbers that identify the cause of the overflow. By looking up the numbers in file MSGS the loop on *true* and *msg* can be found and repaired.

The *skip* statement is useful in cycle or select statements to indicate that the reception of a message will trigger no response:

```
DO
:: pro1?msg1 -> skip
:: pro2?msg1 -> skip
OD;
pro3?msg1; pro3!thanks
```

Here the message *msg1* from process *pro1* and *pro2* are ignored, and only the reception of the same message from *pro3* will trigger the respinse *thanks*, and will abort the loop.

Skip statements are always possible (like send statements), but have no effect. Skips are not represented in OUTPUT or CHARTS.

Acknowledgements

I am grateful to Rob Pike for helping me debug the analyzer and wrote the protocol compiler *prc*. Without Rob the analyzer wouldn't be so easy to use as it is today. Al Aho, Doug McIlroy, Dave McQueen, and Ravi Sethi have made important contributions by criticizing my earlier attempts to develop a useful theory for protocol validation. I also owe a great deal to Lee McMahon and Greg Chesson for introducing me to the subtleties of protocol design.

5. Reference

G.J. Holzmann, *The analysis of message passing schemes*, and *An algebra for protocol validation*. Both in: Computing Science Technical Report No. 93, Bell Laboratories, 1981.

MH-11271-GJH-unix Gerard J. Holzmann

Att.
Appendices (pgs. 6-7)

NAME

pan, prc, orphans, unpan – protocol specification analyzer

SYNOPSIS

`prc` *file ...*
`pan` *file ...*
`orphans`
`unpan`

DESCRIPTION

These programs analyze message-passing protocols among a set of processes communicating via (unbounded) mailboxes. Each *file* specifies the behavior of one process.

prc compiles a set of specifications and prepares input files for *pan* . The protocol is specified in a language based on Hoare's formalism for communicating sequential processes. Messages are written:
processname!messagename
or
processname?messagename
The ! form means send the named message to the named process; the ? form means receive. Process names are identical to the *file* names. Message names can be reused between different pairs of processes.

In the following syntax for protocol specifications anything enclosed in [..] is optional, a star means 'zero or more times,' | means 'or.'

```
specification: PROCESS ( processname ) sequence END
sequence     : statement [;] *
statement    : select | cycle | send | guard
select       : IF [::] option [:: option ] * FI
cycle        : DO [::] option [:: option ] * OD
option       : guard -> sequence
guard        : true | skip | processname?messagename
send         : skip | processname!messagename
```

A comment extends from a # sign to the next newline. Select statements indicate a choice between one or more subsequences; one of the alternatives must be executed. Cycle statements indicate a series of one or more repetitions of the same option(s). A non-event (nil-sequence) is modeled by skip. A non-event is always executable (unlike timeout or a receive). A timeout is modeled by receiving the message true. A timeout can only be received when the process's mailbox is empty.

All possible execution sequences that violate the protocol specification are delivered in a file named CHARTS. The errors traced are errors that may occur within a single loop through the protocol expressions for all participating processes.

Given an OUTPUT file from *pan* , *orphans* places in the file INITIAL messages that can be left in mailboxes after one loop through the protocol expressions. A subsequent run of *pan* will trace the behavior of the next loop starting from this initial condition. (To start with a clean slate, delete INITIAL.) To check fully, run *orphans* and *pan* in alternation until the output stabilizes or disappears.

Each number in INITIAL represents one message; its meaning is given in file MSGS. The order of messages in each process's mailbox makes a difference; consider carefully whether the INITIAL file is adequate and make runs with different orders of messages if appropriate.

Unpan deletes all files created by *pan* , *prc* , and *orphans* .

FILES

Files created by *prc :*
MSGS: a list of messages and numbers assigned to them
SCRIPT: number assignment script
file x.n.s: an intermediate form of the protocol specified in *file ; pan* reads this file when *file* is specified as an argument
INVOCA: invocation file. First line gives initial messages (empty, unless the file INITIAL is present).

Files created by *pan :*
OUTPUT: a symbolic version of the execution sequences
TRANSLATED: a translation of suspect execution sequences from OUTPUT
CHARTS: contains charts with protocol violations

Programs and temporaries:
awk(1), sort(1), grep(1), sed(1)
/usr/bin/rob/qed
/usr/lib/pan/*
/usr/tmp/record??/*

DIAGNOSTICS

'Output overflow' – the protocol expressions may cause infinite looping. The output generated should suggest the cause. Overflow occurs much sooner on a PDP11 than on a VAX.

SEE ALSO

G.J. Holzmann, *Pan – a protocol specification analyzer* , TM 81-11271-5