The Readers-Writers Program

A. Pnueli

The Readers-Writers Program

IN the following example we consider a program with two types of clients: readers and writers. This is a generalization of the mutual exclusion problem, where the exclusion requirement can be stated as:

\[ N \geq N \cdot n_r + n_w \]

This can also be expressed as

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

The following assignments will contain (number of writers), where we assume (number of readers) and (number of writers), we assume variables

To model this problem, we assume variables

nr \geq 0 \land nw \geq 0.

The program will contain

nr + nw \leq N.

The following exclusion requirement can be stated as:

When one client writes, no other client may read or write. However, several clients may read at the same time.

In the following example we consider a program with two types of clients: readers and writers. This is a generalization of the mutual exclusion problem, where the exclusion requirement can be stated as:

nr \geq 0 \land nw \geq 0.

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

This can also be expressed as

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

The following assignments will contain (number of writers), where we assume (number of readers) and (number of writers), we assume variables

To model this problem, we assume variables

nr \geq 0 \land nw \geq 0.

The program will contain

nr + nw \leq N.

The following exclusion requirement can be stated as:

When one client writes, no other client may read or write. However, several clients may read at the same time.

In the following example we consider a program with two types of clients: readers and writers. This is a generalization of the mutual exclusion problem, where the exclusion requirement can be stated as:

nr \geq 0 \land nw \geq 0.

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

This can also be expressed as

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

The following assignments will contain (number of writers), where we assume (number of readers) and (number of writers), we assume variables

To model this problem, we assume variables

nr \geq 0 \land nw \geq 0.

The program will contain

nr + nw \leq N.

The following exclusion requirement can be stated as:

When one client writes, no other client may read or write. However, several clients may read at the same time.

In the following example we consider a program with two types of clients: readers and writers. This is a generalization of the mutual exclusion problem, where the exclusion requirement can be stated as:

nr \geq 0 \land nw \geq 0.

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

This can also be expressed as

The required exclusion property can be written as

\[ N \geq n_r \land n_w \land I \geq n_r \]

The following assignments will contain (number of writers), where we assume (number of readers) and (number of writers), we assume variables

To model this problem, we assume variables

nr \geq 0 \land nw \geq 0.

The program will contain

nr + nw \leq N.
A Program Ensuring Proper Exclusion

The following program ensures the proper exclusion between readers and writers:

```
A Program Ensuring Proper Exclusion
```

A. Pnueli

Let us examine the liveness properties of this program. Due to the general justice requirements associated with each assignment, we know that if there are clients wishing to read or write then, eventually one of them will be admitted to its critical section. However, the program allows a single reader to get in and out of its critical section, and the writer being scheduled always when the reader is critical.
To remedy the situation, we introduce two additional variables: 

- \( \text{md} \): pending writers 
- \( \text{prw} \): priority given to writers.

This leads to the following program:

\[
\begin{align*}
0 & \leq \text{md} \leq \text{md} + \text{nw} \\
0 & \leq \text{mu} \leq \text{mu} + \text{nw} \\
0 & \leq \text{mu} \leq \text{mu} + \text{nw} \\
0 & \leq \text{mu} \leq \text{mu} + \text{nw} \\
\end{align*}
\]

The program guarantees accessibility for a writer.
MODULE idle

! process idle : [0]

VAR
    pm : N..0
    nw : N..0
    taken : N..6

FILE Readers-Writers2.smv

MODULE main

DEFINE
    N := 5
    t := N - (nr + nw * N);

VAR nr : 0..N;
    nw : 0..N;
    taken : 0..6;
    pw : 0..N;
    prw : boolean;

T[0]: process idle;
T[1]: process start_read(nr, N, t, prw, taken);
T[2]: process end_read(nr, N, t, taken);
T[3]: process start_write(nw, N, t, pw, prw, taken);
T[4]: process end_write(nw, N, t, taken);
T[5]: process set_prw(prw, pw, taken);
T[6]: process add_writer(pw, N, taken);

Rigorous System Development, NYU, Fall, 2004
MODULE start_read(nr,N,t,prw,taken)
ASSIGN init(nr):=0;
next(taken):=1;
next(nr):=case
t>=1 & !prw: nr+1;
1: nr;
end case;

JUSTICE

MODULE end_read(nr,N,t,taken)
ASSIGN
next(taken):=2;
next(nr):=case
nr>0: nr-1;
1: nr;
end case;

JUSTICE

FILE readers-writers2.smv 2/4
MODULE start_write(nw,N,t,pw,prw,taken)
ASSIGN init(nw) := 0;
next(taken) := 3;
next(nw) := case
  nu > 0
  nw - I
  otherwise
end-assign
next(pw) := case
  t >= N & pw > 0
  pw - 1
  otherwise
end-assign
next(prw) := case
  t >= N & pw > 0
  0
  otherwise
end-assign
JUSTICE taken = 3

MODULE end_write(nw,N,t,taken)
ASSIGN next(taken) := 4;
next(nw) := case
  nw > 0
  nw - 1
  otherwise
end-assign
JUSTICE taken = 4

Rigorous System Development, NYU, Fall 2004
A. Pnueli

File: Readers-Writers2.smv

4/4

Lecture 6: Readers-Writers Program

Rigorous System Development, NYU, Fall, 2004
Load "unity-deductive.tlv"

Let exclusion := nw <= 1 & (nr = 0 | nw = 0);

Print "Verify Properties by Model Checking"

Print "Check Exclusion"

Call invariance(exclusion);

Print "Check Communal Accessibility"

Call Temp-Entail(1, nr > 0 | nw > 0);

Print "Check Accessibility for Writers"

Call Temp-Entail(1, nw > 0);

Print "Check Accessibility for Readers"

Call Temp-Entail(1, nr > 0);

-- Print "Check invariant(exclusion)"

-- Print "Check Exclusion"

Print "Verify properties by model checking"

Let exclusion := nw <= 1 & (nr = 0 | nw = 0);

Load "unity-deductive.tlv"

File Readers-Writers2.pf
A. Pnueli

CALL unitty-well-lex(start, goal, exclusion, h, del)

CALL unity-well-lex(start, goal, exclusion, h, del)

CALL unitty-well-lex(start, goal, exclusion, h, del)

Let pend := nw = 0 & pw > 0;
Let start := pw > 0;
Let goal := nw > 0;
Let h[0] := 6;
Let h[1] := 0;
Let h[2] := pend & prw & nr > 0;
Let h[3] := pend & prw & nr = 0;
Let h[4] := pend & prw & nr < 0;
Let h[6] := 0;

Let del[0] := 2;
Let del[1] := !prw;
prw : nr;
1 : 0;
end;

Print "In Check Accessibility For Writers"

CALL binary(exclusion);

Print "In Check Exclusion"

CALL binary(exclusion);

Print "In Perform Deductive Verification"
The implementation of

\tex\text{}
For (j in 1 \ldots nh) -- For each j, check Premise 2
Let counter2 := \left( \neg（inv \& helpful[j] \& T(tsn) \rightarrow \right.
\left.\text{next}(goal) | \text{next}（pend） \& \text{delgt}） \right) \end{equation}
If (counter2)
Print "Premise 2 false for j: ", j,
End -- For (j in 1 \ldots nh)
End -- well-lex;

For (j in 1 \ldots nh) -- For each j, check Premise 3
Let counter3 := \left( \neg（inv \& helpful[j] \& s[tsn].t[j] \rightarrow \right.
\left.\text{next}(goal) | \text{next}（pend） \& \text{delgt}） \right) \end{equation}
If (counter3)
Print "Premise 3 false for j: ", j,
End -- For (j in 1 \ldots nh)
End -- well-lex;
Add to the system the variable \( pr \) counting the number of pending readers. Add any necessary auxiliary variables and strengthen the conditions, so as to guarantee accessibility to readers, expressible as the property:

\[ (0 < \text{nr}) \land (0 < \text{wr}) \leftrightarrow (0 < \text{nr}) \]

as well as preserving accessibility for writers.

Verify all safety and liveness properties, using both model checking and deductive verificiation.