Transforming Coloured Petri Nets to Counter systems for Parametric Verification: 

A Stop-and-Wait Protocol Case study

Jonathan Billington    Guy Edward Gallasch    Laure Petrucci
Computer Systems Engineering Centre    LIPN, CNRS UMR 7030
University of South Australia    Université Paris 13
Mawson Lakes    Villetaneuse
AUSTRALIA    FRANCE
Motivation

- analysis of network protocols
- often modelled using (coloured) Petri nets
- state space explosion $\Rightarrow$ difficult to analyse
- parametric models
Motivation

- analysis of network protocols
- often modelled using (coloured) Petri nets
- state space explosion $\Rightarrow$ difficult to analyse
- parametric models

$\Rightarrow$ use:

- acceleration techniques to cope with the state space explosion problem
- FAST tool capabilities for parametric analysis
Outline

- **FAST tool**
  - Counter systems
  - Acceleration technique
  - Input/output of FAST

- **From Petri nets to counter systems**
  - General technique
  - Handling coloured Petri nets

- **Stop-and-wait Protocols**
  - Coloured Petri net model
  - Counter system model
  - Analysis
Counter Systems

- automata (control graph)
- extended with a finite set of unbounded integer variables
- transitions labelled with:
  - a guard expressed in Presburger arithmetics
  - an action expressed as an affine function over the integer variables
Example

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

call < 4 / call++

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

\[ \text{call} < 4 \] / \[ \text{call} \]++

\[ \text{call} > 0 \] / \[ \text{call} \]--

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Example

Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.5
Acceleration technique

Reachability Set

- often infinite → classical algorithm does not terminate
- ⇒ use of acceleration techniques
- semi-algorithm, often terminates
Acceleration technique

Reachability Set

- often infinite $\rightarrow$ classical algorithm does not terminate
- $\Rightarrow$ use of acceleration techniques
- semi-algorithm, often terminates
- symbolic representation of infinite sets
- acceleration: compute the effect of iterating a loop
Acceleration technique

Reachability Set

- often infinite $\rightarrow$ classical algorithm does not terminate
- $\Rightarrow$ use of acceleration techniques
- semi-algorithm, often terminates
- symbolic representation of infinite sets
- acceleration: compute the effect of iterating a loop

\[
x \geq 0 \quad / \quad x := x + 2
\]

\[
x := 0
\]
### Acceleration Technique

#### Reachability Set

- Often infinite → classical algorithm does not terminate
- ⇒ use of acceleration techniques
- Semi-algorithm, often terminates
- Symbolic representation of infinite sets
- Acceleration: compute the effect of iterating a loop

\[
x \geq 0 / x := x + 2
\]

Classical algorithm:

\[
Reach \supseteq \{0\}
\]
Acceleration technique

Reachability Set

- often infinite $\rightarrow$ classical algorithm does not terminate
- $\Rightarrow$ use of acceleration techniques
- semi-algorithm, often terminates
- symbolic representation of infinite sets
- acceleration: compute the effect of iterating a loop

$$x \geq 0 / x := x + 2$$

Classical algorithm:

$$\text{Reach } \supseteq \{0, 2\}$$
Acceleration technique

Reachability Set

- often infinite → classical algorithm does not terminate
- ⇒ use of acceleration techniques
- semi-algorithm, often terminates
- symbolic representation of infinite sets
- acceleration: compute the effect of iterating a loop

\[
x \geq 0 \land x := x + 2
\]

Classical algorithm:

\[
Reach \supseteq \{0, 2, \ldots\}
\]
Acceleration technique

Reachability Set

- often infinite $\rightarrow$ classical algorithm does not terminate
- $\Rightarrow$ use of acceleration techniques
- semi-algorithm, often terminates
- symbolic representation of infinite sets
- acceleration: compute the effect of iterating a loop

\[
x \geq 0 / x := x + 2
\]

\[
x := 0
\]

Acceleration: 
\[
Reach := 2 - N
\]
I/O of FAST

**model**: counter system

**strategy**: sequence of computations to check a safety property, described by a script language operating on:

- regions (sets of states)
- transitions
- booleans

and using operators to perform:

- sets and boolean operations
- forward/backward reachability
Petri Nets → Counter Systems

- a unique state
- one counter per place of the net
- one transition per transition of the net. Each transition:
  - loops onto the unique state
  - guard: enabling condition in the net
  - action: mimics the Petri net firing rule
model n1 {
  var p1, p2, p3;
  states dummy;
  transition t1 := {
    from := dummy;
    to := dummy;
    guard := p1>=1;
    action := p1'=p1-1, p2'=p2+2;
  };
  transition t2 := {
    from := dummy;
    to := dummy;
    guard := p3>=1 && p1=0;
    action := p1'=p1+4, p2'=0, p3'=p3-1;
  };
}
Handling CPNs

- often a single token (or none) in a place $\Rightarrow$ represent
colour set with an integer
- integers or enumerable types easy to map
- queues are more complex
Handling CPNs

- often a single token (or none) in a place ⇒ represent colour set with an integer
- integers or enumerable types easy to map
- queues are more complex
  - several types of messages but not simultaneously ⇒ count the number of messages. One counter per type
Handling CPNs

- often a single token (or none) in a place ⇒ represent colour set with an integer
- integers or enumerable types easy to map
- queues are more complex
  - several types of messages but not simultaneously ⇒ count the number of messages. One counter per type
  - at most two types of messages $a$ and $b$ at the same time in a FIFO queue, the queue being of the form $a*b^* ⇒ 4$ variables:
    1. $a\_type$ type of messages $a$
    2. $nb\_a\_type$ number of messages of type $a$
    3. $b\_type$ type of messages $b$
    4. $nb\_b\_type$ number of messages of type $b$
Transforming Coloured Petri Nets to Counter systems for Parametric Verification: – p.11
SWP counter system model

```plaintext
var SState, SSeqNb, Retrans, MaxRetrans, RSeqNb, RState, MaxSeqNb, 
MCOld, MCNew, NbMCOld, NbMCNew, ACOld, ACNew, NbACOld, NbACNew;
```
SWP counter system model

var SState, SSeqNb, Retrans, MaxRetrans, RSeqNb, RState, MaxSeqNb, MCOld, MCNew, NbMCOld, NbMCNew, ACOld, ACNew, NbACOld, NbACNew;

states dummy;
SWP counter system model

\begin{verbatim}
var SState, SSeqNb, Retrans, MaxRetrans, RSeqNb, RState, MaxSeqNb, MCOld, MCNew, NbMCOld, NbMCNew, ACOld, ACNew, NbACOld, NbACNew;
states dummy;
transition sendM1 := {
    from := dummy;
    to := dummy;
    guard := SState=1 && NbMCOld=0;
    action := SState'=0,
        MCNew'=SSeqNb, NbMCNew'=1, MCOld'=SSeqNb, NbMCOld'=1;}
\end{verbatim}
SWP counter system model

\begin{verbatim}
var SState, SSeqNb, Retrans, MaxRetrans, RSeqNb, RState, MaxSeqNb, MC0ld, MCNew, NbMC0ld, NbMCNew, AC0ld, ACNew, NbAC0ld, NbACNew;
states dummy;
transition sendM1 := {
    from := dummy;
    to := dummy;
    guard := SState=1 && NbMC0ld=0;
    action := SState'=0,
               MCNew'=SSeqNb, NbMCNew'=1, MC0ld'=SSeqNb, NbMC0ld'=1;}
transition sendM2 := {
    from := dummy;
    to := dummy;
    guard := SState=1 && !(NbMC0ld=0);
    action := SState'=0, MCNew'=SSeqNb, NbMCNew'=1;
}
\end{verbatim}

...
strategy analyseSWP {
    setMaxState(0);
    setMaxAcc(0);
strategy analyseSWP {  
  setMaxState(0);  
  setMaxAcc(0);  
  Transitions t := {sendM1, sendM2, ...};
strategy analyseSWP {
    setMaxState(0);
    setMaxAcc(0);

    Transitions t := {sendM1, sendM2, ...};

    Region init := {state=dummy && SState=1 && SSeqNb=0 &&
                  Retrans=0 && MC0ld=0 && MCNew=0 && NbMC0ld=0 && NbMCNew=0 &&
                  AC0ld=1 && ACNew=1 && NbAC0ld=0 && NbACNew=0 &&
                  RSeqNb=0 && RState=1 && MaxSeqNb=5};
strategy analyseSWP {
    setMaxState(0);
    setMaxAcc(0);

    Transitions t := {sendM1, sendM2, ...};

    Region init := {state=dummy && SState=1 && SSeqNb=0 &&
                    Retrans=0 && MC0ld=0 && MCNew=0 && NbMC0ld=0 && NbMCNew=0 &&
                    AC0ld=1 && ACNew=1 && NbAC0ld=0 && NbACNew=0 &&
                    RSeqNb=0 && RState=1 && MaxSeqNb=5};

    Region reach := post*(init, t, 2);
Properties

- **Consecutive sequence numbers** in messages buffer:

\[
\text{Region } \text{diffoldnewM} := \{(\text{MCold} = \text{MCNew}) \, || \, (\text{MCNew} = \text{MCold} + 1) \, || \\
(\text{MCold} = \text{MaxSeqNb} \, && \, \text{MCNew} = 0)\}\;
\]
Consecutive sequence numbers in messages buffer:

Region differoldnewM := 

\[(MC0ld=MC0New) \lor (MC0New=MC0ld+1) \lor (MC0ld=MaxSeqNb \land MC0New=0)\] ;

if (subSet(reach,differoldnewM))
    then print("Consecutive nb in message buffer OK");
    else print("Consecutive nb in message buffer NOK");
endif
Properties

- **Consecutive sequence numbers** in messages buffer:

  Region diffoldnewM := 
  \[(MC0ld=MCNew) \mid\mid (MCNew=MC0ld+1) \mid\mid (MC0ld=MaxSeqNb \&\& MCNew=0)\];

  if (subSet(reach,diffoldnewM))
    then print("Consecutive nb in message buffer OK");
    else print("Consecutive nb in message buffer NOK");
  endif

- **Consecutive sequence numbers** in acknowledgements buffer

- **Modelling assumptions** w.r.t. the queue are valid
- **Lowest upper bound in messages buffer**: \(2 \cdot \text{MaxRetrans} + 1\):

\[
\text{Region Mbound} := \{(\text{MCOld}=\text{MCNew} \land \land \\
\text{NbMCOld} \leq \text{MaxRetrans} + \text{MaxRetrans} + 1) \lor \\
(\neg (\text{MCOld}=\text{MCNew}) \land \land \\
\text{NbMCOld} + \text{NbMCNew} \leq \text{MaxRetrans} + \text{MaxRetrans} + 1)\};
\]
Properties

- **Lowest upper bound in messages buffer**: $2 \cdot \text{MaxRetrans} + 1$

  ```
  Region Mbound := 
  \{ (\text{MCOld} = \text{MCNew} \&\& \\
  \quad \text{NbMCOld} \leq \text{MaxRetrans} + \text{MaxRetrans} + 1) \mid \mid \\
  \quad (! (\text{MCOld} = \text{MCNew}) \&\& \\
  \quad \quad \text{NbMCOld} + \text{NbMCNew} \leq \text{MaxRetrans} + \text{MaxRetrans} + 1) \};
  ```

  ```
  if (\text{subSet}(\text{reach}, \text{Mbound}))
  then \text{print}("\text{Mbound OK}"");
  else \text{print}("\text{Mbound NOK}"");
  endif
  ```
Properties

- **Lowest upper bound in messages buffer**: $2 \cdot \text{MaxRetrans} + 1$

  ```
  Region Mbound := 
  \{ (MCOld=MCNew && 
         NbMCOld<=MaxRetrans+MaxRetrans+1) || 
     (! (MCOld=MCNew) &&
         NbMCOld+NbMCNew<=MaxRetrans+MaxRetrans+1) \};

  if (subSet(reach, Mbound))
  then print("Mbound OK");
  else print("Mbound NOK");
  endif
  ```

- **Lowest upper bound in acknowledgements buffer**: $2 \cdot \text{MaxRetrans} + 1$

- **Lowest upper bound in both buffers**: $2 \cdot \text{MaxRetrans} + 1$
Stop and Wait property needs a bit of instrumentation: add a variable $SR_{prop}$ recording the number of the last message sent +1. Update it when sending a message, reset it when receiving the message. Then check that it is not possible to send a message if the previous one has not been received:
Stop and Wait property needs a bit of instrumentation:
add a variable $SR_{prop}$ recording the number of the last message sent +1. Update it when sending a message, reset it when receiving the message. Then check that it is not possible to send a message if the previous one has not been received:

```c
if (isEmpty(reach && {SRprop>0 && SState=1}))
    then print("Send and then receive OK");
else print("Send and then receive NOK");
endif
```
Properties

- **Stop and Wait** property needs a bit of instrumentation: add a variable $SR_{prop}$ recording the number of the last message sent +1. Update it when sending a message, reset it when receiving the message. Then check that it is not possible to send a message if the previous one has not been received:

```java
if (isEmpty(reach && {SRprop>0 && SState=1}))
    then print("Send and then receive OK");
else print("Send and then receive NOK");
endif
```

- Hence **no loss** except eventually the last message when MaxRetrans is reached.
Properties

No duplication: check that there is no state such that the receiver is ready to accept a new message with a sequence number different from the last message sent:

```java
if (IsEmpty(reach && {SRprop=MC0ld+1 && RState=1 && NbMC0ld>0 && !(MC0ld=RSeqNb)}))
    then print("No duplication OK");
else print("No duplication NOK");
endif
```
In sequence delivery: check that it is not possible to receive an original message with a sequence number different from the most recently sent:

```plaintext
if (isEmpty(reach && {RState=1 && NbMC0ld>0 &&
        MCMin=RSeqNb && !(SRprop=RSeqNb+1)})
    then print("In sequence delivery OK");
    else print("In sequence delivery NOK");
endif
```
Properties

- Deadlocks as expected:
  - Retrans=MaxRetrans
  - Sender not ready to send a new message: SState=0
  - both buffers empty: MCOld=MCNew, ACOld=ACNew and NbMCOld=NbMCNew=NbACOld=NbACNew=0
Conclusion

- **parametric verification** of stop-and-wait protocols with lossy or lossless channels
- verification of **many properties**
- **translation** of some CPNs with queues into counter systems