# A Scalable Incomplete Test for Message Buffer Overflow in Promela Models

Stefan Leue, Richard Mayr, and Wei Wei

Department of Computer Science
Albert-Ludwigs-University Freiburg
Germany

# Motivation

- Promela models are bounded models.
  - The communication channels in a Promela model have fixed lengths.
- The Specification of buffer lengths causes two problems.
  - The specified lengths are too small.
  - The specified lengths are too large.

  How could we know that the specified buffer lengths are sufficiently large?
- Some solutions to buffer overflow detection:
  - Simulation.
  - Verification of the absence of buffer overflow as a safety property.

# An Incomplete Boundedness Test of CFSM-based Models

- We have developed a scalable incomplete boundedness test for CFSM models.
  - If we remove all the fixed buffer lengths from a Promela model, it becomes a CFSM model.

- The general idea of the incomplete test of CFSM-based models:
  - Buffer-boundedness in CFSM-based models is undecidable.
  - Certain aspects of a model is abstracted away to get an overapproximation.
  - The boundedness problem of the obtained overapproximation can be solved efficiently.
  - If the overapproximation is bounded, the model is bounded. Otherwise, don't know!

# The Incomplete Boundedness Test of Promela Models

- Abstraction of Promela Models
  - abstract from program code $\longrightarrow$ CFSM
  - abstract from order of messages $\longrightarrow$ CFSM with Effect Vectors
  - determine all simple cycles (only cyclic behaviours can cause unboundedness)
  - assume that every cycle is enabled and any combination is possible $\longrightarrow$ Independent Cycle System

- Boundedness Check
  - check if there is any combination whose combined effect is sending at least one message without consumption of messages.
  - Absence of such combination implies the boundedness of the model.

# An Example

```
proctype P{
   do
   :: CH?a -> CH!b; CH!c
   od
}

proctype Q{
   do
   :: CH?b -> CH!a
   :: CH?c -> CH!a
   od
}
```

Figure 1: A Promela Model

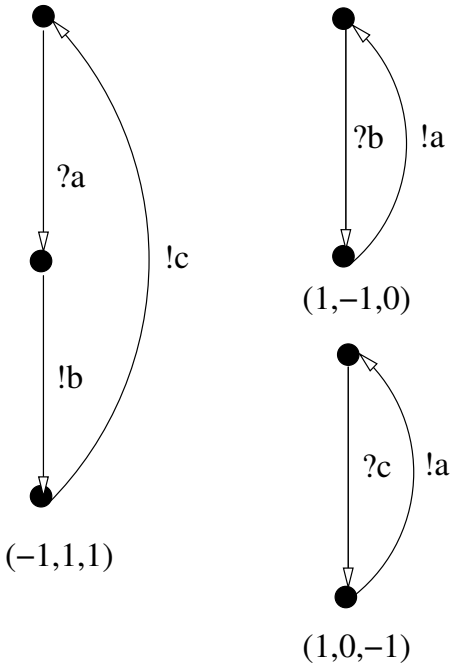



Figure 2: The Independent Cycle System

$$-x_1 + x_2 + x_3 \geq 0$$

$$x_1 - x_2 \geq 0$$

$$x_1 - x_3 \geq 0$$

$$x_1 > 0$$

Figure 3: The LP Problem

$x_1 = 1, x_2 = 1, x_3 = 1$

# Estimating Buffer Bounds

- Assume a path $P$ where the occupancy of a buffer reaches the maximum.
- $P$ can be decomposed into an acyclic part and a cyclic part.
- Encode every acyclic effect with the cycle system to get an optimization LP problem for computing the buffer bound estimate.
- The number of aggregate acyclic effects is exponential in the number of parallel processes.
- An overapproximating solution is to use an upper bound of all acyclic effects for each parallel process.

# Message Types

## Problem 1: How to abstract messages in the system?

- A message consists of several fields.
- Using all combinations of the field values to identify messages is awkward and UNNECESSARY.
- We discriminate between two messages only if they are treated differently in the system.

```
do
   :: C?5,x -> BRANCH1
   :: C?4,y -> BRANCH2
od
```

  - (5,2) and (5,3)
  - (5,2) and (4,3)

# Message Types

## Solution: Using Message Types

- A message type represents a group of messages that are treated exactly in the same way in the system, and are distinguished from any other message.

```
do
   :: C?5,x -> BRANCH1
   :: C?4,y -> BRANCH2
od
```

  - The constants in receive statements are critical to determining message types.
  - The message type (5,int) represents all the messages whose first field is 5.
  - The message type (4,int) represents all the messages whose first field is 4.
  - The message type (int,int) represents all other messages.
  - We obtain three message types instead of $2^{32} \times 2^{32}$ message types.

# Variables

Problem 2: How to model those send/receive statements where variables occur?

- Variables are used in send/receive statements.
  - as a field of a message: C!5,x
  - as an index of a channel array: C[x]!5,3
- The runtime value of a variable is unknown at compile time.
  - The send/receive statements with variables are modeled in a nondeterministic fashion.
  - Learning about the ranges of variables may benefit us in getting a finer overapproximation.
  - Determination of ranges can be achieved by tracking the runtime values of variables.
  - However, tracking variables is very difficult.

# Tracking Variables

Solution: Computing the overapproximations of the ranges

- We compute an overapproximation of the range for each variable through constant propagation.
  - When a constant is assigned to a variable, include the constant into the range of the variable.
  - When a runtime value of a variable $v_2$ is assigned to a variable $v_1$, propagate all possible runtime values of $v_2$ to the range of $v_1$.
  - When an expression, whose runtime value is unknown, is assigned to a variable, set the range of the variable to the domain of its type.

# Channel Assignments

## Problem 3: How to deal with channel assignments?

- A channel is a variable whose runtime value points to an actual message queue.
  - Each channel is initialized with a seperate queue.
  - The queue pointed to by a channel can be changed through assignments.

```
C1 = C2
```

- When two channels point to the same queue, one does not need to discriminate between messages exchanged in these two channels.

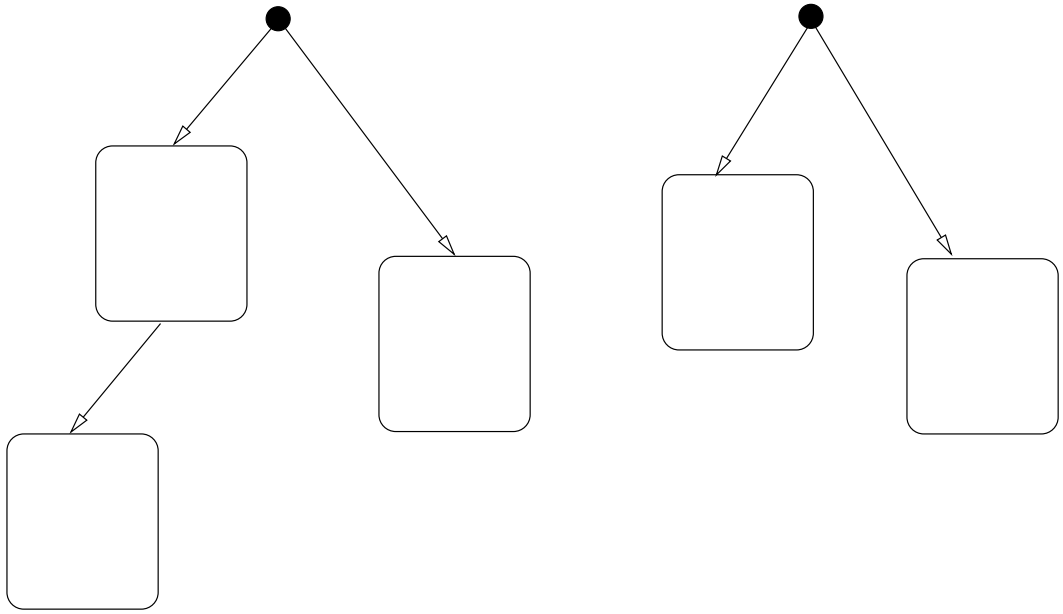
## Solution: Merging channels

- A coarse solution is that any two channels in a channel assignment are merged into one channel.
  - An assignment does not generally affect every part of the model.

# Channel Assignments

**Question:** How can we know which part of the system can be affected by some channel assignment?

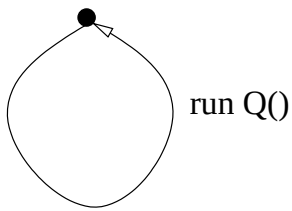
- The Directed Acyclic Graph (DAG) of Strongly Connected Components (SCCs)



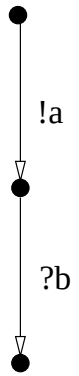
# Unbounded Process Creations

Problem 4: Do unbounded process creations affect our analysis?

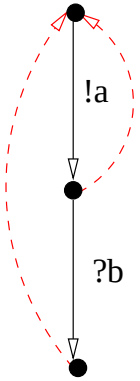
**Scenario 1: Process Creations in Local Loops**




The Process P




The Process Q

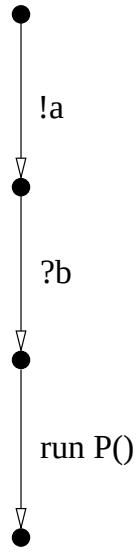



The Process Q with Replication Transitions

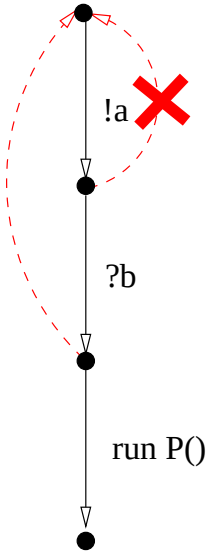
Solution: Overapproximate the unbounded number of acyclic paths in $Q$ by adding an auxillary backward transition (replication transition) to the initial state to form a cycle.

# Unbounded Process Creations

## Scenario 2: Self-Creations




The Process P

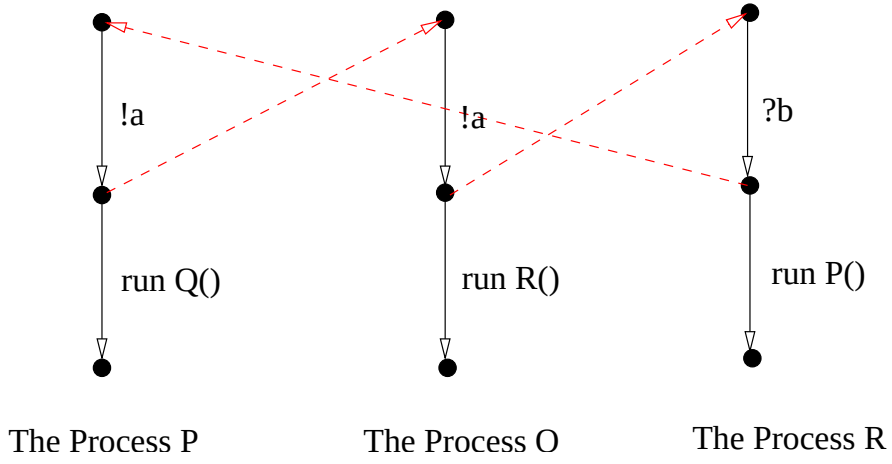



The Process P with Replication Transitions

Note that not every acyclic path can exert its effect alone to the system an unbounded number of times.

# Unbounded Process Creations

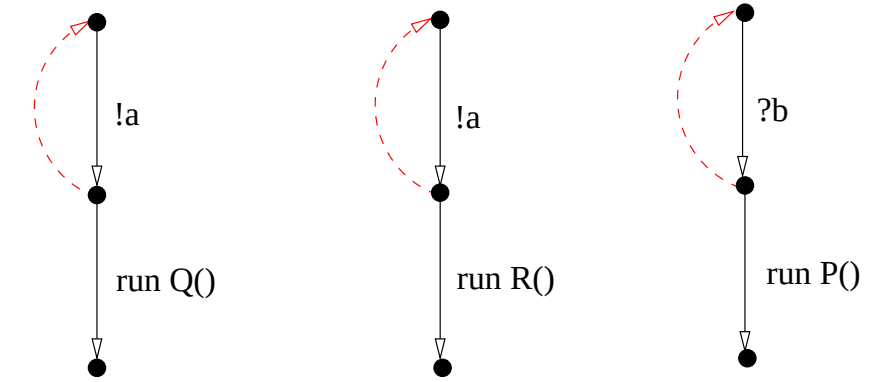
## Scenario 3: Mutual Creations




Note that the use of cross-processes replication transitions is not desirable because we lose the locality of the model in our analysis.

# Unbounded Process Creations

## Scenario 3: Mutual Creations




The Process P  The Process O  The Process R

Replacing cross-processes replication transitions with self-replication transitions is a safe overapproximation but coarser.

# Case study: A GIOP Model

- IBOC: IMCOS Boundedness Checker
- GIOP: The CORBA Inter-ORB Protocol
- A Promela implementation
  - 2 users, 1 GIOP client, 2 GIOP agents, and 2 servers: 7 running processes
  - 78 control-states, 100 transitions.
  - 9 communication buffers, 18 different message types.
- IBOC returned "UNKNOWN" for the GIOP model and suggested two classes of counterexamples.
- After we eliminated all the counterexamples, IBOC used only less than 3 seconds to prove the boundedness of the model and to compute all buffer bound estimates.

# Conclusion

- Our incomplete boundedness analysis for CFSM-based models can be used to test buffer overflow in Promela models.
- Main techniques: Abstraction, Overapproximation, Static analysis.
- Incomplete algorithms can be scalable.
- Future Works:
  - Refine the abstraction of Promela models.
  - Refine the computation of buffer bound estimates.
  - More to do with counterexample analysis and counterexample-guided refinement.