Counterexample-Based Refinement for a Boundedness Test for CFSM Languages

Wei Wei

Chair for Software Engineering
Department of Computer and Information Science
University of Konstanz
Germany

- It is a joint work with

  Stefan Leue.

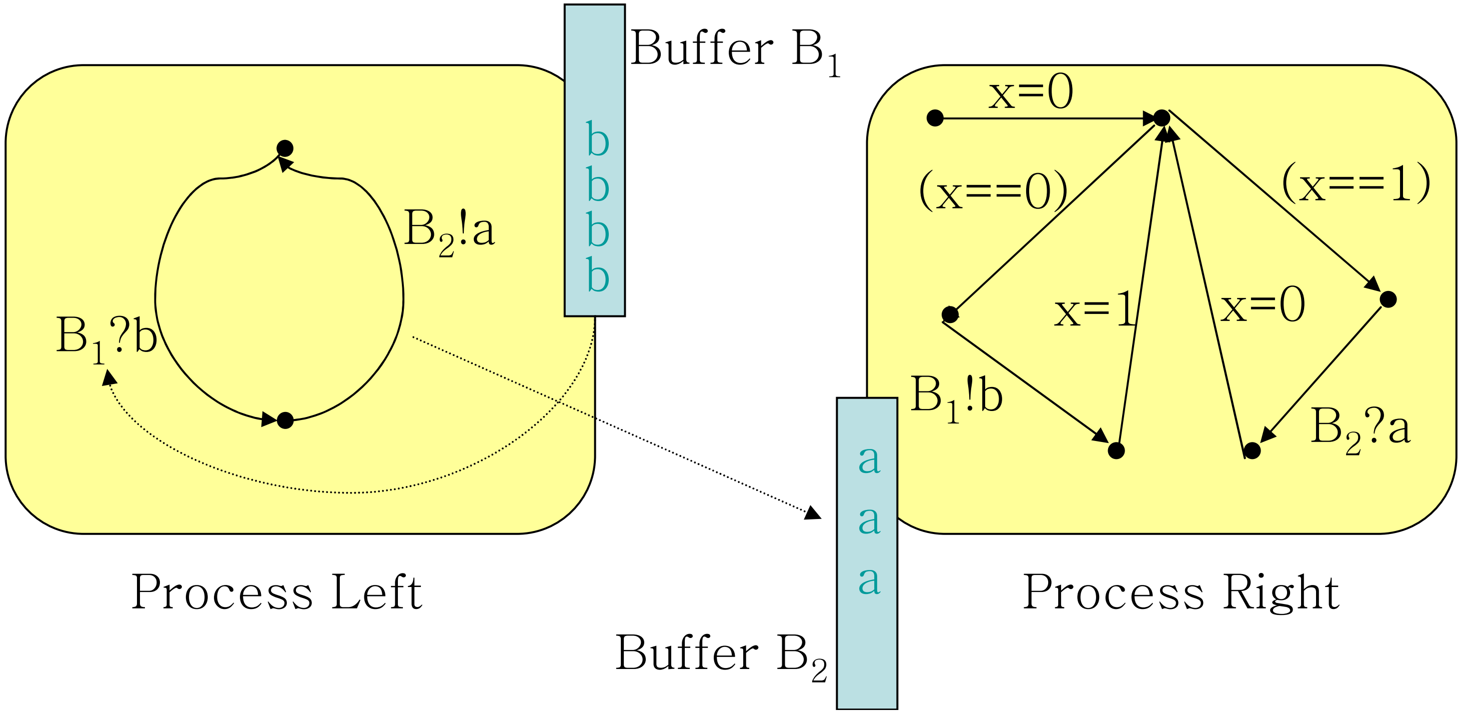
  - University of Konstanz
  - Email: *Stefan.Leue@inf.uni-konstanz.de*

Outline

- CFSMs and Buffer Boundedness
- Boundedness Test and Counterexamples
- Sources of Imprecision
- Cycle Code Analysis
- Graph Structure Analysis
- Complexity
- Experimental Results
- Conclusion and Future Work

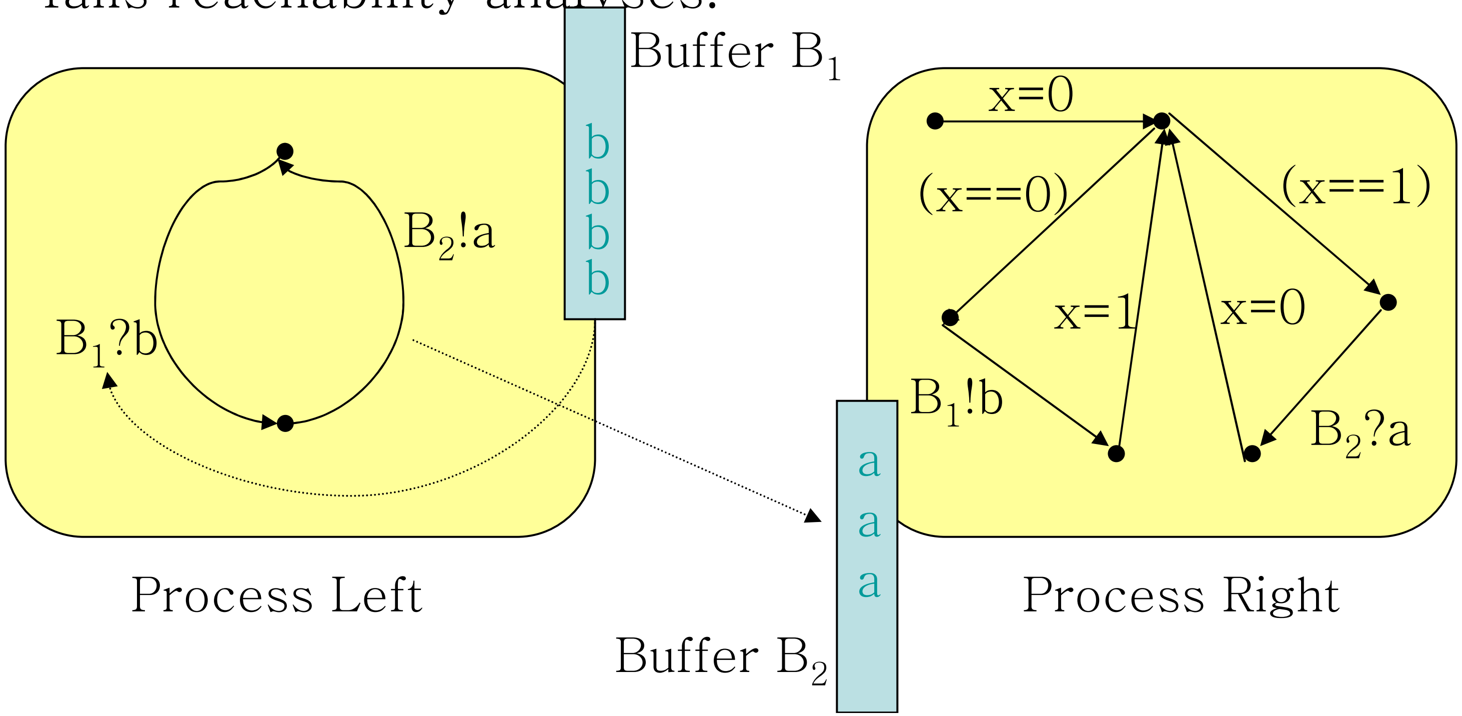
Communicating Finite State Machines

- CFSMs are to model discrete-state systems consisting of a number of processes that
  - execute concurrently,
  - and communicate with each other via asynchronous message exchanges.



Buffer Boundedness

- Buffers are assumed to have unbounded capacities.
  - the number of messages in a buffer may grow unboundedly.
- Unboundedness is not desired.
  - limited resources available.
  - fails reachability analyses.



An Incomplete Boundedness Test

- Buffer boundedness for CFSMs is undecidable.
- We developed an abstraction-based test.

  Stefan Leue, Richard Mayr, and Wei Wei: *A Scalable Incomplete Test for the Boundedness of UML RT Models*, Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2004.

  Stefan Leue, Richard Mayr, and Wei Wei: *A Scalable Incomplete Test for Message Buffer Overflow in Promela Models*, Proceedings of the 11th International SPIN Workshop on Model Checking Software SPIN 2004.

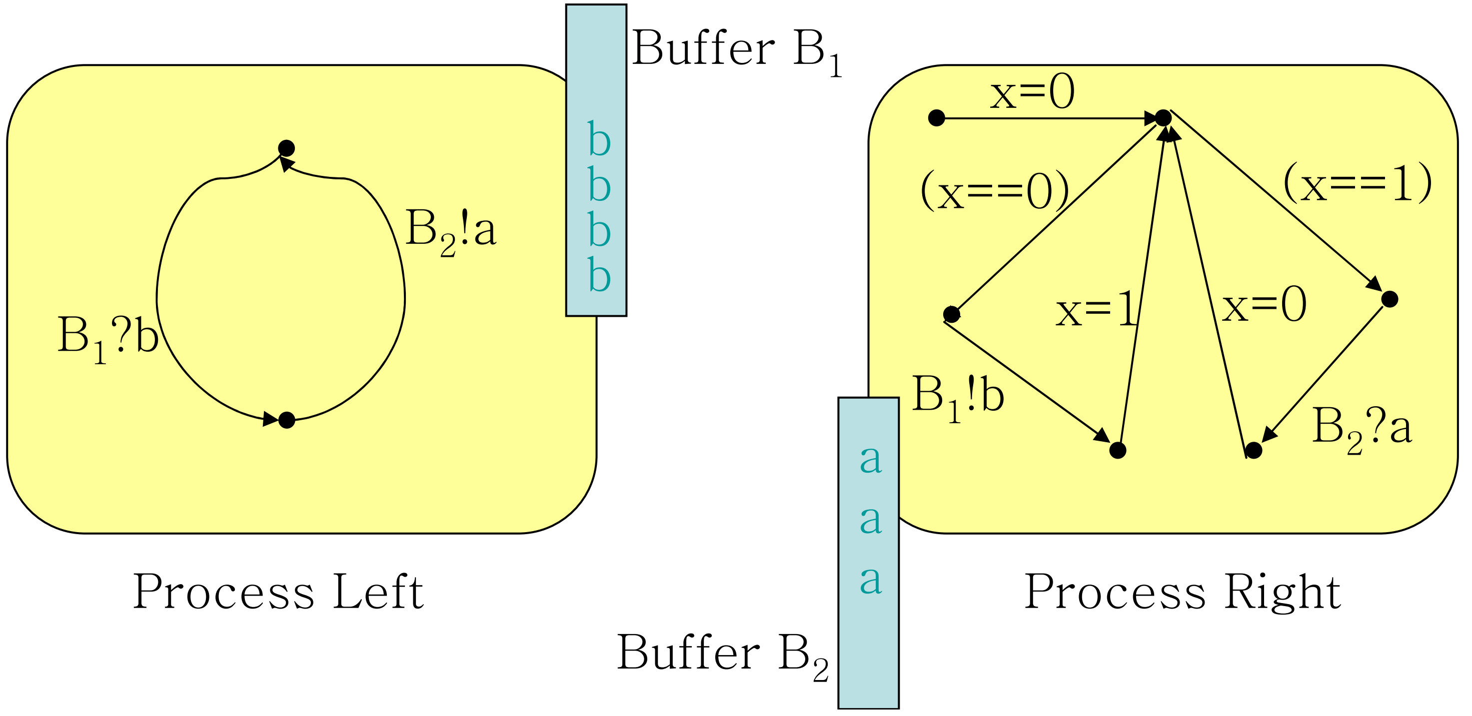
- The idea behind: only cyclic behavior may cause unboundedness.
  - concentrate on control flow cycles of state machines.

An Incomplete Boundedness Test

- What we abstract from:
  - program code
  - message orders
  - activation conditions of cycles
  - cycle dependencies

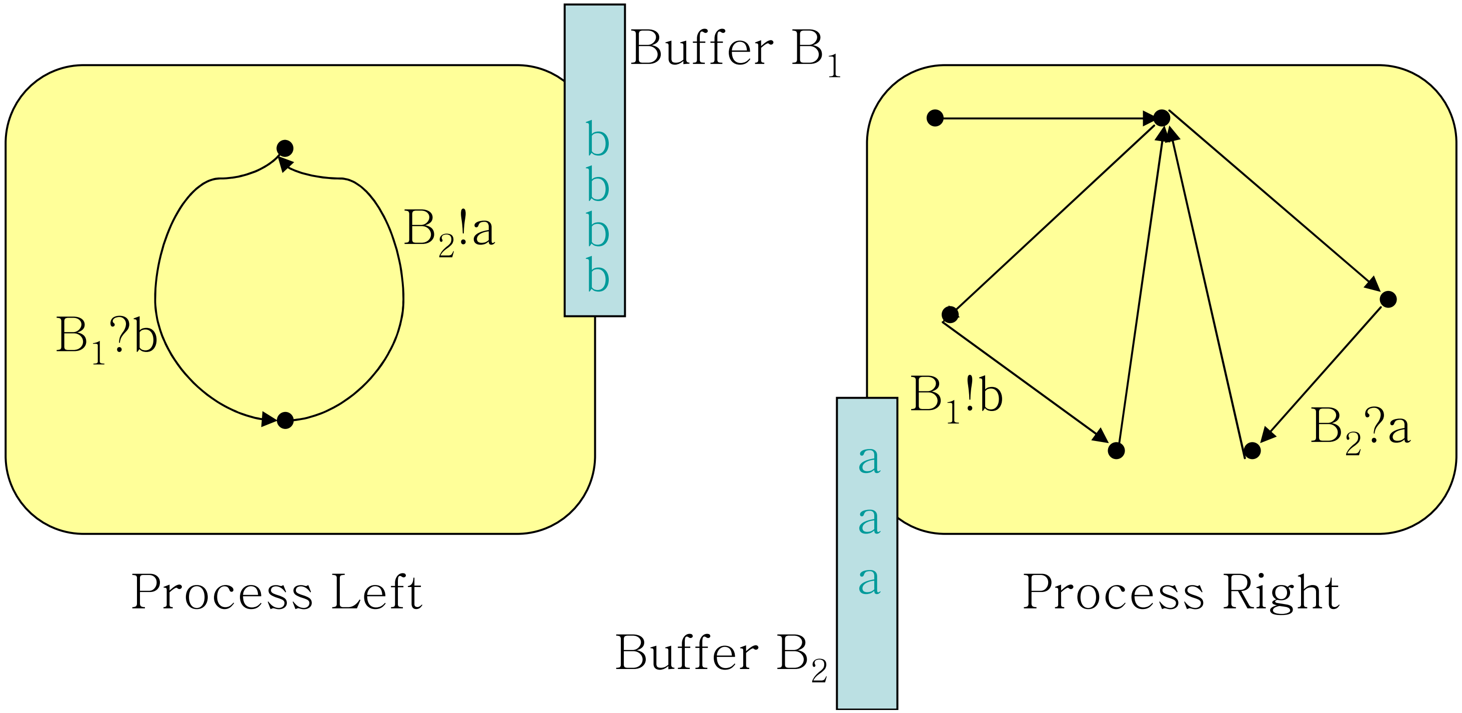
An Incomplete Boundedness Test

- What we abstract from:
  - program code
  - message orders
  - activation conditions of cycles
  - cycle dependencies



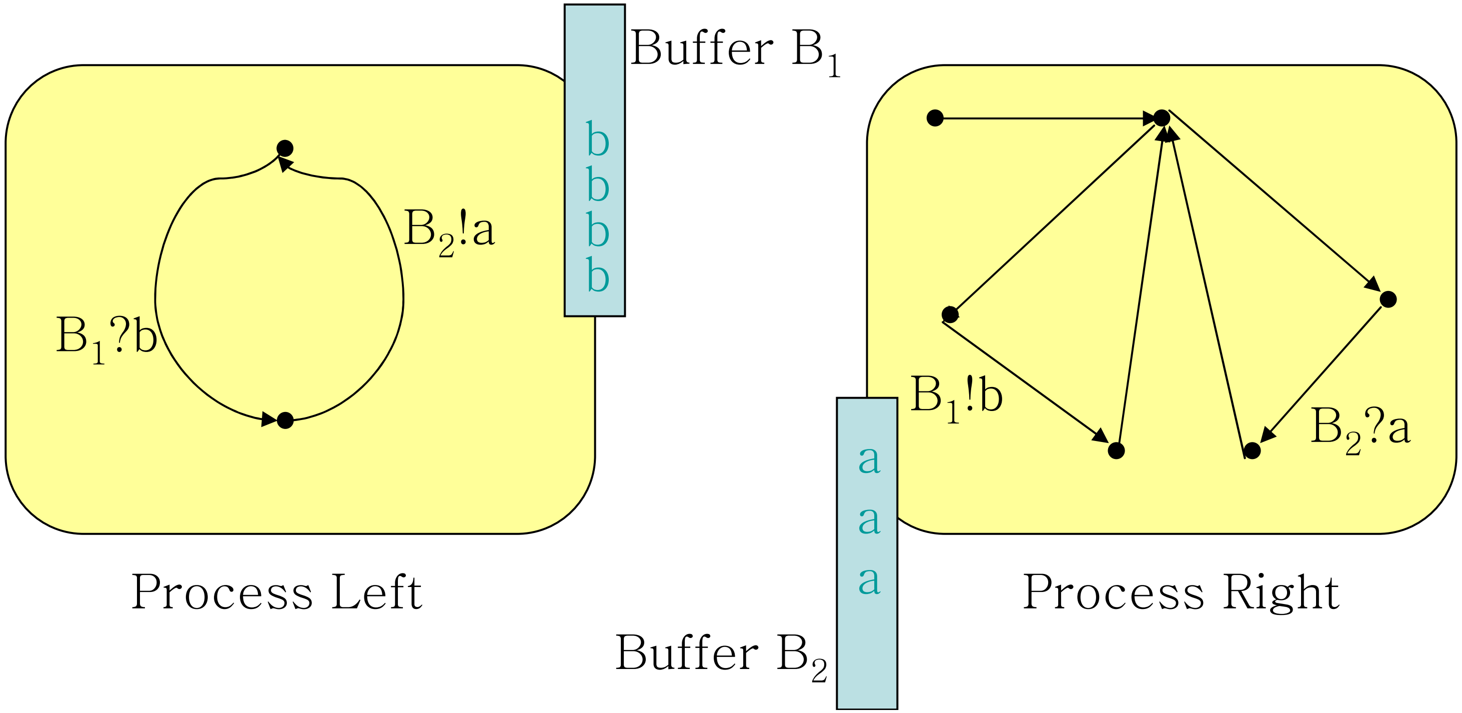
An Incomplete Boundedness Test

- What we abstract from:
  - program code → sequences of send or receive statements
  - message orders
  - activation conditions of cycles
  - cycle dependencies



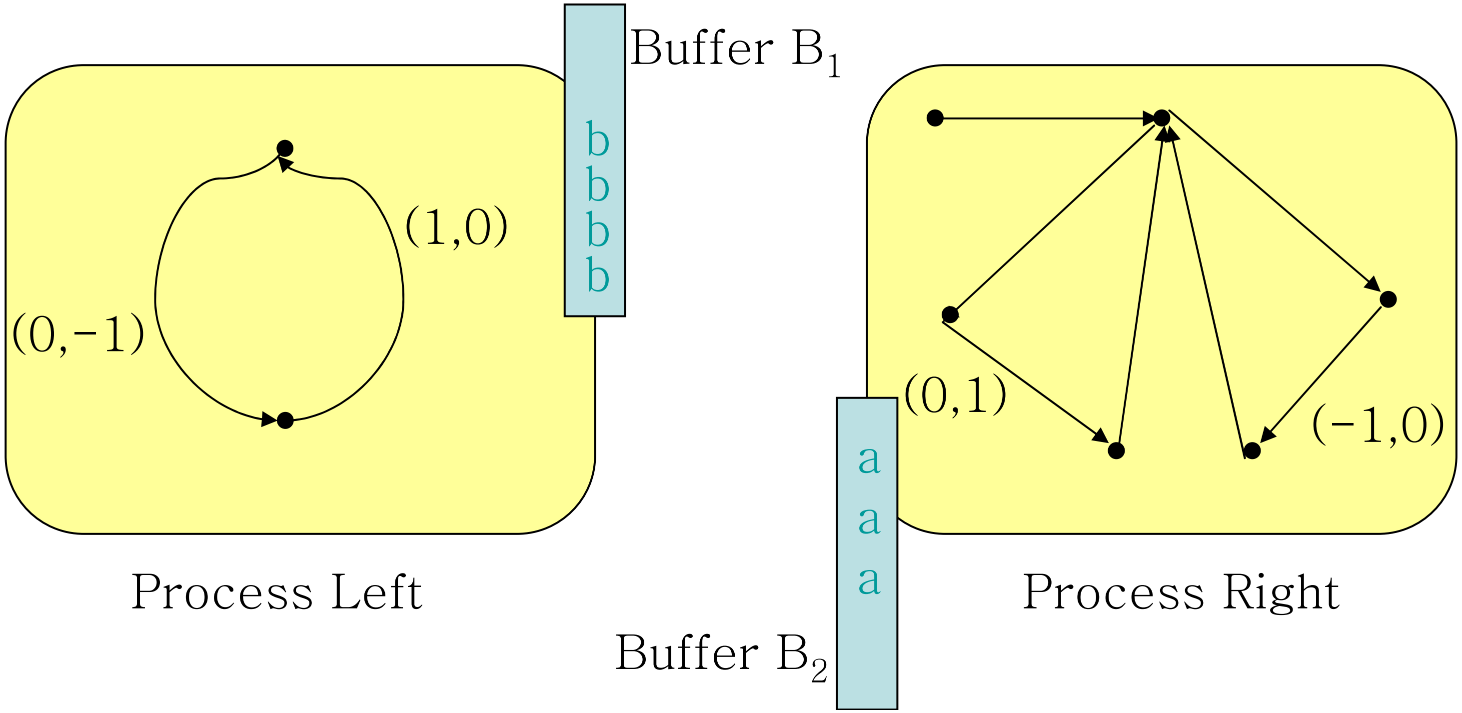
An Incomplete Boundedness Test

- What we abstract from:
  - program code
  - message orders (a,b,b,a) → (2,2)
  - activation conditions of cycles
  - cycle dependencies



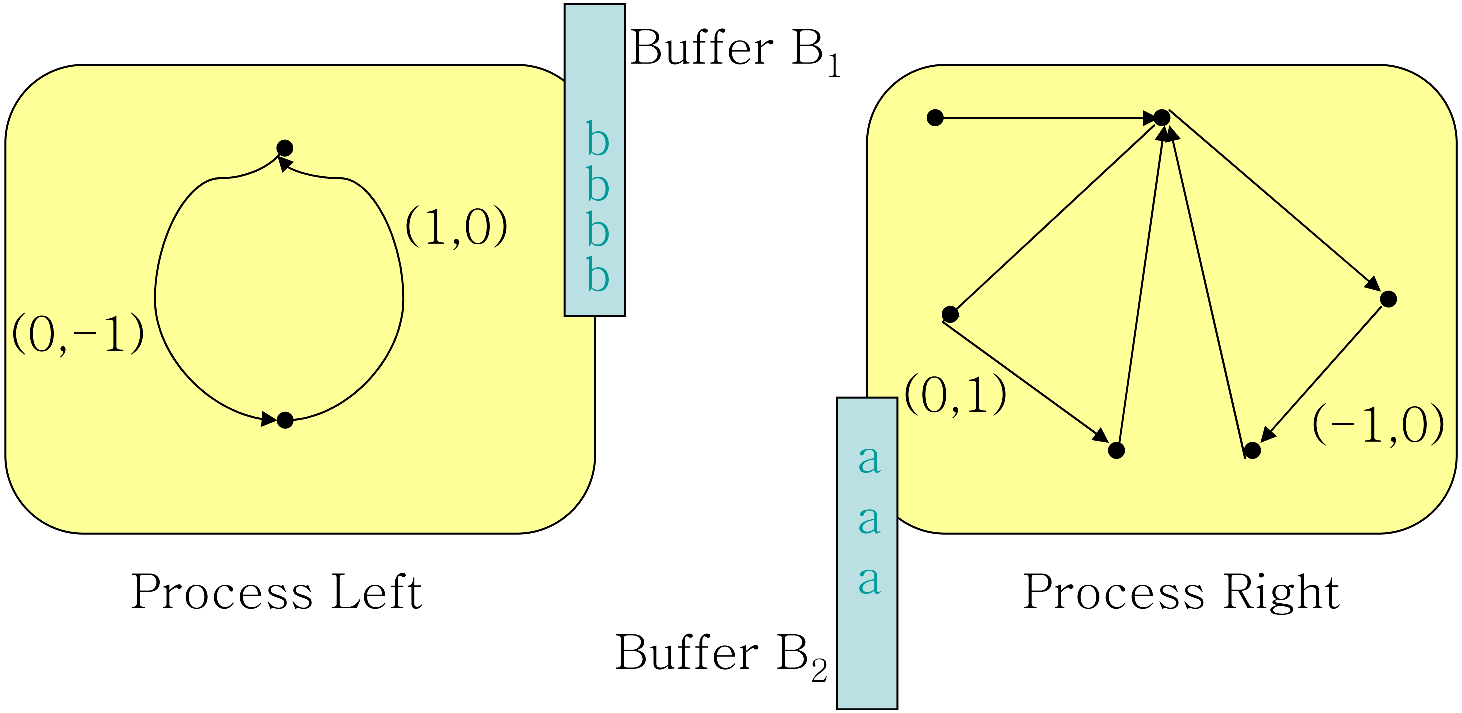
An Incomplete Boundedness Test

- What we abstract from:
  - program code
  - message orders: effect vector
  - activation conditions of cycles
  - cycle dependencies



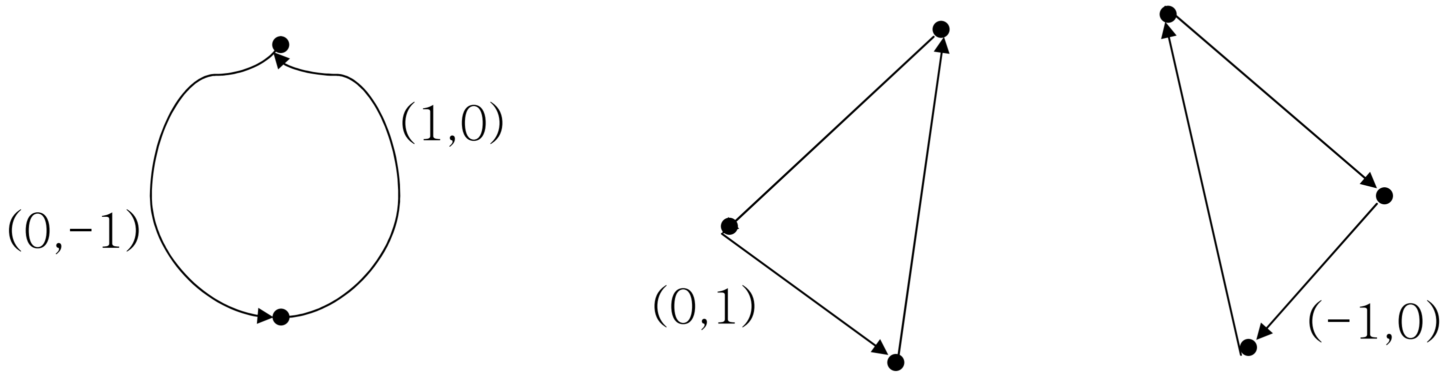
An Incomplete Boundedness Test

- What we abstract from:
  - program code
  - message orders
  - activation conditions of cycles
  - cycle dependencies



An Incomplete Boundedness Test

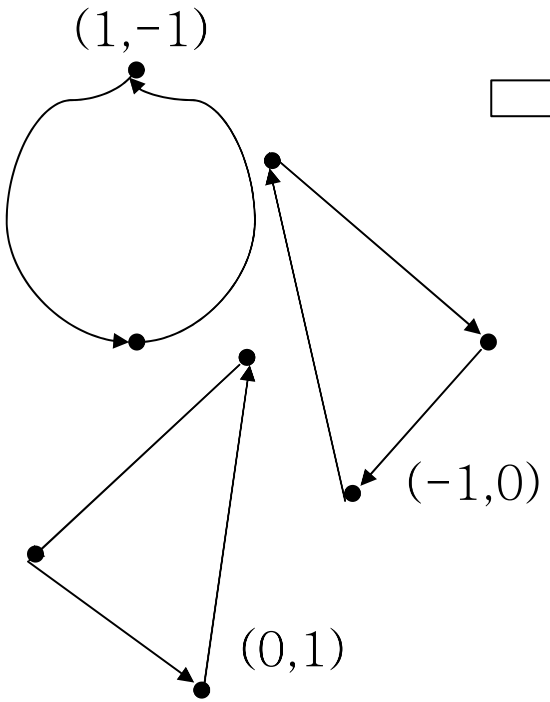
- The abstract model



(1,−1) (0,1) (−1,0)

An Incomplete Boundedness Test

- Use an integer linear programming (ILP) problem to check all the combinatory effects of cycles.



$$x_1\begin{pmatrix}1\\-1\end{pmatrix}+x_2\begin{pmatrix}0\\1\end{pmatrix}+x_3\begin{pmatrix}-1\\0\end{pmatrix}>\begin{pmatrix}0\\0\end{pmatrix}$$

- Any particular linear combination, whose combinatory effect has only nonnegative components and at least one positive component, indicates unboundedness of the abstract model.
- No such combination proves boundedness (of the concrete model.)

An Incomplete Boundedness Test

- Any particular linear combination, whose combinatory effect has only nonnegative components and at least one positive component, indicates unboundedness of the abstract model.

$$x_1\begin{pmatrix}1\\-1\end{pmatrix}+x_2\begin{pmatrix}0\\1\end{pmatrix}+x_3\begin{pmatrix}-1\\0\end{pmatrix}>\begin{pmatrix}0\\0\end{pmatrix}$$

A solution: $x_1 = 0$; $x_2 = 1$; $x_3 = 0$

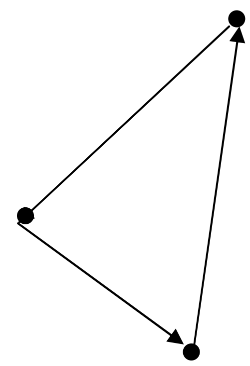
Counterexamples

- Counterexamples are sets of cycles.
  - only cycles in a counterexample are executed an infinite number of times.
- A counterexample is constructed from a particular solution to the boundedness determining ILP problem.
  - consists of all the cycles whose corresponding variable receives a non-zero value in the solution.

Counterexamples

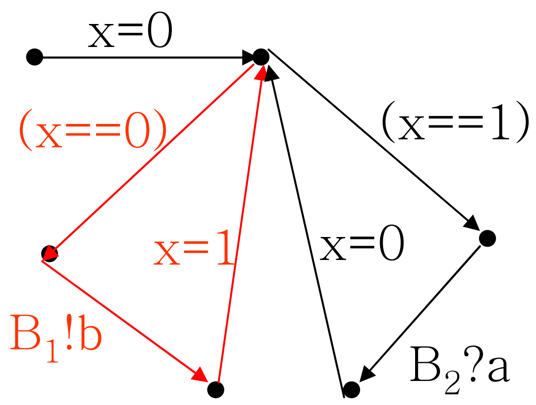
- A counterexample is constructed from a particular solution to the boundedness determining ILP problem.
  - consists of all the cycles whose corresponding variable receives non-zero value in the solution.

A solution: $x_1 = 0$; $x_2 = 1$; $x_3 = 0$ ⇨



(0,1)

Counterexample Spuriousness



- The left cycle cannot be repeated without executions of the right cycle.
- The counterexample constructed from the solution $x_1 = 0$; $x_2 = 1$; $x_3 = 0$ is spurious.

Sources of Imprecision

- What we have abstracted from:
  - program code
    - cycle conditions on variables are abstracted away.
  - message orders
    - not all the messages in a buffer may be available for trigger a transition.
  - activation conditions of cycles
    - a cycle may not be reachable from the initial configuration of the concrete model (no enough messages).
  - cycle dependencies
    - executions of cycles may depend on each other.

Sources of Imprecision

- What we have abstracted from:
  - program code
    - loop conditions on variables are abstracted away.
  - message orders
    - not all the messages in a buffer may be available for trigger a transition.
  - activation conditions of cycles
    - a cycle may not be reachable from the initial configuration of the concrete model (no enough messages).
  - cycle dependencies
    - executions of cycles may depend on each other.

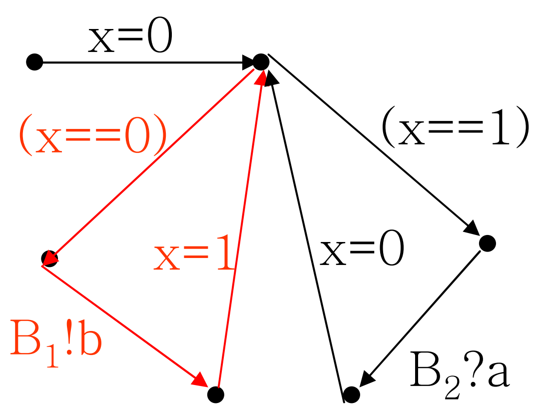
Sources of Imprecision

- We consider the following types of missing detail of concrete models:
  - cycle dependencies imposed by cycle conditions on variables.
    - locally modified variables → local dependencies.
    - integer variables.
    - linear conditions and linear assignments.
  - cycle dependencies imposed by control flow graph structures.
- We determine these two types of cycle dependencies.
  - used to determine spuriousness for counterexamples.
  - used to refine abstract models.

Cycle Code Analysis

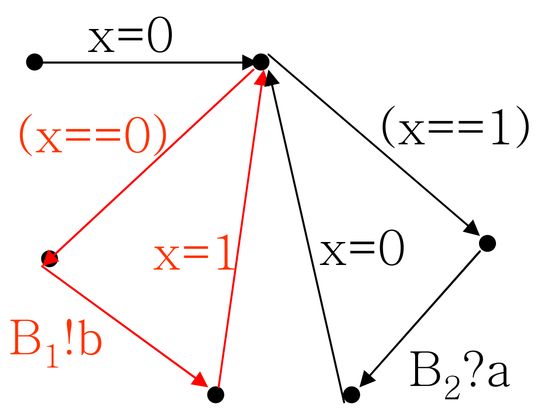
- The executability of a cycle is determined by all the condition statements in the cycle code.
- We check, for each individual condition statement (B), the constraint that it addes on cycle executions.
  - the maximal number of times max_B that (B) can be executed while the variables in the condition B are modified only within the cycle.
    - the cycle can be repeated without interruption no more than max_B times.
    - every max_B times that the cycle is repeated, some other cycles have to be executed at least once.

Cycle Code Analysis



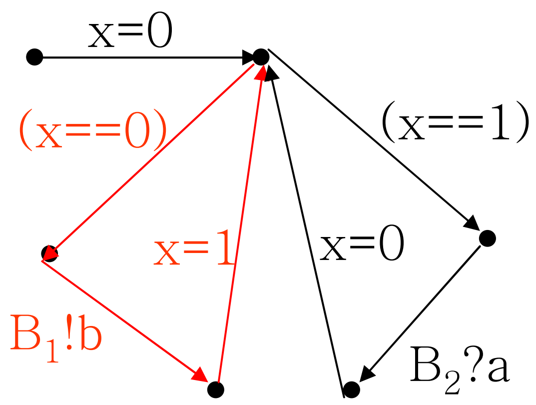
- Neighboring cycles.

Cycle Code Analysis



- Neighboring cycles.
- Supplementary cycles with respect to the condition B.
  - modify some variables in B to render B to be satisfied again.

Cycle Code Analysis



- Neighboring cycles.
- Supplementary cycles with respect to the condition *B*.
    - modify some variables in *B* to render *B* to be satisfied again.
- The right cycle is both a neighboring cycle and a supplementary cycle with respect to *x==0*.

Determining max_B

- It is generally impossible to determine max_B.

B -- max_B

Determining max_B

- It is generally impossible to determine max_B.

$$
\begin{tikzcd}
B \arrow[r, dashed, no head] \arrow[d] & max_B \\
d_1 \lor d_2 \lor \dots \lor d_n \arrow[d] & \\
l_1 \land l_2 \land \dots \land l_m \arrow[d] & \\
a_1 x_1 + \dots + a_k x_k \geq b &
\end{tikzcd}
$$

(control expression)

Determining max_B

- It is generally impossible to determine max_B.

$$B \text{ - - - - - - - - - - } max_B$$

$$d_1 \vee d_2 \vee \ldots \vee d_n \text{ - - - - - - - - - - } max_{B,d_1}, \ldots, max_{B,d_n}$$

$$l_1 \wedge l_2 \wedge \ldots \wedge l_m$$

$$a_1 x_1 + \ldots + a_k x_k \geq b$$

(control expression)

Determining max_B

- It is generally impossible to determine max_B.

B -- max_B

overapproximated by

$d_1 \vee d_2 \vee \ldots \vee d_n$ ------------------------ $max_{B,d1} + \ldots + max_{B,dn}$

$l_1 \wedge l_2 \wedge \ldots \wedge l_m$

$a_1 x_1 + \ldots + a_k x_k \geq b$

(control expression)

Determining max_B

- It is generally impossible to determine max_B.

$$
\begin{tikzcd}
B \arrow[d] \arrow[r, dash, dashed] & max_B \\
d_1 \vee d_2 \vee \ldots \vee d_n \arrow[d] \arrow[r, dash, dashed] & max_{B,d1} + \ldots + max_{B,dn} \arrow[u, "\text{overapproximated by}"'] \\
l_1 \wedge l_2 \wedge \ldots \wedge l_m \arrow[d] \arrow[r, dash, dashed] & max_{B,d1,l1}, \ldots max_{B,d1,lm} \\
a_1 x_1 + \ldots + a_k x_k \geq b &
\end{tikzcd}
$$

(control expression)

Determining max_B

- It is generally impossible to determine max_B.

B -- max_B

overapproximated by

$d_1 \vee d_2 \vee \ldots \vee d_n$ ---------------------------- $max_{B,d1} + \ldots + max_{B,\, dn}$

is

$l_1 \wedge l_2 \wedge \ldots \wedge l_m$ ---------------------------- $min\{\, max_{B,\, d1,\, l1}, \ldots max_{B,\, d1,\, lm} \}$

$a_1 x_1 + \ldots + a_k x_k \geq b$

(control expression)

Determining max_B

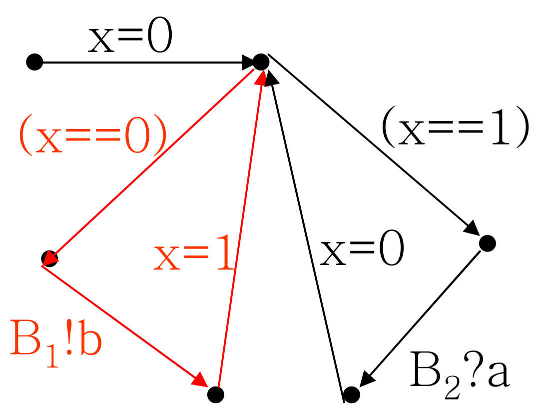
- Compute $max_{B,\ d1,\ l1}$

  $a_1\ x_1 + \ldots + a_k\ x_k \geq b$

  - We can only determine $max_{B,\ d1,\ l1}$ if the value of the control expression $a_1\ x_1 + \ldots + a_k\ x_k$ is always decreased.
    - step values of the control expression are always negative.
    - determine the initial values of the control expression.
    - determine the maximal step value of the control expression.
    - $max_{B,\ d1,\ l1}$ is bounded by

      $max\{\ 1, \lceil (maximal\_initial\_value - b) / - maximal\_step\_value \rceil \}$
  - Otherwise, we set $max_{B,\ d1,\ l1}$ to be ∞.

Determining max_B



$(x \geq 0 \wedge -x \geq 0)$

control expression: $-x$
initial value: 0
maximal step value: -1
$max_{-x \geq 0}$: 1

control expression: x
initial value: 0
maximal step value: 1
$max_{x \geq 0}$: ∞

max: $x' - x$;

$x = 0$;
$x' = 1$;

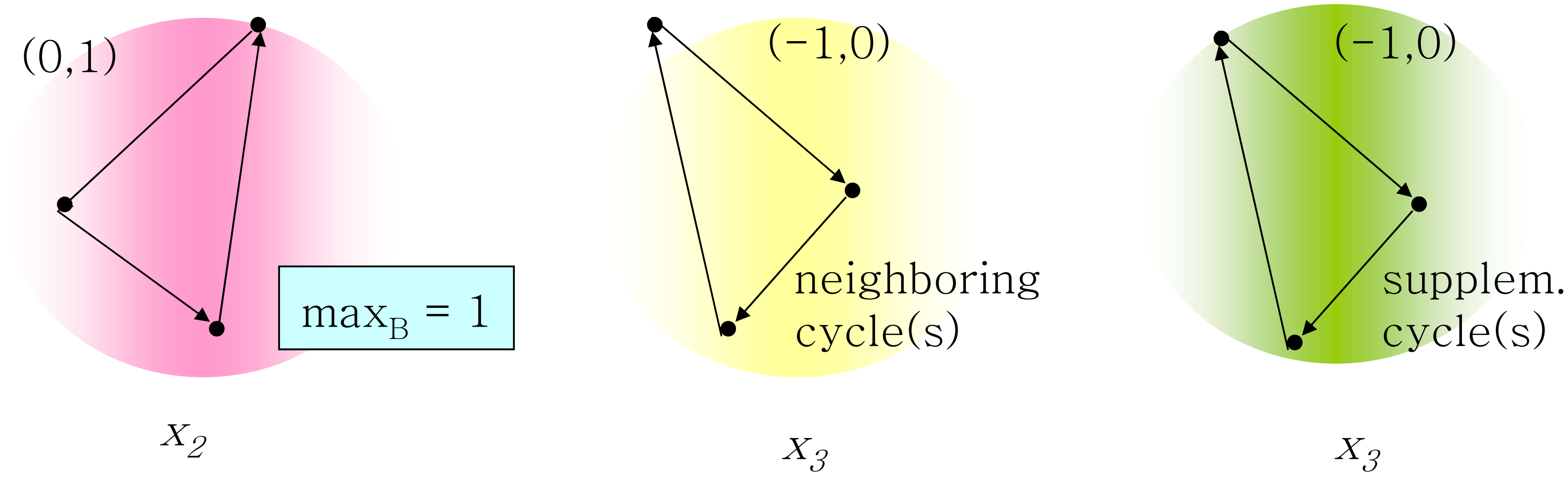
Determining Neighboring and Supplementary Cycles

- Neighboring cycles are easy to collect.
- It is generally impossible to determine the exact set of supplementary cycles.
  - overapproximation: a cycle is regarded as supplementary if it modifies some variables in the considered condition.
  - a finer approach: exclude all the cycles whose executions increase the value of each control expression in the condition.
    - much more expensive, involving code analysis of all the cycles that modify some variables in the condition.

Determining Spuriousness

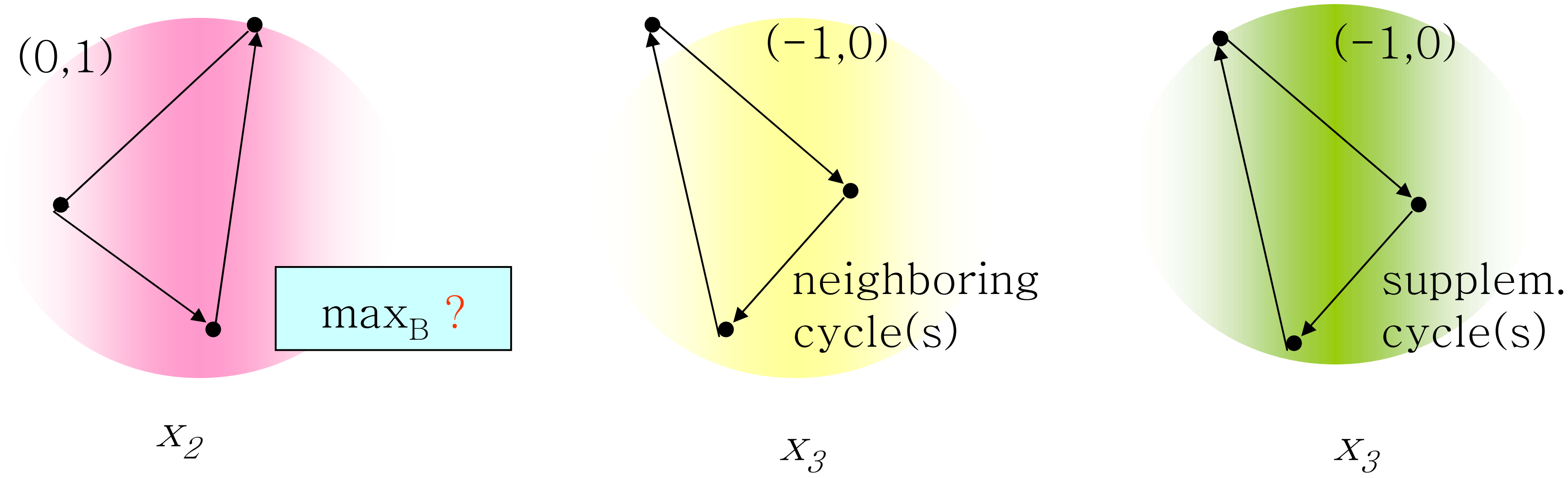
- Every $\max_B$ times that the cycle is executed,
  - at least one neighboring cycle must be executed.
  - at least one supplementary cycle with respect to B must be executed.
- A counterexample is spurious if one of its member cycle violates the above property.

Refinement



- Every $\max_B$ times that the left cycle is executed,
  - at least one neighboring cycle must be executed
    $x_2 \leq \max_B x_3 \rightarrow x_2 \leq x_3$
  - at least one supplementary cycle must be executed
    $x_2 \leq \max_B x_3 \rightarrow x_2 \leq x_3$

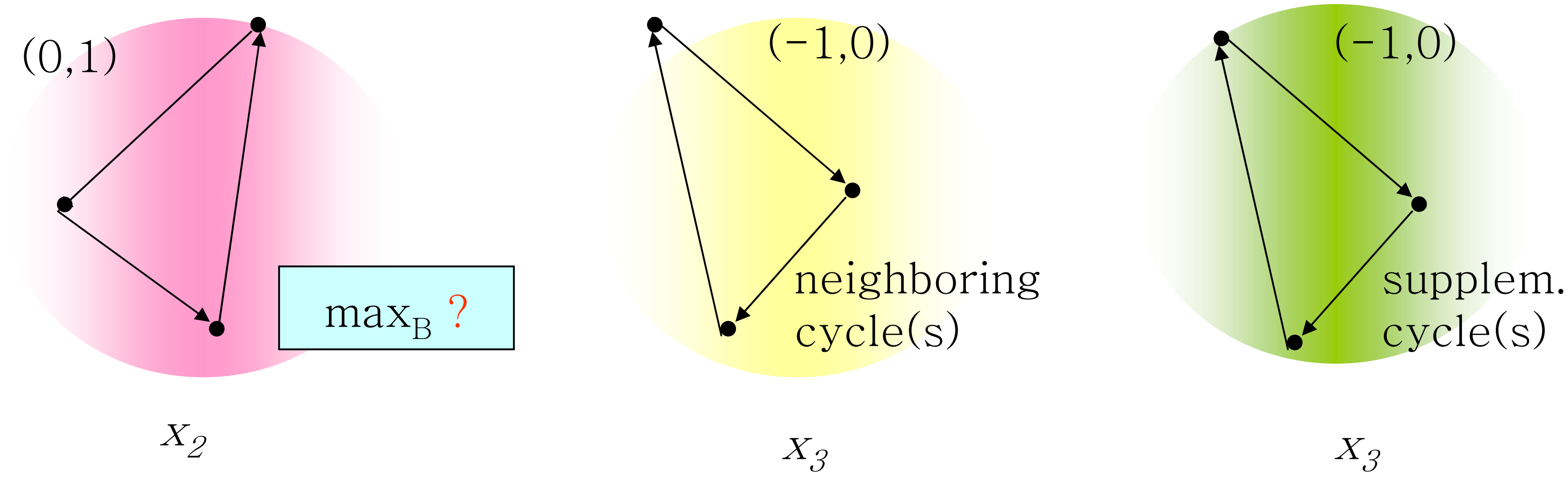
Refinement without max_B



- Two alternatives:
  - the left cycle is not executed infinitely often.

    $x_2 = 0$

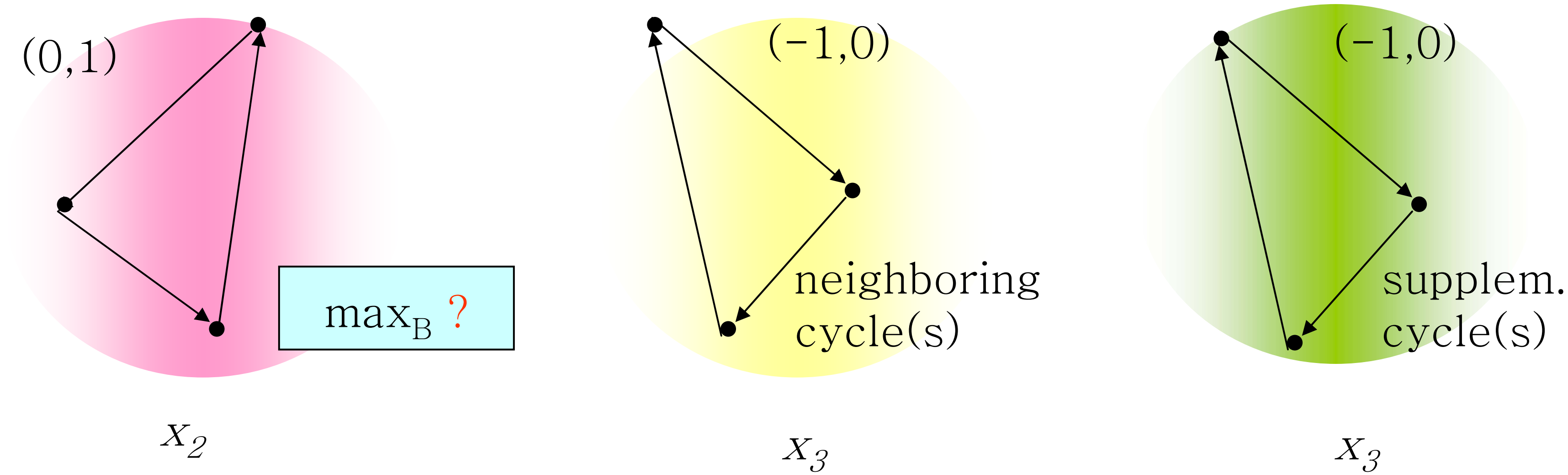
Refinement without max_B

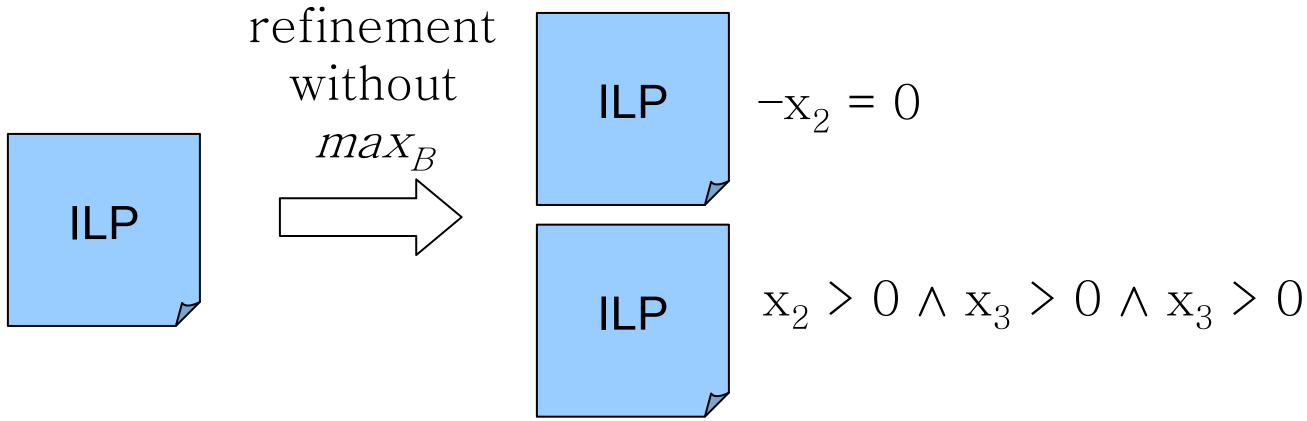


- Two alternatives:
  - the left cycle is executed infinitely often, then at least one of the neighboring cycles and at least one of the supplementary cycles must be also executed infinitely often.

    $x_2 > 0 \wedge x_3 > 0 \wedge x_3 > 0$

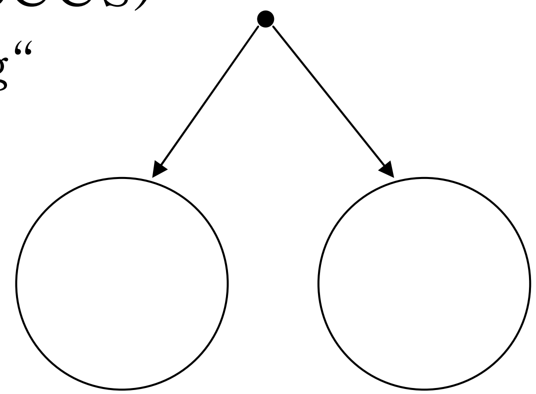
Refinement without max_B



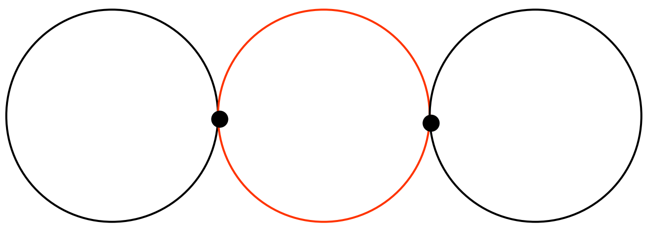


Graph Structure Analysis

- Strongly connected components (SCCs)
  - cycles in different SCCs are „repelling“ each other.

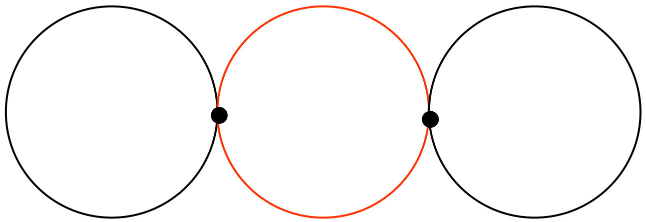


- Cycles that do not share common states need others to bridge them.



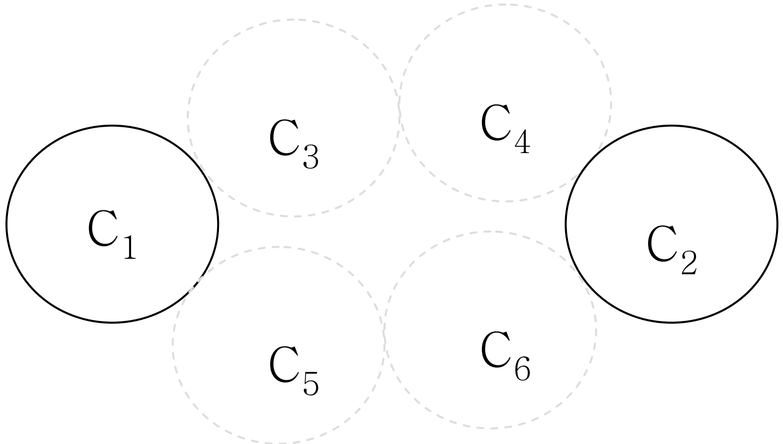
Self-Connected Cycle Set

- A set of cycles in the same process is self-connected if any two cycles in the set are reachable from each other by traversing through only the cycles in the set.



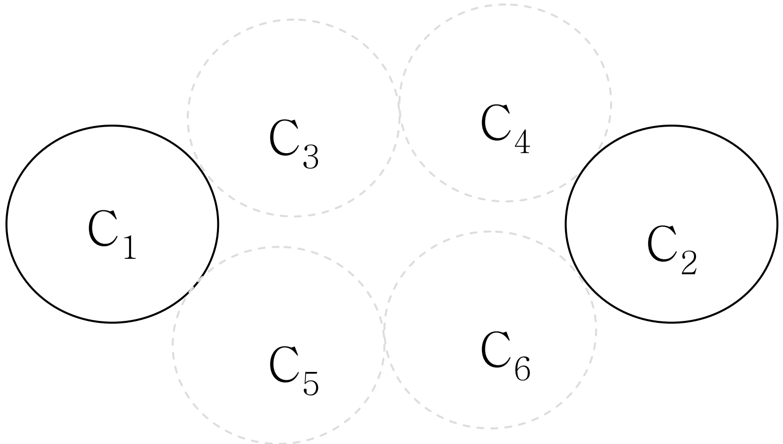
- A counterexample is spurious if, for some process, the set of all member cycles in that process is not self-connected.

Refinement



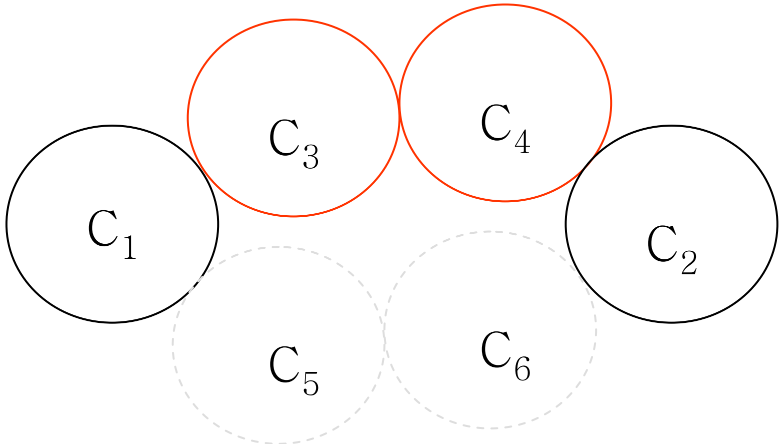
- Consider a counterexample that contains C_1 and C_2 only.

Refinement



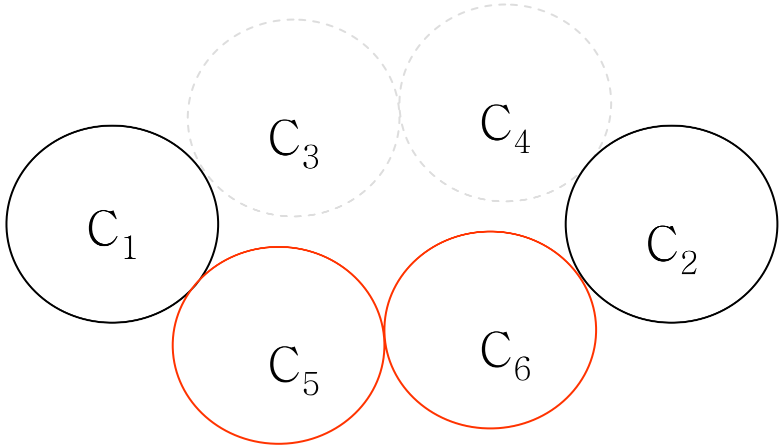
- Consider a counterexample that contains C_1 and C_2 only.
  - determine all the self-connected sets that contain C_1 and C_2.

Refinement



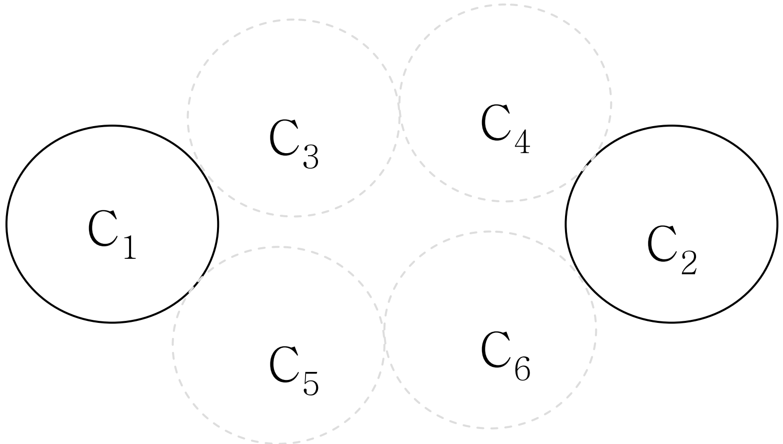
- Consider a counterexample that contains C_1 and C_2 only.
  - determine all the self-connected sets that contain C_1 and C_2.

Refinement



- Consider a counterexample that contains C_1 and C_2 only.
  - determine all the self-connected sets that contain C_1 and C_2.
    - if there is no such set, then C_1 and C_2 belong to differenct SCCs.

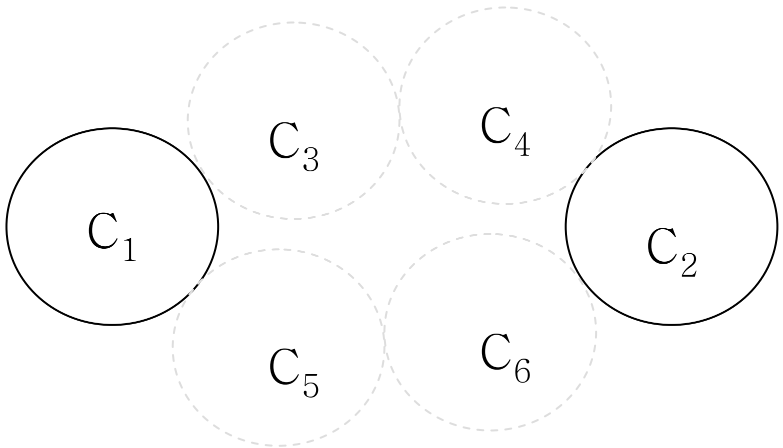
Refinement



- Several alternatives:
  - C_1 and C_2 are not executed infinitely often.

    $x_1 = 0 \wedge x_2 = 0$

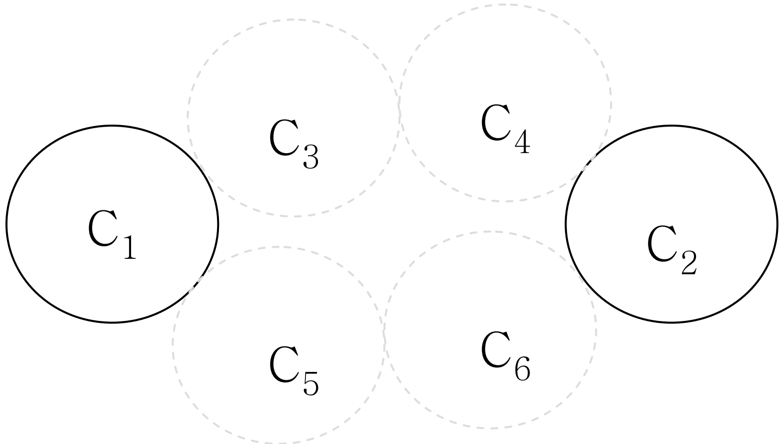
Refinement



- Several alternatives:
  - C_1 is not executed infinitely often while C_2 is.

    $x_1 = 0 \wedge x_2 > 0$

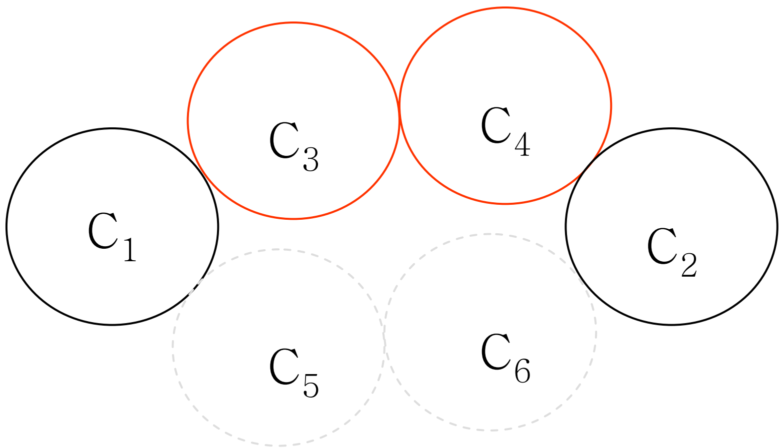
Refinement



- Several alternatives:
  - C_2 is not executed infinitely often while C_1 is.

    $x_1 > 0 \wedge x_2 = 0$

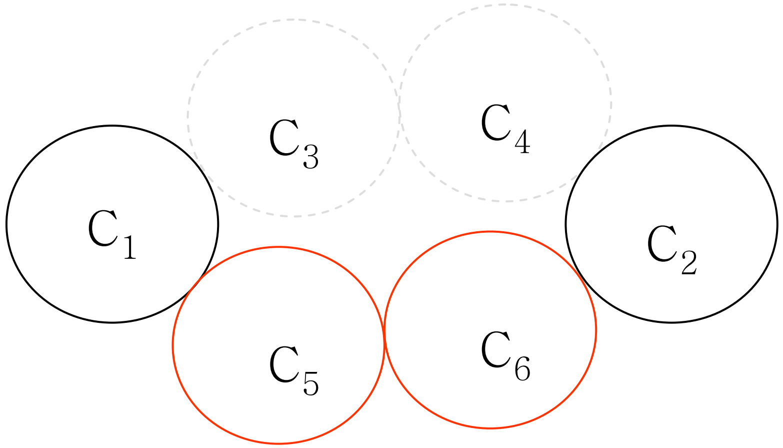
Refinement



- Several alternatives:
  - C_1 and C_2 are both executed infinitely often, C_3 and C_4 are also executed infinitely often.

    $x_1 > 0 \wedge x_2 > 0 \wedge x_3 > 0 \wedge x_4 > 0$

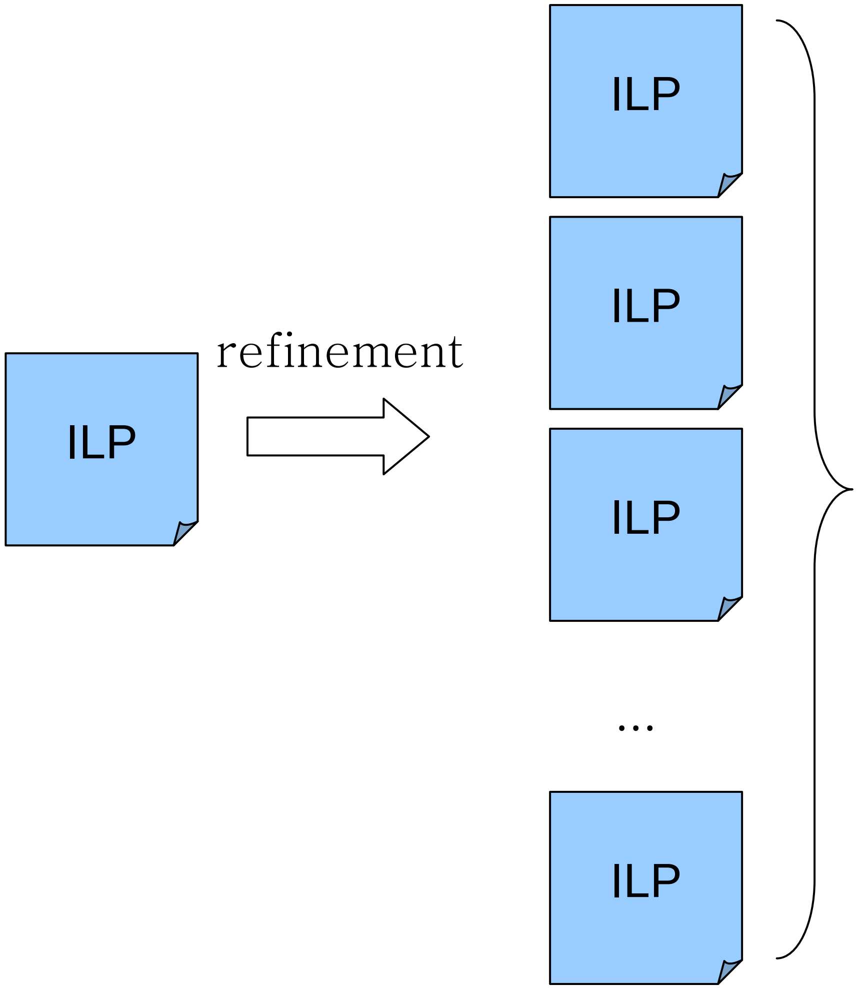
Refinement



- Several alternatives:
  - C_1 and C_2 are both executed infinitely often, C_5 and C_6 are also executed infinitely often.

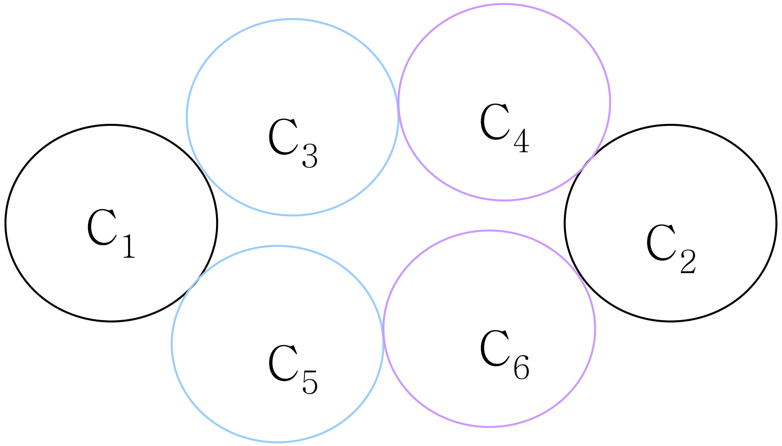
    $x_1 > 0 \wedge x_2 > 0 \wedge x_5 > 0 \wedge x_6 > 0$

Refinement



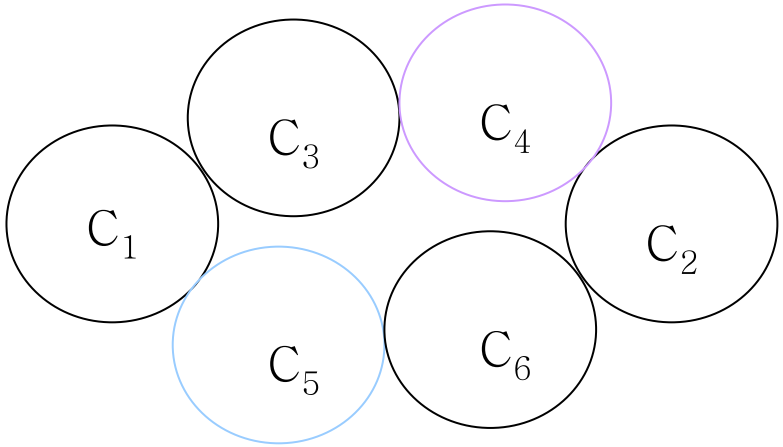
n + 3 new ILP problems where n is the number of self-connected cycle sets, which could be exponential in the number of cycles.

Coarser Refinement



- $x_1 = 0 \wedge x_2 = 0$
- $x_1 = 0 \wedge x_2 > 0$
- $x_1 > 0 \wedge x_2 = 0$
- $x_1 > 0 \wedge x_2 > 0 \wedge x_3 + x_5 > 0 \wedge x_4 + x_6 > 0$

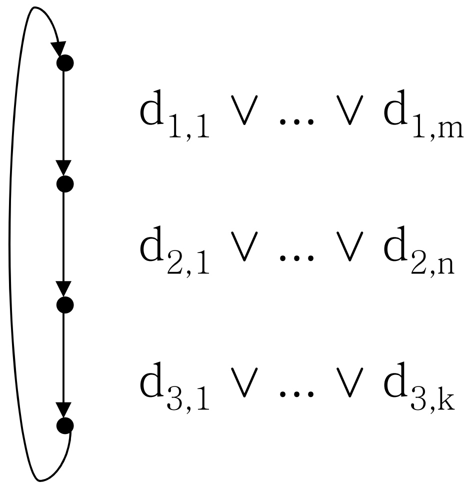
Coarser Refinement



- $x_1 = 0 \wedge x_2 = 0$
- $x_1 = 0 \wedge x_2 > 0$
- $x_1 > 0 \wedge x_2 = 0$
- $x_1 > 0 \wedge x_2 > 0 \wedge x_3 + x_5 > 0 \wedge x_4 + x_6 > 0$

Complexity

- Counterexample spuriousness is undecidable.
- High complexity in theory.
  - The number of ILP-problems to determine the maximal step value of a control expression is exponential both in the number of condition statements and in the size of each condition statement.
- Efficient in practice.



Experimental Results

- IBOC (IMCOS Boundedness Checker)

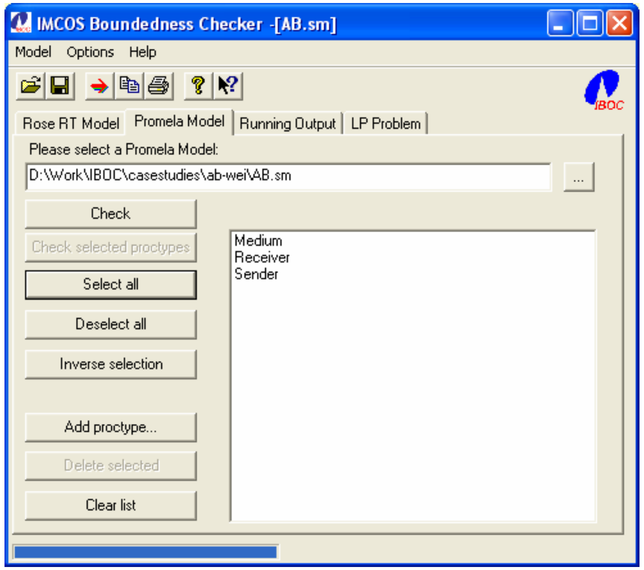
  http://www.inf.uni-konstanz.de/soft/tools_en.php?sys=3

- Tests on 31 models:
  - 8 of 31 are proved bounded without counterexamples reported.
  - 2 of 31 are proved bounded after refinement.
  - IBOC returned „UNKNOWN“ for 21 of 31.
    - 12 of 21 are truly unbounded.

- On the model of the MVCC protocol, IBOC found 4 counterexamples and determined 3 of them as spurious.

Conclusion

- Determine spuriousness for counterexamples by analyzing cycle code and control flow graph structures.
- Refine abstract models by use of cycle dependency information obtained from counterexample analyses.
- We have implemented the method in IBOC.



Future Work

- Study of global cycle dependencies.
- Application of the method to UML RealTime models.

Thank you!

Welcome to visit Konstanz and the beautiful Bodensee.