An Analytical and Experimental Comparison of CSP Extensions and Tools

*Ling Shi*¹, Yang Liu², Jun Sun³, Jin Song Dong¹, and Gustavo Carvalho⁴

¹SoC, National University of Singapore  
²SCE, Nanyang Technological University  
³ISTD, Singapore University of Technology and Design  
⁴Centro de Informática, UFPE, Brazil

ICFEM 2012: 14th International Conference on Formal Engineering Methods

http://www.comp.nus.edu.sg/~pat/compare
Outline

1. Motivation

2. CSP\textsubscript{M} vs. CSP#
   - Syntax
   - Operational Semantics

3. Verification Tool Support
   - Verification
   - Experiment

4. Conclusion
Outline

1 Motivation

2 CSP\textsubscript{M} vs. CSP#
   - Syntax
   - Operational Semantics

3 Verification Tool Support
   - Verification
   - Experiment

4 Conclusion
CSP and its Extensions

- CSP
- Extensions
  - CSP$_M$, CSP#, Circus, ...
CSP and its Extensions

- CSP
- Extensions
  - CSP\(_M\), CSP\#, Circus, . . .
- Differences?
  - Little work on comprehensive comparisons
CSP and its Extensions

- CSP
- Extensions
  - CSP\textsubscript{M}, CSP#, Circus, \ldots
- Differences?
  - Little work on comprehensive comparisons
- Our goals
  - To explore modeling capabilities and verification power
  - To derive practical guidelines
  - To help users to choose more suitable languages and tools
Outline

1 Motivation

2 $\text{CSP}_M$ vs. CSP#
   - Syntax
   - Operational Semantics

3 Verification Tool Support
   - Verification
   - Experiment

4 Conclusion
## Common Process Syntax

<table>
<thead>
<tr>
<th>CSP</th>
<th>CSP_M</th>
<th>CSP#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
<td>STOP</td>
<td>Stop</td>
<td>deadlock</td>
</tr>
<tr>
<td>SKIP</td>
<td>SKIP</td>
<td>Skip</td>
<td>termination</td>
</tr>
<tr>
<td>\textit{a} → \textit{P}</td>
<td>\textit{a} → \textit{P}</td>
<td>\textit{a} → \textit{P}</td>
<td>event prefixing</td>
</tr>
<tr>
<td>\textit{c}!\textit{e} → \textit{P}</td>
<td>\textit{c}?\textit{x}!\textit{x'} : \textit{V}!\textit{e} → \textit{P}</td>
<td>\textit{c}!\textit{e} → \textit{P}</td>
<td>(synchronous) channel communication</td>
</tr>
<tr>
<td>\textit{c}?\textit{x} → \textit{P}</td>
<td>\textit{c}?\textit{b}!\textit{x} → \textit{P}</td>
<td>\textit{c}?\textit{b}!\textit{x} → \textit{P}</td>
<td></td>
</tr>
<tr>
<td>\textit{P} \sqcap \textit{Q}</td>
<td>\textit{P} \sqcup \textit{Q}</td>
<td>\textit{P} \sqcup \textit{Q}</td>
<td>external choice</td>
</tr>
<tr>
<td>\textit{P} \sqcup \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>internal choice</td>
</tr>
<tr>
<td>\textit{P}; \textit{Q}</td>
<td>\textit{P}; \textit{Q}</td>
<td>\textit{P}; \textit{Q}</td>
<td>sequential composition</td>
</tr>
<tr>
<td>\textit{P} \backslash \textit{A}</td>
<td>\textit{P} \backslash \textit{A}</td>
<td>\textit{P} \backslash \textit{A}</td>
<td>hiding</td>
</tr>
<tr>
<td>\textit{P} \triangledown \textit{b} \triangledown \textit{Q}</td>
<td>\textit{if} \textit{b} \textit{then} \textit{P} \textit{else} \textit{Q}</td>
<td>\textit{if} \textit{b} \textit{then} \textit{P} \textit{else} \textit{Q}</td>
<td>conditional choice</td>
</tr>
<tr>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>interleaving</td>
</tr>
<tr>
<td>\textit{P} \triangle \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>\textit{P} \triangledown \textit{Q}</td>
<td>interrupt</td>
</tr>
</tbody>
</table>
Different Syntax - Data Perspective

- Process parameter
  - CSP$_M$: variable, process, function, and channel
  - CSP#: variable

- Channel
  - CSP$_M$: declared with an explicit type
  - CSP#: declared without type information

- Shared variable
  - CSP#: data operation prefixing ($a\{prog\} \rightarrow P$)
    - built-in type: integer, Boolean, array of integer or Boolean
    - user-defined type: written in an external C# (Java, C++, ...)
      class

L. Shi, Y. Liu, J. Sun, J. S. Dong, and G. Carvalho
Asynchronous channel

CSP#: $ac!e \rightarrow P$, $ac?x \rightarrow P$ or $ac?[b]x \rightarrow P$
Asynchronous channel
- CSP#: \( ac!e \rightarrow P \), \( ac?x \rightarrow P \) or \( ac?[b]x \rightarrow P \)

Parallel composition
- CSP\(_M\): sharing \((P[| A ||] Q)\), alphabetized \((P[A || A'] Q)\), and linked \((P[c \leftrightarrow c'] Q)\)
- CSP#: parallel \((P || Q)\) without specified alphabet
Different Syntax - Process Perspective

- Asynchronous channel
  - CSP#: \( ac!e \rightarrow P, \ ac?x \rightarrow P \) or \( ac?[b]x \rightarrow P \)

- Parallel composition
  - CSP\(_M\): sharing \( P[]A[]Q \), alphabetized \( P[A]A'[Q] \), and linked \( P[c \leftrightarrow c']Q \)
  - CSP#: parallel \( P || Q \) without specified alphabet

- Chaotic process \( \text{CHAOS}(A) \), event renaming \( P[[c \leftarrow c']] \), and untimed timeout \( P[> Q] \) are supported by CSP\(_M\)

- General choice \( P[]Q \), atomic/blocking conditional choice \( \text{ifa} b \{P\} \text{ else } \{Q\} / \text{ifb} b \{P\} \) are supported by CSP#
Operational Semantics

- Semantic model: Labeled Transition Systems (LTS)
  - Configuration: a pair of process and environment

- Common semantics
  - *Stop*, *event prefixing*, *external/external choice*, . . .

- Different semantics
  - *Channel communication*, *shared variable*, *conditional choice*, . . .
Channel Communication (1/2)

- Synchronous channel communication
  - CSP\textsubscript{M}: alphabetized event synchronization
  - CSP\#: hand shaking
Channel Communication (1/2)

- Synchronous channel communication
  - CSP\textsubscript{M}: alphabetized event synchronization
  - CSP\#: hand shaking

- Model CSP\textsubscript{M} synchronous channel in CSP\#
  - Output \( c!e \rightarrow P \Rightarrow \) event prefixing \( c.e \rightarrow P \)
  - Input \( \Rightarrow \) enumerate all possible communications using general choice ([]) of event prefixing process

L. Shi, Y. Liu, J. Sun, J. S. Dong, and G. Carvalho

http://www.comp.nus.edu.sg/~pat/compare 10/25
Asynchronous channel communication (CSP# only)
- Message passing
  - buffer size > 0
  - buffer in a first-in-first-out order (FIFO)
Asynchronous channel communication (CSP# only)
- Message passing
  - buffer size > 0
  - buffer in a first-in-first-out order (FIFO)
- Model asynchronous channel in CSP$_M$
  - Process $Buffer$ specifies the FIFO buffer
    
    $Buffer(c, \langle \rangle, N) = receive?c?x \rightarrow Buffer(c, \langle x \rangle, N)$
    
    $Buffer(c, s \triangleleft \langle a \rangle, N) = \#s < N - 1 \& receive?c?x \rightarrow Buffer(c, \langle x \rangle \triangleleft s \triangleleft \langle a \rangle, N)$
    
    $send!c!a \rightarrow Buffer(c, s, N)$

- E.g., process $P$ performs a communication over an asynchronous channel $ac$ with buffer size 2 can be modeled as $P[send \leftrightarrow receive, receive \leftrightarrow send]Buffer(ac, \langle \rangle, 2)$
Shared Variables

- **Shared variables in CSP#**
  - Reading/writing variables are modeled as terminating sequential programs in the form $a\{\text{prog}\} \rightarrow P$

- **Model shared variables in CSP$_M$**
  - a shared variable ($v$) is represented by a couple of processes ($\text{Var}$ and $\text{Var}_A$)
  - Accomplish atomic execution - variable $v$ can be accessed only by one process that invokes the variable

\[
\begin{align*}
\text{Var}(v, \text{val}) &= \text{read}?i!v!\text{val} \rightarrow \text{Var}(v, \text{val}) \\
\hspace{1em} \text{[] write}?i!v?x & \rightarrow \text{Var}(v, x) \text{[] start_at}?j!v & \rightarrow \text{Var}_A(j, v, \text{val}) \\
\text{Var}_A(j, v, \text{val}) &= \text{read}.j!v!\text{val} \rightarrow \text{Var}_A(j, v, \text{val}) \\
\hspace{1em} \text{[] write}.j!v?x & \rightarrow \text{Var}_A(j, v, x) \text{[] end_at}?j!v & \rightarrow \text{Var}(v, \text{val})
\end{align*}
\]
Outline

1 Motivation

2 CSP_M vs. CSP#
   - Syntax
   - Operational Semantics

3 Verification Tool Support
   - Verification
   - Experiment

4 Conclusion
Verification (1/3)

- CSP$_M$ supported by
  - FDR: refinement checker
  - ProB: type checker, animator and model checker
- CSP# supported by
  - PAT: simulator, model checker

Supported assertions

<table>
<thead>
<tr>
<th>Assertion</th>
<th>FDR</th>
<th>ProB</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadlock</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Livelock</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Determinism</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Refinement</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reachability</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LTL</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

$^1$ also PAT can perform LTL checking under fairness assumptions
**Model checking techniques**

<table>
<thead>
<tr>
<th>Technique</th>
<th>FDR</th>
<th>ProB</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BFS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCC for LTL checking</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reduction</td>
<td>–</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• six state-space compression methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• partial-order compression chase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduction</td>
<td>–</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• atomic sequence construct to compress state space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• partial order reduction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• process counter abstraction</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More about FDR and PAT

<table>
<thead>
<tr>
<th>Feature</th>
<th>FDR</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time</td>
<td>digitisation</td>
<td>digitisation zone abstraction</td>
</tr>
<tr>
<td>Symbolic technique</td>
<td>SAT BDD</td>
<td>SAT BDD</td>
</tr>
<tr>
<td>Multi-core</td>
<td>–</td>
<td>LTL checking Swarm verification Nested DFS search</td>
</tr>
<tr>
<td>Compositional reasoning</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Probability</td>
<td>–</td>
<td>✓</td>
</tr>
</tbody>
</table>
Experiment Overview

- Four aspects over eight benchmark systems
  - Shared variables (different models)
  - LTL Checking (same model)
  - Refinement checking (same model)
  - Solving puzzles (different models)
Experiment Results on Shared Variables

- CSPₘ model - alphabetized event synchronization
- CSP# model - shared variables

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>Property</th>
<th>FDR</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>State</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Concurrent Stack*2</td>
<td>3</td>
<td>P [T= S</td>
<td>453,456</td>
<td>3.833</td>
</tr>
<tr>
<td>Concurrent Stack*2</td>
<td>4</td>
<td>P [T= S</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Concurrent Stack*2</td>
<td>5</td>
<td>P [T= S</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peterson</td>
<td>3</td>
<td>mutual exclusion</td>
<td>1,011</td>
<td>1.192</td>
</tr>
<tr>
<td>Peterson</td>
<td>4</td>
<td>mutual exclusion</td>
<td>105,493</td>
<td>20.067</td>
</tr>
<tr>
<td>Peterson</td>
<td>5</td>
<td>mutual exclusion</td>
<td>14,810,779</td>
<td>387.645</td>
</tr>
</tbody>
</table>

N: number of processes; State: number of visited states; Time(s): running time in seconds; “-”: memory overflow or execution time exceeding two hours

‡ Performed by DFS with anti-chain algorithm (Session 9: "More Anti-Chain Based Refinement Checking")
## Experiment Results on LTL Checking

### Same models - common processes

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>Property</th>
<th>Result</th>
<th>FDR</th>
<th>ProB</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>State</td>
<td>Time(s)</td>
<td>State</td>
</tr>
<tr>
<td>R/W</td>
<td>6</td>
<td>$\Box \neg \text{error}$</td>
<td>true</td>
<td>8</td>
<td>0.023</td>
<td>122722</td>
</tr>
<tr>
<td>R/W</td>
<td>200</td>
<td>$\Box \neg \text{error}$</td>
<td>true</td>
<td>202</td>
<td>1.455</td>
<td>-</td>
</tr>
<tr>
<td>R/W</td>
<td>500</td>
<td>$\Box \neg \text{error}$</td>
<td>true</td>
<td>502</td>
<td>19.901</td>
<td>-</td>
</tr>
<tr>
<td>R/W</td>
<td>1000</td>
<td>$\Box \neg \text{error}$</td>
<td>true</td>
<td>1,002</td>
<td>154.33</td>
<td>-</td>
</tr>
<tr>
<td>DP</td>
<td>6</td>
<td>$\Box \Box \text{eat}.0$</td>
<td>false</td>
<td>N.A.</td>
<td>-</td>
<td>2,420</td>
</tr>
<tr>
<td>DP</td>
<td>8</td>
<td>$\Box \Box \text{eat}.0$</td>
<td>false</td>
<td>N.A.</td>
<td>-</td>
<td>13,312</td>
</tr>
<tr>
<td>DP</td>
<td>12</td>
<td>$\Box \Box \text{eat}.0$</td>
<td>false</td>
<td>N.A.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

R/W: Readers/Writers  
DP: Dinning philosopher

---

G. Lowe. (FDR)
### Experiment Results on Refinement Checking

- **Same models - common processes**
- **Different model checking techniques**

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>Property</th>
<th>FDR</th>
<th>ProB</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>State</td>
<td>Time(s)</td>
<td>State</td>
</tr>
<tr>
<td>R/W</td>
<td>6</td>
<td>P [T= S]</td>
<td>8</td>
<td>0.024</td>
<td>61,365</td>
</tr>
<tr>
<td>R/W</td>
<td>200</td>
<td>P [T= S]</td>
<td>202</td>
<td>1.434</td>
<td>-</td>
</tr>
<tr>
<td>R/W</td>
<td>500</td>
<td>P [T= S]</td>
<td>502</td>
<td>19.651</td>
<td>-</td>
</tr>
<tr>
<td>R/W</td>
<td>1000</td>
<td>P [T= S]</td>
<td>1,002</td>
<td>156.162</td>
<td>-</td>
</tr>
<tr>
<td>DP</td>
<td>6</td>
<td>P [F= S]</td>
<td>1</td>
<td>0.06</td>
<td>14,510</td>
</tr>
<tr>
<td>DP</td>
<td>8</td>
<td>P [F= S]</td>
<td>1</td>
<td>0.071</td>
<td>-</td>
</tr>
<tr>
<td>DP</td>
<td>12</td>
<td>P [F= S]</td>
<td>1</td>
<td>0.104</td>
<td>-</td>
</tr>
<tr>
<td>MCS</td>
<td>20</td>
<td>P [FD= S]</td>
<td>40</td>
<td>0.043</td>
<td>-</td>
</tr>
<tr>
<td>MCS</td>
<td>50</td>
<td>P [FD= S]</td>
<td>100</td>
<td>0.086</td>
<td>-</td>
</tr>
<tr>
<td>MCS</td>
<td>100</td>
<td>P [FD= S]</td>
<td>200</td>
<td>0.246</td>
<td>-</td>
</tr>
</tbody>
</table>

R/W: Readers/Writers  
DP: Dinning philosopher  
MCS: Minler’s cyclic scheduler

L. Shi, Y. Liu, J. Sun, J. S. Dong, and G. Carvalho
### Experiment Results on Solving Puzzles

- **CSP\textsubscript{M} model** - alphabetized event synchronization, FDR, ProB - trace refinement
- **CSP\# model** - shared variables, PAT - reachability analysis

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>FDR</th>
<th>FDR-\textsubscript{Div}*</th>
<th>ProB</th>
<th>PAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>State</td>
<td>Time(s)</td>
<td>State</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Solitaire</td>
<td>26</td>
<td>4,048,216</td>
<td>46.303</td>
<td>1</td>
<td>0.169</td>
</tr>
<tr>
<td>Solitaire</td>
<td>29</td>
<td>28,249,254</td>
<td>387.737</td>
<td>1</td>
<td>0.217</td>
</tr>
<tr>
<td>Solitaire</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>5.318</td>
</tr>
<tr>
<td>Solitaire</td>
<td>35</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>377.297</td>
</tr>
<tr>
<td>Knight</td>
<td>5</td>
<td>508,450</td>
<td>3.522</td>
<td>1</td>
<td>0.037</td>
</tr>
<tr>
<td>Knight</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>15.399</td>
</tr>
<tr>
<td>Knight</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>94.713</td>
</tr>
<tr>
<td>Hanoi</td>
<td>6</td>
<td>729</td>
<td>0.052</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>Hanoi</td>
<td>7</td>
<td>2,187</td>
<td>0.086</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>Hanoi</td>
<td>8</td>
<td>6,561</td>
<td>0.181</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

*FDR-\textsubscript{Div}*: check the divergence of a new system which only performs up to \(N\) events of the puzzle model and then performs an infinite number of events

*N.A.:* no models for the tool

_L. Shi, Y. Liu, J. Sun, J. S. Dong, and G. Carvalho_
Outline

1. Motivation

2. CSP_M vs. CSP#
   - Syntax
   - Operational Semantics

3. Verification Tool Support
   - Verification
   - Experiment

4. Conclusion
Conclusion

What have done

- Proposed transformation rules between CSP\textsubscript{M} and CSP#
- Evaluated the efficiency of three model checkers FDR, ProB and PAT
- Provided guideline for specifying and verifying concurrent systems

What to do

- To explore the semantic equivalence of transformation rules
- To extend the comparison to real-time operators
References

C. Hoare.  

M. Leuschel and M. Fontaine.  

A. W. Roscoe.  

J. Sun, Y. Liu, J. S. Dong, and C. Chen.  

J. Sun, Y. Liu, J. S. Dong, and J. Pang.  
Acknowledgement:
Thanks to Professor Bill Roscoe and Professor Michael Leuschel

Thank you!