An Analytical and Experimental Comparison of CSP Extensions and Tools

Ling Shi[1], Yang Liu[2], Jun Sun[3], Jin Song Dong[1], and Gustavo Carvalho[4]

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

ICFEM 2012: 14th International Conference on Formal Engineering Methods

Outline

1. Motivation
2. CSP_M vs. CSP#
   - Syntax
   - Operational Semantics
3. Verification Tool Support
   - Verification
   - Experiment
4. Conclusion

Outline

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_M, CSP#, Circus, . . .
- 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

Common Process Syntax

CSP	**CSP$_M$**	**CSP#**	**Description**
$STOP$	$STOP$	$Stop$	deadlock
$SKIP$	$SKIP$	$Skip$	termination
$a \rightarrow P$	$a \rightarrow P$	$a \rightarrow P$	event prefixing
$c!e \rightarrow P$ $c?x \rightarrow P$	$c?x?x' : V!e \rightarrow P$	$c!e \rightarrow P$ $c?[b]x \rightarrow P$	(synchronous) channel communication
$P \Box Q$	$P \; [] \; Q$	$P \; [*] \; Q$	external choice
$P \sqcap Q$	$P \; \|\sim\| \; Q$	$P <> Q$	internal choice
$P; \; Q$	$P; \; Q$	$P; \; Q$	sequential composition
$P \setminus A$	$P \setminus A$	$P \setminus A$	hiding
$P \lhd b \rhd Q$	$if \; b \; then \; P \; else \; Q$	$if \; b \; then \; P \; else \; Q$	conditional choice
$P \; \|\|\| \; Q$	$P \; \|\|\| \; Q$	$P \; \|\|\| \; Q$	interleaving
$P \triangle Q$	$P /\backslash Q$	$P \; interrupt \; Q$	interrupt

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

Different Syntax - Process Perspective

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

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 ($P[A \parallel A']Q$), and linked ($P[c \leftrightarrow c']Q$)
  - CSP#: parallel ($P \parallel Q$) 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 ($P[A \parallel A']Q$), and linked ($P[c \leftrightarrow c']Q$)
  - CSP#: parallel ($P \parallel Q$) without specified alphabet
- Chaotic process ($CHAOS(A)$), event renaming ($P[[c \leftarrow c']]$), and untimed timeout ($P[> Q$) are supported by CSP_M
- General choice ($P[]Q$), *atomic*/*blocking* conditional choice ($ifa\ b\ \{P\}\ else\ \{Q\} / ifb\ b\ \{P\}$) 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/internal choice*, ...
- Different semantics
  - *Channel communication*, *shared variable*, *conditional choice*, ...

Channel Communication (1/2)

- Synchronous channel communication
  - CSP_M: alphabetized event synchronization
  - CSP#: hand shaking

Channel Communication (1/2)

- Synchronous channel communication
  - CSP_M: alphabetized event synchronization
  - CSP#: hand shaking
- Model CSP_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

Channel Communication (2/2)

- Asynchronous channel communication (CSP# only)
  - Message passing
    - buffer size > 0
    - buffer in a first-in-first-out order (FIFO)

Channel Communication (2/2)

- 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

$$Buffer(c, \langle\rangle, N) = receive?c?x \rightarrow Buffer(c, \langle x\rangle, N)$$

$$Buffer(c, s \frown \langle a\rangle, N) = \#s < N - 1 \& receive?c?x \rightarrow Buffer(c, \langle x\rangle \frown s \frown \langle a\rangle, N) \; [] \; send!c!a \rightarrow Buffer(c, s, N)$$

  - 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 $a\{prog\} \rightarrow P$
- Model shared variables in CSP_M
  - a shared variable (v) is represented by a couple of processes (Var and Var_A)
  - Accomplish atomic execution - variable v can be accessed only by one process that invokes the variable

$$Var(v,\ val) = read?i!v!val \rightarrow Var(v,\ val)$$
$$[]\ write?i!v?x \rightarrow Var(v,\ x)\ []\ start\_at?j!v \rightarrow Var\_A(j,\ v,\ val)$$
$$Var\_A(j,\ v,\ val) = read.j!v!val \rightarrow Var\_A(j,\ v,\ val)$$
$$[]\ write.j!v?x \rightarrow Var\_A(j,\ v,\ x)\ []\ end\_at?j!v \rightarrow Var(v,\ val)$$

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

Assertion	FDR	ProB	PAT
Deadlock	✓	✓	✓
Livelock	✓	✓	✓
Determinism	✓	✓	✓
Refinement	✓	✓	✓
Reachability	-	✓	✓
LTL	-	✓	✓[1]

[1] also PAT can perform LTL checking under fairness assumptions

Verification (2/3)

- Model checking techniques

Technique	FDR	ProB	PAT
DFS	✓	✓	✓
BFS	✓	✓	✓
SCC for LTL checking	–	✓	✓
Reduction	•six state-space compression methods •partial-order compression *chase*	–	•atomic sequence construct to compress state space •partial order reduction •process counter abstraction

Verification (3/3)

- More about FDR and PAT

Feature	**FDR**	**PAT**
Real-time	digitisation	digitisation zone abstraction
Symbolic technique	SAT BDD	SAT BDD
Multi-core	–	LTL checking Swarm verification Nested DFS search
Compositional reasoning	✓	✓
Probability	–	✓

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

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

Model	N	Property	FDR		PAT	
			State	Time(s)	State	Time(s)
Concurrent Stack*2	3	P [T= S	453,456	3.833	10,860	1.023‡
Concurrent Stack*2	4	P [T= S	-	-	189,920	75.915‡
Concurrent Stack*2	5	P [T= S	-	-	693,828	293.382‡
Peterson	3	mutual exclusion	1,011	1.192	3,257	0.105
Peterson	4	mutual exclusion	105,493	20.067	104,686	3.776
Peterson	5	mutual exclusion	14,810,779	387.645	5,722,863	294.005

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

Model	N	Property	Result	FDR		ProB		PAT	
				State	Time(s)	State	Time(s)	State	Time(s)
R/W	6	$\Box !error$	true	8	0.023	122722	104.8	15	0.059
R/W	200	$\Box !error$	true	202	1.455	-	-	403	0.086
R/W	500	$\Box !error$	true	502	19.901	-	-	1,003	0.071
R/W	1000	$\Box !error$	true	1,002	154.33	-	-	2,003	0.148
DP	6	$\Box \Diamond eat.0$	false	N.A.	N.A.	2,420	1.11	166	0.019
DP	8	$\Box \Diamond eat.0$	false	N.A.	N.A.	13,312	1.75	256	0.024
DP	12	$\Box \Diamond eat.0$	false	N.A.	N.A.	-	-	460	0.049

R/W: Readers/Writers DP: Dinning philosopher

G. Lowe. (FDR)
Specification of communicating processes: temporal logic versus refusals-based refinement.
Formal Aspect of Computing, 220(3):277-294, May 2008.

Experiment Results on Refinement Checking

- Same models - common processes
- Different model checking techniques

Model	N	Property	FDR		ProB		PAT	
			State	Time(s)	State	Time(s)	State	Time(s)
R/W	6	P [T= S	8	0.024	61,365	125.94	9	0.04
R/W	200	P [T= S	202	1.434	-	-	203	0.11
R/W	500	P [T= S	502	19.651	-	-	503	0.057
R/W	1000	P [T= S	1,002	156.162	-	-	1,003	0.108
DP	6	P [F= S	1	0.06	14,510	82.42	1,762	0.174
DP	8	P [F= S	1	0.071	-	-	22,362	2.995
DP	12	P [F= S	1	0.104	-	-	-	-
MCS	20	P [FD= S	40	0.043	-	-	60	0.114
MCS	50	P [FD= S	100	0.086	-	-	150	0.143
MCS	100	P [FD= S	200	0.246	-	-	300	0.53

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

Experiment Results on Solving Puzzles

- CSP_M mdoel - alphabetized event synchronization, FDR, ProB - trace refinement
- CSP# model - shared variables, PAT - reachability analysis

Model	N	FDR		FDR-Div*		ProB		PAT	
		State	Time(s)	State	Time(s)	State	Time(s)	State	Time(s)
Solitaire	26	4,048,216	46.303	1	0.169	-	-	11,950	5.356
Solitaire	29	28,249,254	387.737	1	0.217	-	-	104,395	54.681
Solitaire	32	-	-	1	5.318	-	-	10,955	5.301
Solitaire	35	-	-	1	377.297	-	-	443,230	279.454
Knight	5	508,450	3.522	1	0.037	-	-	4,256	0.29
Knight	6	-	-	1	15.399	-	-	129,269	9.143
Knight	7	-	-	1	94.713	-	-	77,238	6.754
Hanoi	6	729	0.052	N.A.	N.A.	1,667	57.84	5,775	0.416
Hanoi	7	2,187	0.086	N.A.	N.A.	4,969	196.5	92,680	6.837
Hanoi	8	6,561	0.181	N.A.	N.A.	14,853	660.59	150,918	11.524

*FDR-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

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_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.
Communicating Sequential Processes. Prentice-Hall, 1985.

M. Leuschel and M. Fontaine.
Probing the depths of CSP-M: A new FDR-compliant validation tool. *ICFEM*, pages 278-297, 2008.

A. W. Roscoe.
The Theory and Practice of Concurrency. Prentice Hall PTR, 1997.

J. Sun, Y. Liu, J. S. Dong, and C. Chen.
Integrating specification and programs for system modeling and verification. *TASE*, pages 127-135, 2009.

J. Sun, Y. Liu, J. S. Dong, and J. Pang.
PAT: Towards flexibile verification under fairness. *CAV*, pages 702-708, 2009.

Acknowledgement:

Thanks to Professor Bill Roscoe and Professor Michael Leuschel

Thank you!