Comparing CSP_M (FDR) and CSP# (PAT)

SHI Ling

School of Computing

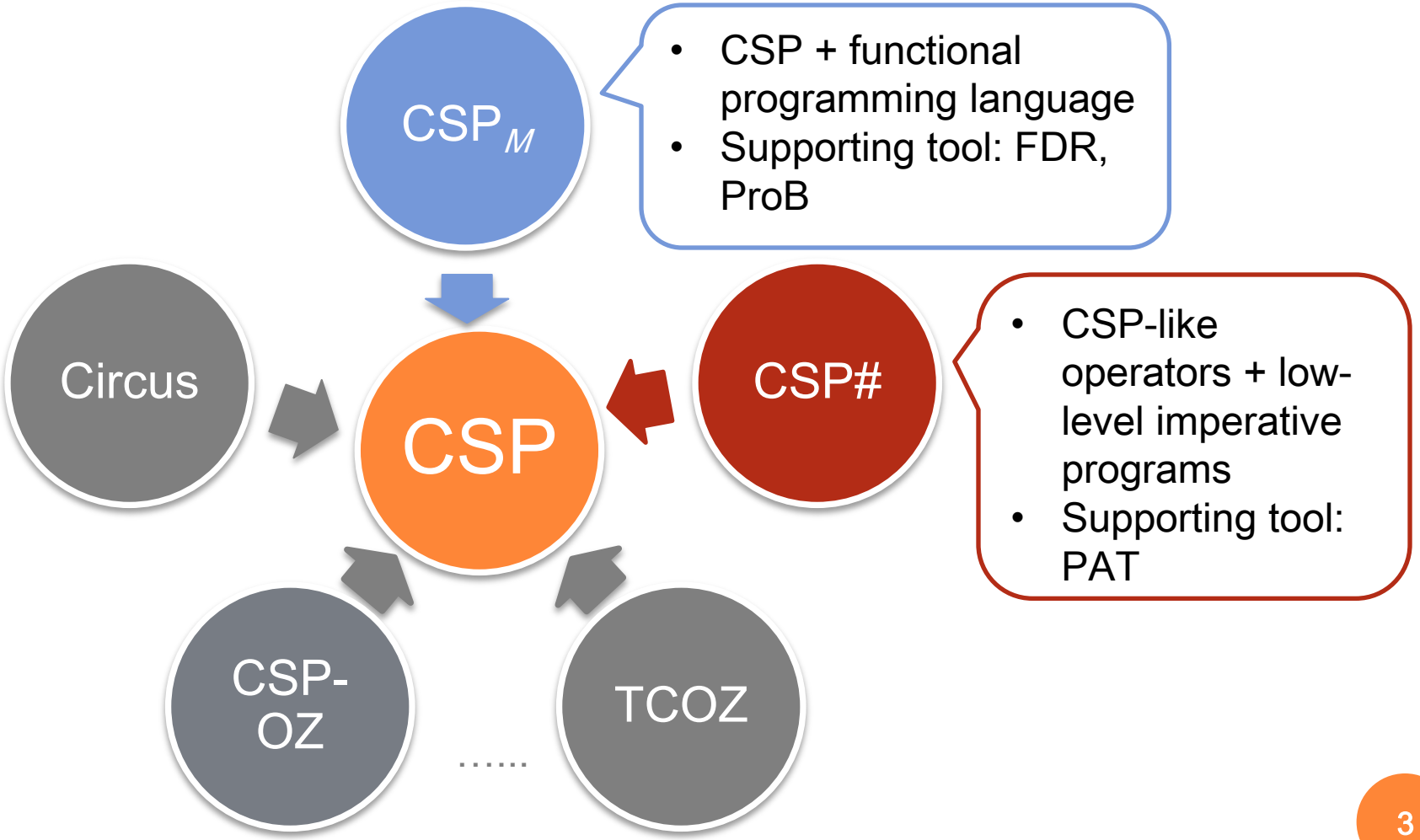
National University of Singapore

February 23， 2012

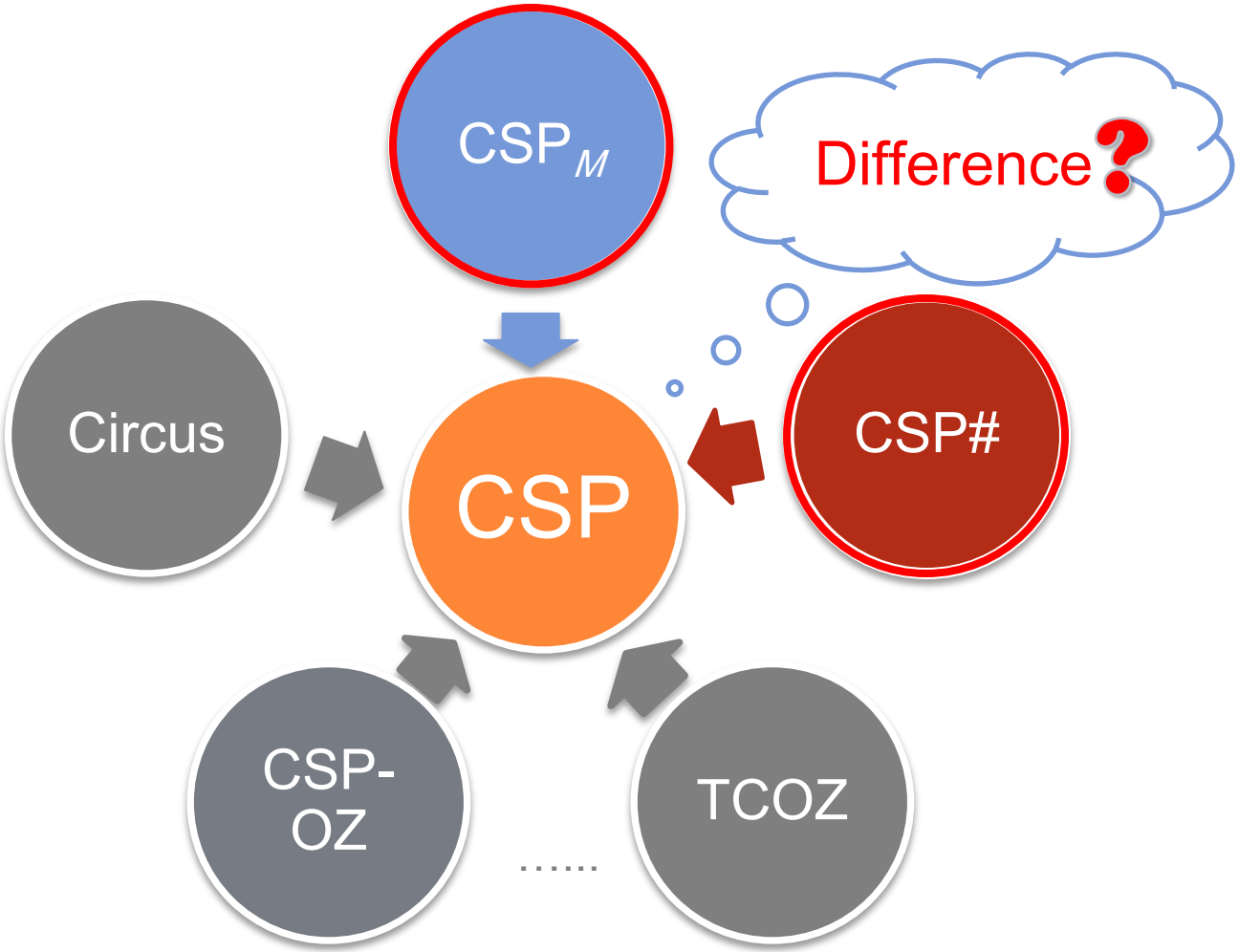
Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

CSP and its Extensions



Motivation



Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

Common Syntax

CSP	CSP_M	CSP#	Description
$STOP$	$STOP$	$Stop$	deadlock
$SKIP$	$SKIP$	$Skip$	termination
$CHAOS$	$CHAOS(A)$	-	chaotic process
$a \rightarrow P$	$a \rightarrow P$	$a \rightarrow P$	event prefixing
$c!e \rightarrow P$ $c?x \rightarrow P$	$c?x?x' : V!e \rightarrow P$	$c!e \rightarrow P$ $c?[b]x \rightarrow P$	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
$x := e$	-	$x := e^{\dagger}$	assignment
$P \lhd b \rhd Q$	$if \; b \; then \; P \; else \; Q$	$if \; b \; then \; P \; else \; Q$	conditional choice
$P \parallel Q$	$P[\| A \|]Q$ $P[A \parallel A']Q$ $P[c <- > c']Q$	$P \parallel Q$	parallel composition
$P \mid\mid\mid Q$	$P \mid\mid\mid Q$	$P \mid\mid\mid Q$	interleaving
$P \triangle Q$	$P /\backslash Q$	$P \; interrupt \; Q$	interrupt

† Assignment is in the data operation process.

Additional Syntax (1/2)

CSP_M

- Parallel composition
  - sharing: P [|A |] Q
  - alphabetized parallel: P [A || A'] Q
  - linked parallel: P [c ↔ c'] Q
- Chaotic process: $CHAOS(A)$
- Event renaming: P [[$c \leftarrow c'$]]
- Untimed time-out: P [> Q
- Boolean guard: b & P
- Replicated operators
  - *replicated external / internal choice, parallel composition…*

CSP#

- Shared variables
  - type: integer, Boolean, …, user-defined type written in an external C# (Java, C++, …) class
  - data operation: $a\{prog\} \rightarrow P$
- Asynchronous channel
  - output: $ac!e \rightarrow P$
  - input: $ac?x \rightarrow P$, $ac?[b]x \rightarrow P$
- General choice: P [] Q
- *Atomic* / *blocking* conditional choices
  - *ifa* $b\{P\}$ *else* $\{Q\}$ | *ifb* $b\{P\}$
- Guarded process: $[b]P$
- Replicated operators

Additional Syntax (2/2)

CSP_M

- Process parameter
  - variable, process, function, channel
- Typing
  - channel: declared with an explicit type, values communicated through channels have to be in the type range; an error about the invalid value is reported at run-time by ProB, ignored by FDR
  - FDR checks CSP_M dynamically
  - ProB type checks the CSP_M model in a dynamic or (optional) static way [LeuschelF08]

CSP#

- Process parameter
  - variable
- Typing
  - a weak typing (a.k.a. loose typing)
  - channel: declared without type, channel input variables can take in values with different types at different time; invalid type casting exception can be raised at run-time
  - variable: declared with no type information

[LeuschelF08] M. Leuschel and M. Fontaine. Probing the Depths of CSP-M: A New fdr-Compliant Validation Tool. *ICFEM*, pages 278–297, 2008.

Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

Operational Semantics

- Semantic model: Labeled Transition Systems (LTS)
  - Configuration
    - CSP_M: a pair of process and environment
    - CSP#: a pair (V, P)
  - Semantics is defined by firing rules
- Common semantics of CSP_M and CSP# processes
  - Stop
  - hiding
  - interrupt
  - event prefixing
  - Boolean guard
  - external/internal choice
  - sequential composition

Firing Rules – Channel Communication

- Channel communication
  - Synchronous channel communication
    - alphabetized event synchronization in CSP$_M$
      - general form: $cf \rightarrow P$ where c is a channel name, f is a sequence of communication fields
      - a communication field: 1) output: $!e$, 2) unconstrained input $?x$ and 3) constrained input $?x:V$
    - hand shaking in CSP#
      - channel output: $c!e \rightarrow P$
      - channel input: $c?x \rightarrow$ P or $c?[b]x \rightarrow$ P
  - Asynchronous channel communication
    - message passing in CSP#
      - buffer size >0
      - buffer in a first-in-first-out order (FIFO)

Firing Rules – Synchronous Channel

CSP_M

a set of possible events

$$\frac{a \in comms(cf)}{cf \rightarrow P \xrightarrow{a} subs(a, cf, P)}\ [\ M\_com\ ]$$

substitution

CSP#

evaluation

$$\frac{(V, c!e \rightarrow P) \xrightarrow{c!eva(V,e)} (V, P),\ (V, c?[b]x \rightarrow Q) \xrightarrow{c?[b]x} (V, Q),\ (V \wedge x = eva(V, e)) \Rightarrow b}{(V, c!e \rightarrow P \parallel c?[b]x \rightarrow Q) \xrightarrow{c.eva(V,e)} (V, P \parallel Q[eva(V, e)/x])}$$

- Model CSP_M synchronous channel in CSP#
  - output $c!e \rightarrow P$: event prefixing $c.e \rightarrow P$
  - input: enumerate all possible communications using general choice ([]) of event prefixing process

$$datatype\ Drink = Sprite \mid Coke \mid Tea \mid Coffee$$
$$channel\ offer : Drink$$
$$VM = offer?x : diff(Drink, \{Coffee\}) \rightarrow VM$$

CSP_M model

$$VM = offer.Sprite \rightarrow VM$$
$$[]\ offer.Coke \rightarrow VM$$
$$[]\ offer.Tea \rightarrow VM$$

transformed CSP# model

Firing Rules – Asynchronous channel

- Asynchronous channel output in CSP#

$$\frac{ac \text{ is not full in } V}{(V, ac!e \rightarrow P) \overset{ac!eva(V,e)}{\rightarrow} (app(V, ac!e), P)} [\ \#\_out\ ]$$

- Asynchronous channel input in CSP#

$$\frac{ac \text{ is not empty in } V \ \wedge (V \wedge x = top(ac)) \Rightarrow b}{(V, ac?[b]x \rightarrow P) \overset{ac?top(ac)}{\rightarrow} (pop(V, ac?x), P[top(ac)/x])} [\ \#\_in2\ ]$$

- Asynchronous channel modeled in CSP_M
  - process *Buffer* modeling the FIFO buffer, runs in parallel with other processes which performs the asynchronous communication

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

  - e.g., process *P* performs a communication over an asynchronous channel *ac* with buffer size 2 can be modeled as *P* [*snd* ↔ *rcv*, *rcv* ↔*snd*] *Buffer*(*ac*, ⟨ ⟩ ,2)

Firing Rules - Shared Variables

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

$$\frac{}{(V, a\{prog\} \rightarrow P) \xrightarrow{a} (upd(V, prog), P)} [\ \#\_dataPrefix\ ]$$

- Shared variables in CSP_M
  - a shared variable is represented by a *variable process,* executed concurrently with other *user processes* which invoke the variable

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

read

start atomic execution

write

end atomic execution

Firing Rules - Shared Variables

- Example
  - CSP# model

$$
\begin{aligned}
&\#define\ N\ 3;\\
&var\ count\ =\ 0;\\
&var\ sum : \{0..10\}\ =\ 0;\\
&P(i)\ =\ [count\ <\ N]add\{sum = sum + i;\ \ count = count + 1\}\ \rightarrow\ P(i);\\
&System()\ =\ |||\ \ i : \{1..3\}@P(i);
\end{aligned}
$$

  - CSP_M model

$$
\begin{aligned}
&datatype\ Var\ =\ count\ |\ sum\quad T\ =\ \{1..3\}\quad Range\ =\ \{0..10\}\\
&channel\ read,\ write :\ T.Var.Range\\
&channel\ start\_at,\ end\_at :\ T.Var\quad channel\ a\\
&P(i)\ =\ start\_at!i!count\ \rightarrow\ read!i?count?x\ \rightarrow\ x < N\ \&\ add\\
&\qquad\rightarrow\ start\_at!i!sum\ \rightarrow\ read!i?sum?y\ \rightarrow\ write!i!sum!(y + i)\\
&\qquad\rightarrow\ write!i!count!(x + 1)\ \rightarrow\ end\_at!i!sum\ \rightarrow\ end\_at!i!count\ \rightarrow\ P(i)\\
&Processes()\ =\ |||\ \ i : \{1..3\}@P(i)\\
&Variables()\ =\ Var(count,\ 0)\ |||\ \ Var(sum,\ 0)\\
&SharedEvent\ =\ \{read.t.v.val,\ write.t.v.val,\ start\_at.t.v,\ end\_at.t.v\ |\\
&\qquad\qquad t \leftarrow T,\ v \leftarrow Var,\ val \leftarrow Range\}\\
&System()\ =\ Variables()\ [|\ SharedEvent\ |]\ Processes()
\end{aligned}
$$

Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

VERIFICATION (1/2)

- 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	-	√	√ *

* PAT can perform LTL checking under fairness assumptions

VERIFICATION (2/2)

- Model checking techniques
  - DFS, BFS are all supported in FDR, ProB, PAT
  - SCC searching algorithm for LTL checking, is supported in ProB, PAT
  - Reduction techniques
    - FDR provides six compression methods to process LTS, such as eliminating τ actions, translating the LTS to a generalized LTS with divergence information or minimal acceptance sets of each node
    - PAT provides atomic sequence construct: *atomic*{*P*} to compress the state space
    - Partial order reduction (POR), applied not only to τ actions (similar to FDR) but also the visible events in some cases, is implemented in PAT
    - Process counter abstraction is supported in PAT to model checking the parameterized systems

Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

VERIFICATION OF FOUR BENCHMARK SYSTEMS

Model	N	Property	FDR		ProB		PAT	
			State	Time(s)	State	Time(s)	State	Time(s)
DP	5	TR	6	0.039	3K	5.58	393	0.159
DP	6	TR	7	0.045	14K	43.35	1K	0.124
DP	8	TR	9	0.054	-	-	14K	1.451
DP	12	TR	13	0.09	-	-	1.7M	316.135
MCS	20	TR	31M	301.03	-	-	153	<1
MCS	24	TR	596M	6967.463	-	-	185	<1
MCS	100	TR	-	-	-	-	793	<1
Hanoi	8	reachability/TR	6K	0.103	15K	456.93	151K	7.396
Hanoi	9	reachability/TR	19K	0.232	44K	1519.02	753K	39.39
Hanoi	10	reachability/TR	58K	0.584	-	-	49M	275.288
Peterson	4	mutual exclusion	241K	3.382	N.A.	N.A.	10K	0.243
Peterson	5	mutual exclusion	32M	371.729	N.A.	N.A.	239K	5.606
Peterson	6	mutual exclusion	-	-	N.A.	N.A.	6M	181.254

DP: dinning philosopher, MCS: Milner's cyclic scheduler, TR: trace refinement

LTL Model checking

Model	N	Property	Result	ProB		PAT	
				State	Time(s)	State	Time(s)
DP	6	[]<>eat.0	false	2420	0.94	270	0.066
DP	8	[]<>eat.0	false	13312	9.98	506	0.083
DP	10	[]<>eat.0	false	130604	141.85	846	0.106
MCS	10	[]<>a.0	true	68K	99.54	199	0.08
MCS	12	[]<>a.0	true	319K	741.32	243	0.075
MCS	100	[]<>a.0	true	-	-	2179	1.062

The detailed experiment descriptions and can be obtained from http://www.comp.nus.edu.sg/~pat/compare/.

Outline

- Motivation
- CSP_M vs. CSP#
  - Syntax
  - Operational Semantics
- Verification Tool Support
  - Verification
  - Experiment
- Conclusion

Conclusion

- Modeling language
  - CSP_M: more suitable for concurrent systems containing event-based synchronization
  - CSP#: more appropriate for systems involving various communication techniques (e.g., message passing) and systems requiring shared resources
- Model checker
  - FDR: best candidate for refinement checking
  - ProB: good for verifying LTL properties of CSP_M models
  - PAT: a better option to handle models where POR is applicable, and verify LTL properties under fairness assumptions

Reference

- [Hoa85] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.
- [LF08] Michael Leuschel and Marc Fontaine. Probing the depths of csp-m: A new fdr compliant validation tool. *ICFEM*, pages 278–297, 2008.
- [RHB97] A. W. Roscoe, C. A. R. Hoare, and Richard Bird. The Theory and Practice of Concurrency. Prentice Hall PTR, 1997.
- [Ros10] A. W. Roscoe. Understanding Concurrent Systems. Springer-Verlag New York, Inc., 2010.
- [SLDC09] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. Integrating specification and programs for system modeling and verification. In *TASE*, pages 127–135, 2009.
- [SLDP09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards Flexible Verification under Fairness. In *CAV*, pages 702–708, 2009.

Question?

Thank You !

Firing Rules – *SKIP and CHAOS*

CSP_M

- *SKIP*

$$\frac{}{Skip \xrightarrow{\checkmark} \Omega}\ [\ M\_skip\ ]$$

- *CHAOS*

$$\frac{}{CHAOS(A) \xrightarrow{\tau} STOP}\ [\ M\_c1\ ]$$

$$\frac{a \in A}{CHAOS(A) \xrightarrow{a} CHAOS(A)}\ [\ M\_c2\ ]$$

CSP#

- *Skip*

$$\frac{}{(V, Skip) \xrightarrow{\checkmark} (V, Stop)}\ [\ \#\_skip\ ]$$

- *CHAOS*
  - Not directly supported
  - Example

    Given set $A = \{a, b\}$, *CHAOS*(*A*) can be modeled in CSP# as

    CHAOS_A = $\tau \rightarrow$ *Stop*

    [] $a \rightarrow$ *CHAOS_A*

    [] $b \rightarrow$ *CHAOS_A*

Firing Rules – Parallel Composition

- Firing rules on handling √ event for parallel process P || Q (CSP#) and $P \underset{X}{\parallel} Q$ (CSP_M)
  - CSP#: distributed termination

$$\frac{(V,P) \xrightarrow{\checkmark} (V,P'), (V,Q) \xrightarrow{\checkmark} (V,Q')}{(V,P \parallel Q) \xrightarrow{\checkmark} (V,Stop)} [\ \#\_par2\ ]$$

  - CSP_M
    - either *P* or *Q* can terminate even if another process cannot

$$\frac{P \xrightarrow{\checkmark} P'}{P \underset{X}{\parallel} Q \xrightarrow{\tau} \Omega \underset{X}{\parallel} Q} [\ M\_par1\ ]$$

    - distributed termination

$$\frac{}{\Omega \underset{X}{\parallel} \Omega \xrightarrow{\checkmark} \Omega} [\ M\_par2\ ]$$

- Other firing rules in CSP# and CSP_M are similar