An Observational Theory of Imperative Concurrent Data Structures in the π-Calculus

Luca Fossati   Kohei Honda

Electronics, Engineering and Computer Science
Queen Mary University of London

Lisboa, 19/04/2011

Intro

Traditional global progress properties of concurrent programs :

- *Deadlock-Freedom*
- *Starvation-Freedom*

Intro

Traditional global progress properties of concurrent programs :

- *Deadlock-Freedom*
- *Starvation-Freedom*

$\Downarrow$ Critical section

$\rightarrow$ Lock-based only

Intro

A more general approach :

- *Non-Blockingness*
- *Wait-Freedom*

Intro

A more general approach :

- *Non-Blockingness*
- *Wait-Freedom*

$\Uparrow$ Abstraction

$\Uparrow$ Extensionality

Intro

A more general approach :

- *Non-Blockingness*
- *Wait-Freedom*

$\Uparrow$ Abstraction

$\Uparrow$ Extensionality

$\Downarrow$ Lack of rigorous semantic basis

Intro

Non-Blockingness

"A data structure is *non-blocking* if it guarantees that *some* process will always be able to complete its pending operation in a finite number of its own steps, regardless of the execution speed of other processes." [Taubenfeld, '06]

Intro

How to formalise:

- "a finite number of its own steps"
- "regardless of the execution speed of other processes"

?

Intro

How to formalise:

- "a finite number of its own steps"
- "regardless of the execution speed of other processes"

$\Rightarrow$ *Fairness*

$\Rightarrow$ *Partial Failures*

Intro

How to formalise:

- "a finite number of its own steps"
- "regardless of the execution speed of other processes"

$\Rightarrow$ *Fairness*

$\Rightarrow$ *Partial Failures*

$\Rightarrow$ π-*calculus*

Intro

AIM :

To provide an **extensional** theory which is **general** enough to cover all the concurrent data structures whose behaviours are representable in the π-calculus.

Index

- π-calculus
- Linear/Affine Types
- Asynchronous fair LTS + partial failures
- Global Progress :
  - Non-blockingness
  - Wait-Freedom
- Case study : Queues
  - Correctness (state space)
  - Behavioural Classification

The Calculus

$$
\begin{array}{rcl}
P & ::= & u\&_{i\in I}\{\mathtt{l}_i(\vec{x}_i).P_i\} \\
 & \mid & \overline{u} \oplus \mathtt{l}\langle\vec{e}\rangle \\
 & \mid & \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \\
 & \mid & P|Q \\
 & \mid & (\boldsymbol{\nu} u)P \\
 & \mid & (\mu X(\vec{x}).P)\langle\vec{e}\rangle \\
 & \mid & X\langle\vec{x}\rangle \\
 & \mid & \mathbf{0}
\end{array}
$$

Reductions

One rule:

$$u\&_{i\in I}\{\mathtt{l}_i(\vec{x}_i).P_i\} \mid \overline{u} \oplus \mathtt{l_j}\langle\vec{e}\rangle \quad \longrightarrow \quad P_j\{\vec{e}/\vec{x}_j\} \quad (j \in I)$$

Closed under the standard structural congruence, $\equiv$.
Note in particular:

$$(\mu X(\vec{x}).P)\langle\vec{e}\rangle \equiv P\{(\mu X(\vec{x}).P)/X\}\{\vec{e}/\vec{x}\}$$

Example

Some simple concurrent data structures :

$$\mathsf{Ref}\langle u, v\rangle \quad \stackrel{\text{def}}{=} \quad u\& \left\{ \begin{array}{l} \mathit{read}(z) : \overline{z}\langle v\rangle \mid \mathsf{Ref}\langle u, v\rangle, \\ \mathit{write}(y, z) : \overline{z} \mid \mathsf{Ref}\langle u, y\rangle \end{array} \right\}$$

$$\mathsf{Ref}^{\mathsf{cas}}\langle u, v\rangle \quad \stackrel{\text{def}}{=} \quad u\& \left\{ \begin{array}{l} \mathit{read}(z) : \overline{z}\langle v\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle u, v\rangle, \\ \mathit{write}(y, z) : \overline{z} \mid \mathsf{Ref}^{\mathsf{cas}}\langle u, y\rangle, \\ \mathit{cas}(x, y, z) : \texttt{if } x = v \texttt{ then } \overline{z}\langle \mathsf{tt}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle u, y\rangle \\ \qquad\qquad\qquad\qquad\quad \texttt{else } \overline{z}\langle \mathsf{ff}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle u, v\rangle \end{array} \right\}$$

Example

Reduction :

$$
\begin{array}{rl}
& \mathsf{Ref}^{\mathsf{cas}}\langle a,0\rangle | (\boldsymbol{\nu} c)(\overline{a} \oplus cas\langle 0,1,c\rangle | c(x).P) \\
\longrightarrow & (\boldsymbol{\nu} c)((\texttt{if}\ 0 = 0\ \texttt{then}\ \overline{c}\langle \mathsf{tt}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle a,1\rangle\ \texttt{else}\ \overline{c}\langle \mathsf{ff}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle a,0\rangle)) \mid c(x).P) \\
\longrightarrow & (\boldsymbol{\nu} c)((\texttt{if}\ \mathsf{tt}\ \texttt{then}\ \overline{c}\langle \mathsf{tt}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle a,1\rangle\ \texttt{else}\ \overline{c}\langle \mathsf{ff}\rangle \mid \mathsf{Ref}^{\mathsf{cas}}\langle a,0\rangle)) \mid c(x).P) \\
\longrightarrow^{*} & \mathsf{Ref}^{\mathsf{cas}}\langle a,1\rangle \mid P\{\mathsf{tt}/x\}
\end{array}
$$

Example

Two different mutex agents:

$$
\begin{aligned}
\mathsf{Mtx}\langle u\rangle &\stackrel{\text{def}}{=} u(x).\overline{x}(h)h.\mathsf{Mtx}\langle u\rangle \\
\mathsf{Mtx}^{\mathsf{spin}}\langle u\rangle &\stackrel{\text{def}}{=} (\boldsymbol{\nu} c)(!u(x).\mu X. \\
&\quad (\mathtt{if}\ \mathtt{cas}(c,0,1)\ \mathtt{then}\ \overline{x}(h)h.\mathtt{CAS}(c,1,0)\ \mathtt{else}\ )\ | \\
&\quad \mathsf{Ref}^{\mathsf{cas}}\langle c,0\rangle)
\end{aligned}
$$

where

$$\mathtt{if}\ \mathtt{cas}(u,v,w)\ \mathtt{then}\ P\ \mathtt{else}\ Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} c)(\overline{u}\oplus cas\langle v,w,c\rangle | c(x).\mathtt{if}\ x\ \mathtt{then}\ P\ \mathtt{else}\ Q)$$

and

$$\mathtt{CAS}(u,v,w) \stackrel{\text{def}}{=} \mathtt{if}\ \mathtt{cas}(u,v,w)\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ \mathbf{0}$$

Types

$$\tau \quad ::= \quad \&^{\mathrm{M}}_{i\in I} l_i(\vec{\tau}_i) \;\mid\; \oplus^{\mathrm{M}}_{i\in I} l_i(\vec{\tau}_i) \;\mid\; \mathsf{int} \;\mid\; \mathsf{bool} \;\mid\; \bot$$

Modalities (as in *Linear Logic*, *Games*, ...):

- L channel can be used "exactly once" (*linear*)
- A channel can be used "at most once" (*affine*)
- L* input end always available and shared by unboundedly many outputs (*unbounded l.*)
- A* input end as above but may be unavailable (*unbounded a.*)

Example

Typings for the previously introduced examples :

1.

$$u : \&^{L*}\{read(\uparrow^L(\mathsf{nat})), write(\mathsf{nat}\uparrow^L())\} \ \vdash \ \mathsf{Ref}\langle u, 3\rangle$$

2.

$$u : \&^{L*}\{read(\uparrow^L(\mathsf{nat})), write(\mathsf{nat}\uparrow^L()), cas(\mathsf{natnat}\uparrow^L(\mathsf{bool})),\} \ \vdash \ \mathsf{Ref}^{\mathsf{cas}}\langle u, 0\rangle$$

3.

$$u : \downarrow^{A*}(\uparrow^A(\downarrow^A())) \ \vdash \ P \qquad (\ P \in \{\mathsf{Mtx}\langle u\rangle, \mathsf{Mtx}^{\mathsf{spin}}\langle u\rangle\}\ )$$

Labelled Transition System

Labels :

$$\ell \quad ::= \quad \tau \mid (\boldsymbol{\nu}\vec{c})a\&l(\vec{v}) \mid (\boldsymbol{\nu}\vec{c})a \oplus l\langle\vec{v}\rangle$$

Untyped transitions :

(Bra)

$$P \xrightarrow{(\boldsymbol{\nu}\vec{c})a\&l\langle\vec{v}\rangle} P|\overline{a} \oplus \mathtt{l}\langle\vec{v}\rangle$$

(Sel)

$$(\boldsymbol{\nu}\vec{c})(P|\overline{a} \oplus \mathtt{l}\langle\vec{v}\rangle) \xrightarrow{(\boldsymbol{\nu}\vec{c})\overline{a}\oplus l\langle\vec{v}\rangle} P$$

Labelled Transition System

Environment transitions :

$$\Gamma,\, a : \&^{L*,A*}\{l_i(\vec{\tau}_i)\}_{i\in I} \stackrel{(\boldsymbol{\nu}\vec{c})a\&l_j\langle\vec{v}_j\rangle}{\longrightarrow} \Gamma \odot \vec{v} : \overline{\vec{\tau}_j},\, a : \&^{L*,A*}\{l_i(\vec{\tau}_i)\}_{i\in I}$$

$$(\Gamma \odot \vec{v} : \overline{\vec{\tau}_j})/\vec{c},\, a : \oplus^{L*,A*}\{l_i(\vec{\tau}_i)\}_{i\in I} \stackrel{(\boldsymbol{\nu}\vec{c})a\oplus l_j\langle\vec{v}_j\rangle}{\longrightarrow} \Gamma,\, a : \oplus^{L*,A*}\{l_i(\vec{\tau}_i)\}_{i\in I}$$

$$\Gamma,\, a : \&^{L,A}\{l_i(\vec{\tau}_i)\}_{i\in I} \stackrel{(\boldsymbol{\nu}\vec{c})a\&l_j\langle\vec{v}_j\rangle}{\longrightarrow} \Gamma \odot \vec{v} : \overline{\vec{\tau}_j},\, a : \perp$$

$$(\Gamma \odot \vec{v} : \overline{\vec{\tau}_j})/\vec{c},\, a : \oplus^{L,A}\{l_i(\vec{\tau}_i)\}_{i\in I} \stackrel{(\boldsymbol{\nu}\vec{c})a\oplus l_j\langle\vec{v}_j\rangle}{\longrightarrow} \Gamma$$

Typed transitions :

$$\Gamma \vdash P \xrightarrow{\ell} \Gamma' \vdash P' \quad \stackrel{\text{def}}{\Leftrightarrow} \quad P \xrightarrow{\ell} P' \wedge \Gamma \xrightarrow{\ell} \Gamma'$$

Bisimilarity

Definition 3.3 (bisimilarity) A typed relation $\mathcal{R}$ is a *weak bisimulation* or often *bisimulation* when, for each $\Gamma \vdash P\mathcal{R}Q$, we have: $P \xrightarrow{\ell} P'$ implies $Q \xRightarrow{\hat{\ell}} Q'$ s.t. $P'\mathcal{R}Q'$, and the symmetric case. The maximum bisimulation is written $\approx$.

Proposition 3.4 $\approx$ is a typed congruence.

Fairness

Definition 3.5 (Fairness) A maximal transition sequence Φ from closed $\Gamma \vdash P$ is *fair* if no subject is infinitely often enabled in Φ.

Fairness

Definition 3.5 (Fairness) A maximal transition sequence Φ from closed $\Gamma \vdash P$ is *fair* if no subject is infinitely often enabled in Φ.

Let $P = !a.(\bar{b}|\overline{a})|\overline{a}$ and $Q = \mathsf{Ref}\langle r, 3\rangle|\overline{r} \oplus \mathtt{read}\langle c\rangle$.

Then $P|Q$ admits an infinite unfair transition sequence.

Fairness

Definition 3.5 (Fairness) A maximal transition sequence Φ from closed $\Gamma \vdash P$ is *fair* if no subject is infinitely often enabled in Φ.

Let $P = !a.(\bar{b}|\overline{a})|\overline{a}$ and $Q = \mathsf{Ref}\langle r, 3\rangle|\overline{r} \oplus \mathtt{read}\langle c\rangle$.

Then $P|Q$ admits an infinite unfair transition sequence.

Fairness induces a *fair pre-order* $\precapprox_{\mathsf{fair}}$, where:

$$P \precapprox_{\mathsf{fair}} Q \iff P \approx Q \ \wedge \ \mathsf{WFT}(P) \supseteq \mathsf{WFT}(Q)$$

Partial Failure

We first augment the dynamics with *failing reductions*:

$$\overline{u} \oplus^{\mathrm{M}} \mathtt{l_j}\langle \vec{e} \rangle \;\longrightarrow\; \mathbf{0} \quad (\mathrm{M} \neq L) \qquad\qquad \mathtt{if}\ v\ \mathtt{then}\ P\ \mathtt{else}\ Q \;\longrightarrow\; \mathbf{0}$$

Then we augment the τ-transition accordingly.

NOTE :

Only affine outputs may fail!

$$\Downarrow$$

Linearity $\Rightarrow$ *Atomicity*

Global Progress

Definition 3.11 (Resilience) Let $\Gamma \vdash P$ such that for all $\mathsf{wft}(\Phi) \in \mathsf{WFT}(P)$, the set $\mathtt{blocked}(\Phi)$ is finite. Then we say that $\Gamma \vdash P$ is *resilient*.

Definition 3.12 (NB/WF) A closed process $\Gamma \vdash P$ is:

1. *non-blocking* (NB) when it is resilient and, for any $\Phi \in \mathsf{FT}(P)$ s.t. $\Delta \vdash Q$ is in Φ and $\mathtt{allowed}(\Delta) \setminus \mathtt{blocked}(\Phi) \neq \emptyset$, some affine output occurs in Φ.

2. *wait-free* (WF) when it is resilient and, for any $\Phi \in \mathsf{FT}(P)$ s.t. $\Delta \vdash Q$ is in Φ and $c \in \mathtt{allowed}(\Delta) \setminus \mathtt{blocked}(\Phi)$, an output at c occurs in Φ.

Weak Global Progress

Let **WNB** be as **NB** but without failures.

Let **WWF** be as **WF** but without failures.

Then $\mathbf{WF} \subsetneq \mathbf{NB} \cap \mathbf{WWF}$ and $\mathbf{NB} \cup \mathbf{WWF} \subsetneq \mathbf{WNB}$.

Abstract Queue Specification

Abstract Queue:

$$\mathsf{AQ}(r, \langle \mathsf{Rs};\ \mathsf{Vs};\ \mathsf{As} \rangle)$$

Abstract Queue Specification

Abstract Queue:

$$\mathsf{AQ}(r, \langle \mathsf{Rs};\ \mathsf{Vs};\ \mathsf{As} \rangle)$$

Example:

$$\mathsf{AQ}(r, \langle \{\mathtt{enq}(6, g_1), \mathtt{deq}(g_2)\}, 2\cdot 3\cdot 1, \{\overline{g_3}\langle 5\rangle\}\rangle)$$

State Space Abstraction

Abstract Queue:

$$\mathsf{AQ}(r, \langle \mathsf{Rs};\ \mathsf{Vs};\ \mathsf{As}\rangle)$$

State Transitions

$$\mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs},\ \mathsf{As}\rangle) \xrightarrow{r\&\mathtt{enq}(v,g)} \mathsf{AQ}(r, \langle \mathsf{Rs} \uplus \mathtt{enq}(v,g),\ \mathsf{Vs},\ \mathsf{As}\rangle)$$

$$\mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs},\ \mathsf{As}\rangle) \xrightarrow{r\&\mathtt{deq}(g)} \mathsf{AQ}(r, \langle \mathsf{Rs} \uplus \mathtt{deq}(g),\ \mathsf{Vs},\ \mathsf{As}\rangle)$$

$$\mathsf{AQ}(r, \langle \mathsf{Rs} \uplus \mathtt{enq}(v,g),\ \mathsf{Vs},\ \mathsf{As}\rangle) \xrightarrow{\tau} \mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs}\cdot v,\ \mathsf{As} \uplus \overline{g}\rangle)$$

$$\mathsf{AQ}(r, \langle \mathsf{Rs} \uplus \mathtt{deq}(g),\ v\cdot\mathsf{Vs},\ \mathsf{As}\rangle) \xrightarrow{\tau} \mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs},\ \mathsf{As} \uplus \{\overline{g}\langle v\rangle\}\rangle)$$

$$\mathsf{AQ}(r, \langle \mathsf{Rs} \uplus \mathtt{deq}(g),\ ,\ \mathsf{As}\rangle) \xrightarrow{\tau} \mathsf{AQ}(r, \langle \mathsf{Rs},\ ,\ \mathsf{As} \uplus \{\overline{g}\langle KO\rangle\}\rangle)$$

$$\mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs},\ \mathsf{As} \uplus \{\overline{g}\langle v\rangle\}\rangle) \xrightarrow{\overline{g}\langle v\rangle} \mathsf{AQ}(r, \langle \mathsf{Rs},\ \mathsf{Vs},\ \mathsf{As}\rangle)$$

cas-based Queue in π

Auxiliary data structures :

$$\begin{aligned}\mathsf{Node}(r, v, ptr) &\stackrel{\text{def}}{=} \mathsf{Ref}\langle r, \langle v, ptr\rangle\rangle \\ \mathsf{Ptr}(r, nd, ctr) &\stackrel{\text{def}}{=} \mathsf{Ref}^{\mathsf{cas}}\langle r, \langle nd, ctr\rangle\rangle\end{aligned}$$

Shortened forms :

$$x \triangleleft read(\vec{y}).P \stackrel{\text{def}}{=} (\nu c)(\overline{x} \oplus \mathtt{read}\langle c\rangle | c(\vec{y}).P)$$

We use ad-hoc names when we only want to keep a projection of the contents ($getPtr$, $getVal$, $\dots$).

cas-based Queue in π

$$
\begin{aligned}
\mathsf{CQemp}(r) \quad &\stackrel{\text{def}}{=} \quad (\nu h)(\nu t)(\nu s)(\mathsf{CQ}(r,h,t) \mid \mathsf{Ptr}(h,s,0) \mid \\
&\qquad\qquad\qquad \mathsf{Ptr}(t,s,0) \mid \mathsf{Node}(s,0,\mathsf{null}))
\end{aligned}
$$

$$
\mathsf{CQ}(r,h,t) \quad \stackrel{\text{def}}{=} \quad r\& \left\{ \begin{array}{l} enqueue(x,u): \\ \quad (\mathsf{CQ}(r,h,t) \mid P_{enq}(x,t)) \\ \\ dequeue(u): \\ \quad (\mathsf{CQ}(r,h,t) \mid P_{deq}(h,t)) \end{array} \right\}
$$

cas-based Queue in π

$$
\begin{array}{l}
P_{enq}(x, tail) = \\
\quad \mathsf{NullPtr}(nlPtr) \mid \mathsf{Node}(node, v, nlPtr) \mid \\
\quad (\mu X_{tag}(u'). \\
\qquad tail \triangleleft read(last, ctrT). \quad last \triangleleft getPtr(tPtr). \quad tPtr \triangleleft read(next, ctr). \\
\qquad \texttt{if } (next = null) \texttt{ then} \\
\qquad\quad \texttt{if cas}(tPtr, \langle next, ctr\rangle, \langle node, ctr+1\rangle) \texttt{ then} \\
\qquad\qquad \texttt{if cas}(tail, \langle last, ctrT\rangle, \langle node, ctrT+1\rangle) \texttt{ then } \overline{u'} \texttt{ else } \overline{u'} \\
\qquad\quad \texttt{else } X_{tag}\langle u'\rangle) \\
\qquad \texttt{else} \\
\qquad\quad \texttt{if cas}(tail, \langle last, ctrT\rangle, \langle next, ctrT+1\rangle) \texttt{ then } X_{tag}\langle u'\rangle \texttt{ else } X_{tag}\langle u'\rangle) \\
\qquad \texttt{endif})\langle u\rangle
\end{array}
$$

cas-based Queue in π

$$
\begin{array}{l}
P_{deq}(head, tail) = (\mu X_{tag}(u'). \\
\quad head \lhd read(hNdRef, ctrH). \qquad tail \lhd read(tNdRef, ctrT). \\
\quad hNdRef \lhd getPtr(hNdPtr). \qquad hNdPtr \lhd getNxt(next). \\
\quad \texttt{if } (hNdRef = tNdRef) \texttt{ then} \\
\quad\quad \texttt{if } (next = null) \texttt{ then } \overline{u'}\langle null \rangle \texttt{ else} \\
\quad\quad\quad \texttt{if } (\mathsf{cas}(tail, \langle tNdRef, ctrT \rangle, \langle next, ctrT + 1 \rangle)) \texttt{ then } X_{tag}\langle u' \rangle \\
\quad\quad\quad \texttt{else } X_{tag}\langle u' \rangle) \\
\quad \texttt{else } next \lhd getVal(x). \\
\quad\quad \texttt{if } (\mathsf{cas}(head, \langle hNdRef, ctrH \rangle, \langle next, ctr + 1 \rangle)) \texttt{ then } \overline{u'}\langle x \rangle \\
\quad\quad \texttt{else } X_{tag}\langle u' \rangle)\langle u \rangle
\end{array}
$$

Lock-Based Queue in π

$$\mathsf{LQemp}(r) \stackrel{\text{def}}{=} (\nu u)(\nu h)(\nu t)(\nu s)(\mathsf{LQ}(r,h,t)|\mathsf{Mtx}\langle u\rangle| \\ \mathsf{LPtr}(h,s)|\mathsf{LPtr}(t,s)|\mathsf{LENode}(s,0))$$

$$\mathsf{LQ}(r,h,t,l) \stackrel{\text{def}}{=} r\&\left\{\begin{array}{l} \mathtt{enqueue}(v,u) : \mathsf{LQ}(r,h,t,l)| \\ \quad (\bar{l}(g)g(y).P^{lck}_{enq}(v,t,c')|c'.(\bar{y}|\bar{u})), \\ \mathtt{dequeue}(u) : \mathsf{LQ}(r,h,t,l)| \\ \quad (\bar{l}(g)g(y).P^{lck}_{deq}(h,t,c')|c'.(\bar{y}|\bar{u})) \end{array}\right\}$$

Correctness

- **state space abstraction** (*correctness by bisimilarity*)
- **molecular transitions** ($\rightsquigarrow$*atomic operation*)
- **commit event** (*i.e.* cas *on successor pointer*)
- **normal form** (*each thread is either:*
    1. *ready to commit,*
    2. *or ready to output*)
- **normalisation through linearisation** (*local permutations*)

Global Progress

Proposition 4.6 Let $\Phi : \mathsf{CQemp}(r) \longrightarrow^* P$ be a queue process. Then an output is blocked in Φ iff its thread fails in Φ.

Proposition 4.8 $\Gamma_Q \vdash \mathsf{CQemp}(r)$ is non-blocking.

Behavioural Classification

$\Gamma_Q \vdash \mathsf{LQemp}(r)$ *is* blocking (it is weakly wait-free).

$\Gamma_Q \vdash \mathsf{LQemp}(r)$ has more fair sequences

Theorem 4.11 $\mathsf{AQ}(r,\varepsilon) \approx \mathsf{LQemp}(r) \precnapprox_{\mathsf{fair}} \mathsf{CQemp}(r)$

Results and Future Works

- Fairness
- Fine-grained analysis
- Generality
- Extensionality
- State space abstraction

$\Rightarrow$ Automated verification tools

$\Rightarrow$ Encoding from imperative languages

References

[1] D. Cacciagrano, F. Corradini and C. Palamidessi. Fair π. Proc. *EXPRESS'06. ENTCS*, 175:3–26. Elsevier Science, 2007.

[2] M. M. Michael and M. L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. *PODC-15*. 267–275. ACM, 1996.

[3] R. Milner and J. G. Parrow and D. J. Walker, A calculus of Mobile Processes, Information and Computation 100(1), 1–77, 1992.

[4] G. Taubenfeld. *Synchronization Algorithms and Concurrent Programming*. Pearson–Prentice Hall, 2006.

Thank you for your attention!