Distributed synthesis, well-quasi-orders, and memory for games

Corto Mascle
MPI-SWS Kaiserslautern

Joint work with Anca Muscholl and Igor Walukiewicz

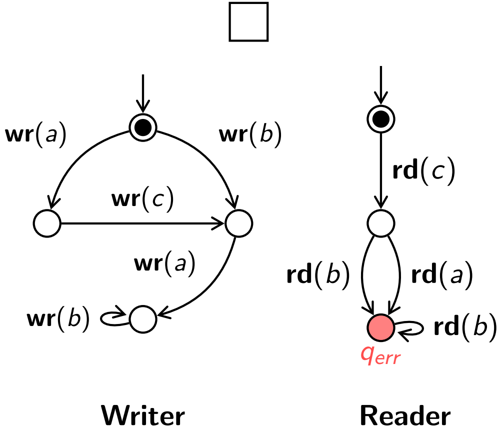
Highlights 2025

Asynchronous distributed systems

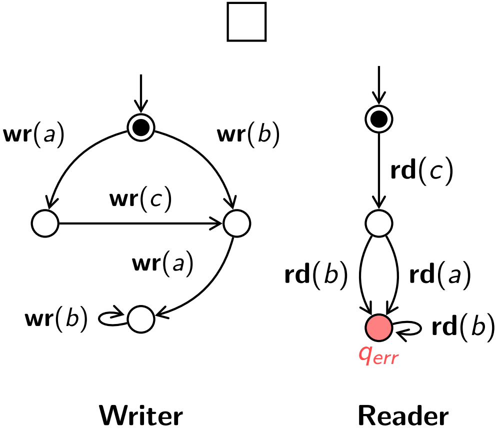
- Several processes running asynchronously

Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps



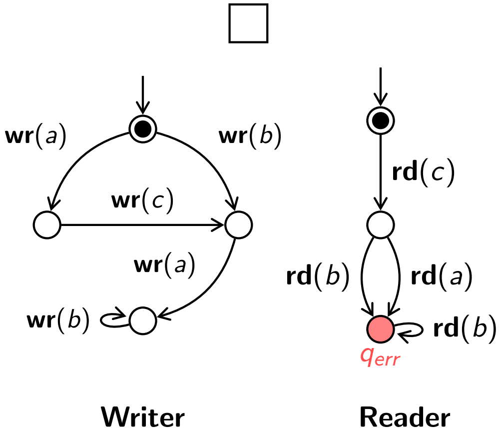
Asynchronous distributed systems



- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...

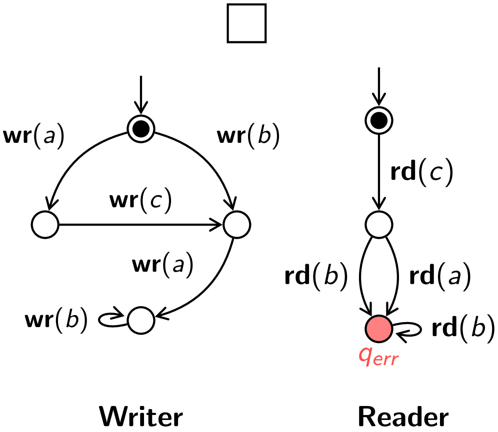
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



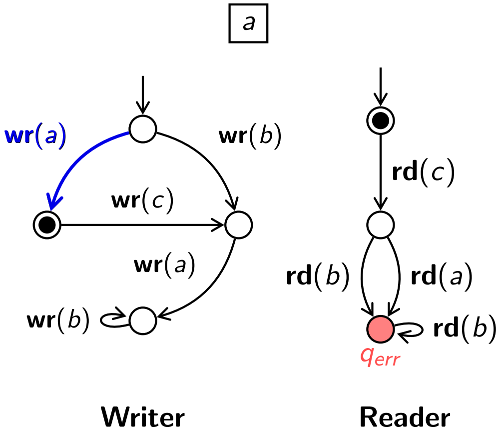
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



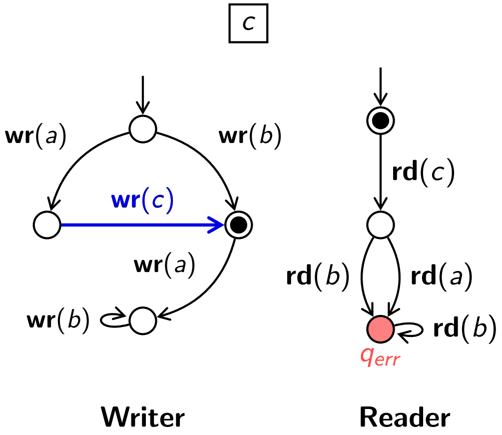
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



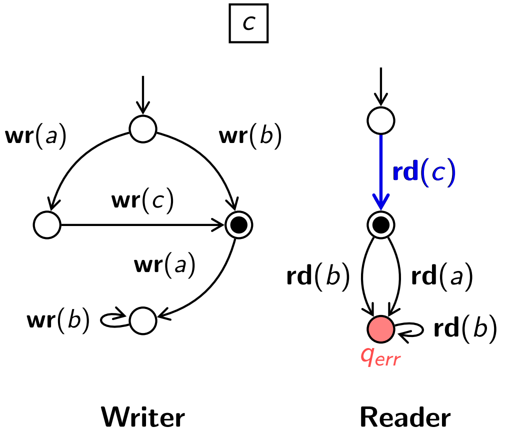
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



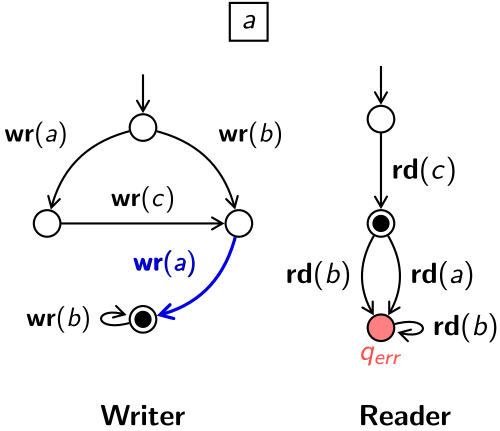
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



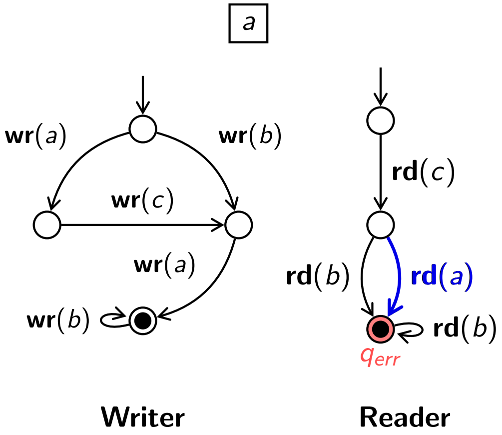
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



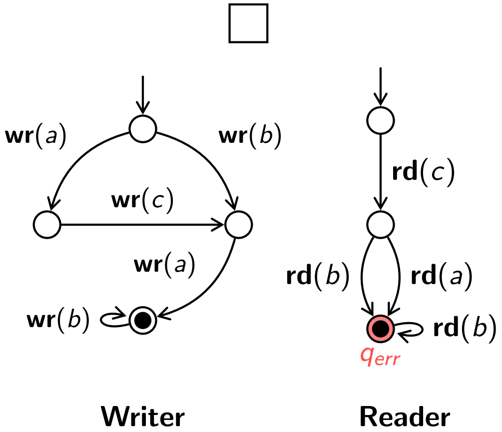
Asynchronous distributed systems

- Several processes running asynchronously
- Execution = sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}



Asynchronous distributed systems

- Several processes running asynchronously
- Execution $=$ sequence of local steps
- Constraint on scheduling: Shared memory, locks, broadcast, rendez-vous, token-passing, ...
- Specification: avoid q_{err}

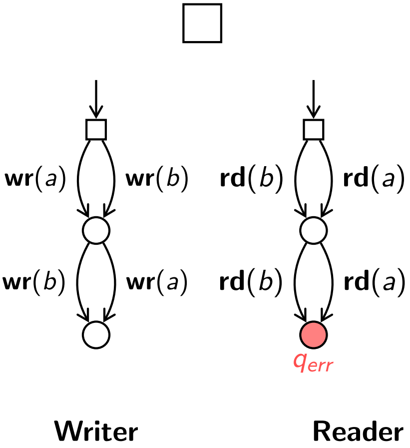


Local runs of writer and reader can be combined whenever the sequence read by Reader is a **subword** *of the sequence written by Writer.*

Ex: *bab* is a subword of $a\underline{b}b\underline{ab}a$.

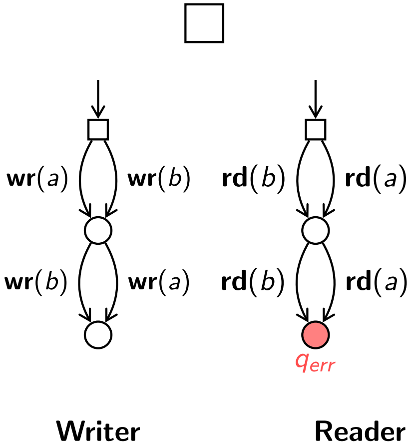
Distributed synthesis

- Each process has Controller and Environment states.



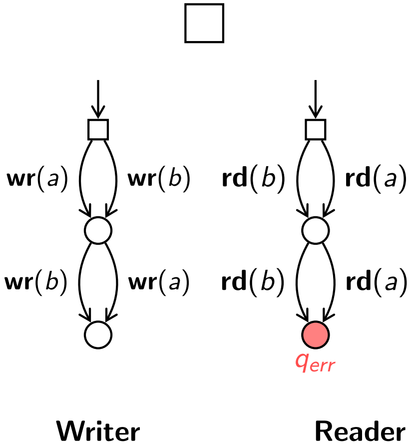
Distributed synthesis

- Each process has Controller and Environment states.
- One *local strategy* for each process: σ_w, σ_r



Distributed synthesis

- Each process has Controller and Environment states.
- One *local strategy* for each process: σ_w, σ_r
- Local strategy = function $\sigma_w : \mathit{Paths}_w \to \mathit{Transitions}_w$ that chooses the next transition from controllable states based on past transitions.



Invariants

Lemma

Local strategies (σ_w, σ_r) *successfully avoid* q_{err}

$$\Leftrightarrow$$

Invariants

Lemma

Local strategies (σ_w, σ_r) *successfully avoid* q_{err}

$$\Leftrightarrow$$

there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. *Every local* σ_w*-run of the writer writes a sequence in* $\mathcal{I}$
2. *Every local* σ_r*-run of the reader reaching* q_{err} *reads a sequence* $\notin \mathcal{I}$
3. $\mathcal{I}$ *is subword-closed*

Invariants

Lemma

Local strategies (σ_w, σ_r) *successfully avoid* q_{err}

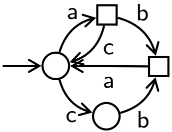
$$\Leftrightarrow$$

there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. *Every local* σ_w*-run of the writer writes a sequence in* $\mathcal{I}$
2. *Every local* σ_r*-run of the reader reaching* q_{err} *reads a sequence* $\notin \mathcal{I}$
3. $\mathcal{I}$ *is subword-closed*

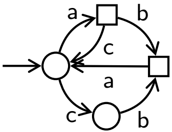
- $\mathcal{I}$ a priori arbitrarily large
- If we were given $\mathcal{I}$ $\rightarrow$ reduces to *regular two-player safety game*

Objective-independent memory

Arena $=$ 

Safety objective $= W \subseteq \{a, b, c\}^*$
Strategy is *winning* if all runs stay in W

Objective-independent memory

Arena $=$ 

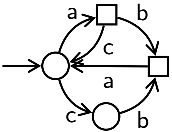
Safety objective $= W \subseteq \{a, b, c\}^*$
Strategy is *winning* if all runs stay in W

Theorem

Let W be a subword-closed objective and $\mathcal{G}$ an arena.
If Controller has a winning strategy for W in $\mathcal{G}$, then she has one with memory $\leq |\mathcal{G}|!$.

Objective-independent memory

Arena =

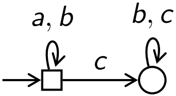


Safety objective = $W \subseteq \{a, b, c\}^*$

Strategy is *winning* if all runs stay in W

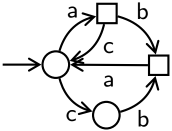
Theorem

Let W be a subword-closed objective and $\mathcal{G}$ an arena.
If Controller has a winning strategy for W in $\mathcal{G}$, then she has one with memory $\leq |\mathcal{G}|!$.



For all W subword-closed,
we need memory $\leq 5!$

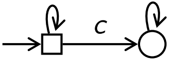
Objective-independent memory

Arena $=$ 

Safety objective $= W \subseteq \{a, b, c\}^*$
Strategy is *winning* if all runs stay in W

Theorem

Let W be a subword-closed objective and $\mathcal{G}$ an arena.
If Controller has a winning strategy for W in $\mathcal{G}$, then she has one with memory $\leq |\mathcal{G}|!$.



For all W subword-closed, we need memory ≤ 5!

$\mathcal{C} \subseteq 2^{\Sigma^*}$ has *objective-independent memory* if $\exists f : \mathbb{N} \to \mathbb{N}$, $\forall$ arena $\mathcal{G}$ and $W \in \mathcal{C}$, if Controller has a winning strategy for W in $\mathcal{G}$, then she has one with memory $\leq f(|\mathcal{G}|)$.

Bounding strategies

Lemma

There are local strategies (σ_w, σ_r) to avoid q_{err} iff there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. $\mathcal{I}$ *is subword-closed*
2. *Every local σ_w-run of the writer writes a sequence in $\mathcal{I}$*
3. *Every local σ_r-run of the reader reaching q_{err} reads a sequence $\notin \mathcal{I}$*

Winning strategy $\Rightarrow$ Subword-closed $\mathcal{I}$

Bounding strategies

Lemma

There are local strategies (σ_w, σ_r) to avoid q_{err} iff
there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. $\mathcal{I}$ *is subword-closed*
2. *Every local σ_w-run of the writer writes a sequence in $\mathcal{I}$*
3. *Every local σ_r-run of the reader reaching q_{err} reads a sequence $\notin \mathcal{I}$*

Winning strategy $\Rightarrow$ Subword-closed $\mathcal{I}$
$\Rightarrow$ Bound Writer's strat.

Bounding strategies

Lemma

There are local strategies (σ_w, σ_r) *to avoid* q_{err} *iff*
there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. $\mathcal{I}$ *is subword-closed*
2. *Every local* σ_w*-run of the writer writes a sequence in* $\mathcal{I}$
3. *Every local* σ_r*-run of the reader reaching* q_{err} *reads a sequence* $\notin \mathcal{I}$

Winning strategy $\Rightarrow$ Subword-closed $\mathcal{I}$
$\Rightarrow$ Bound Writer's strat.
$\Rightarrow$ Bound $\mathcal{I}$

Bounding strategies

Lemma

There are local strategies (σ_w, σ_r) to avoid q_{err} iff there exists an **invariant** $\mathcal{I} \subseteq \Sigma^*$ *such that*

1. $\mathcal{I}$ *is subword-closed*
2. *Every local σ_w-run of the writer writes a sequence in $\mathcal{I}$*
3. *Every local σ_r-run of the reader reaching q_{err} reads a sequence $\notin \mathcal{I}$*

Winning strategy $\Rightarrow$ Subword-closed $\mathcal{I}$
$\Rightarrow$ Bound Writer's strat.
$\Rightarrow$ Bound $\mathcal{I}$
$\Rightarrow$ Bound Reader's strat.

Bounding strategies

Theorem

The distributed synthesis problem is NEXPTIME-complete for reader-writer systems.

Bounding strategies

Theorem

The distributed synthesis problem is NEXPTIME-complete for reader-writer systems.

"Easy" extension: everyone can read and write, but the process writing can change $\leq K$ times.

Theorem

The distributed synthesis problem is NEXPTIME-complete for shared-memory systems with bounded writer switches.

Burning questions

Which classes of languages have objective-independent memory?

Which communication primitives yield a well quasi-order on local executions?

Burning questions

Which classes of languages have objective-independent memory?

Which communication primitives yield a well quasi-order on local executions?

Many thanks to Antonio Casares, Pierre Ohlmann, Isa Vialard and people from Autobóz '25.

Burning questions

Which classes of languages have objective-independent memory?

Which communication primitives yield a well quasi-order on local executions?

Many thanks to Antonio Casares, Pierre Ohlmann, Isa Vialard and people from Autobóz '25.

Thank you for your attention!